

第 13 章

Nginx 的标准 HTTP 模块

本章介绍的 HTTP 模块会在编译 Nginx 时自动编译进来，除非使用 `configure` 命令禁止编译这些模块。

在本章的指令介绍中，指令的“使用环境”是该指令可以在 Nginx 配置文件中使用的位置，例如使用环境为“`http, server, location`”，表示该指令可以在以下位置使用：

`http { }`大括号内；`server { }`大括号内；`location { }`大括号内。

13.1 HTTP 的核心模块

13.1.1 alias 指令

语法：`alias file-path|directory-path;`

默认值：`no`

使用环境：`location`

该指令用于在 URL 和文件系统路径之间实现映射。它与 `root` 指令类似，但是网页文件的 `root` 根目录不会改变，改变的只是请求 URL 的文件系统路径。

示例如下：

```
location /i/ { alias /spool/w3/images/; }
```

在示例中，访问 URL 地址 “/i/top.gif” 会返回文件 “/spool/w3/images/top.gif”。

在被替换的路径中，可以使用变量。在含有正则表达式的 location 中，不能使用 alias 指令，如果你想使用，可以使用 rewrite 和 root 指令的组合来实现。

13.1.2 client_body_in_file_only 指令

语法：client_body_in_file_only on|off

默认值：off

使用环境：http, server, location

该指令允许将一个客户端的请求内容记录到一个文件中，该文件在请求完成后不会被删除。在内置 Perl 中，该指令可以用于调试 \$r->request_body_file 方法。

13.1.3 client_body_in_single_buffer 指令

语法：client_body_in_single_buffer on|off

默认值：off

使用环境：http, server, location

该指令（0.7.58 以上版本支持）指定是否保持整个内容在一个单一的客户端请求缓冲区中。该指令在使用变量 \$request_body 时被推荐使用。

13.1.4 client_body_buffer_size 指令

语法：client_body_buffer_size the_size

默认值：8k/16k

使用环境：http, server, location

示例：client_body_buffer_size 128k;

该指令指定客户端请求内容的缓冲区大小。如果客户端请求内容大于缓冲区，整个请求内容或部分请求内容将被写入临时文件。缓冲区默认大小相当于网页大小的两倍，为 8k 或 16k。

13.1.5 client_body_temp_path 指令

语法: `client_body_temp_path dir-path [level1 [level2 [level3]`

默认值: `client_body_temp`

使用环境: `http, server, location`

该指令用于指定存放请求内容临时文件的目录。缓存目录最多支持 3 层子目录。

示例:

```
client_body_temp_path /spool/nginx/client_temp 1 2;
```

以上示例的目录结构类似:

```
/spool/nginx/client_temp/7/45/00000123457
```

13.1.6 client_body_timeout 指令

语法: `client_body_timeout time`

默认值: `60`

使用环境: `http, server, location`

该指令用于设置读取客户端请求内容的超时时间。如果超过该指令设置的时间, Nginx 将返回 “Request time out” 错误信息 (HTTP 状态码为 408)。

13.1.7 client_header_buffer_size 指令

语法: `client_header_buffer_size size`

默认值: `1k`

使用环境: `http, server`

该指令用于设置客户端请求的 Header 头缓冲区大小。对绝大多数请求来说, 1KB 大小的 Header 头缓冲区已经足够, 但是对于 Cookie 内容较大的请求来说, 可能不够用, 你可以加大该值。

13.1.8 client_header_timeout 指令

语法: `client_header_timeout time`



默认值: 60

使用环境: http, server

该指令用于设置读取客户端请求 Header 头信息的超时时间。如果超过该指令设置的时间, Nginx 将返回 “Request time out” 错误信息 (HTTP 状态码为 408)。

13.1.9 client_max_body_size 指令

语法: client_max_body_size size

默认值: client_max_body_size 1m

使用环境: http, server, location

示例: client_max_body_size 300m;

该指令用于设置允许接受的客户端请求内容的最大值, 即客户端请求 Header 头信息中设置的 Content-Length 的最大值。如果超过该指令设置的最大值, Nginx 将返回 “Request Entity Too Large” 错误信息 (HTTP 状态码为 413)。当默认值为 1MB 时, 如果 Nginx 服务器提供上传 1MB 以上的大文件等操作, 则要加大该值。

13.1.10 default_type 指令

语法: default_type MIME-type

默认值: default_type text/plain

使用环境: http, server, location

MIME-type 是用来告诉浏览器请求的文件媒体类型的, 例如 MIME-type 名 text/plain 表示该文件为文本文件, text/html 表示该文件为 HTML 网页文件。如果 Nginx 无法识别该文件属于何种 MIME-type 类型, 则将该文件标记为 default_type 指令设置的 MIME-type。

例如:

```
location = /proxy.pac { default_type application/x-ns-proxy-autoconfig; }
location = /wpad.dat { rewrite . /proxy.pac; default_type
application/x-ns-proxy-autoconfig; }
```

13.1.11 directio 指令

语法: directio [sizeloff]

默认值: `directio off`

使用环境: `http, server, location`

该指令用于设置一个文件大小，当读取的文件超过该大小时，将使用 `O_DIRECT` 标签（FreeBSD, Linux）、`F_NOCACHE` 标签（Mac OS X）或 `directio()` 函数（Solaris）来读取该文件。打开文件时如果用 `O_DIRECT`，系统就不会使用 `buffer` 缓冲区，而直接通过 `DMA` 读取文件。`DMA` 的英文拼写是“Direct Memory Access”，汉语的意思就是直接内存访问，是一种不经过 `CPU` 而直接从内存存取数据的数据交换模式。

`directio` 指令将使 `sendfile` 功能失效。对于大文件来说，使用该指令是有益的。在以下的示例中，设置了对 4MB 以上的文件使用 `O_DIRECT`、`F_NOCACHE` 标签或 `directio()` 函数读取：

```
directio 4m;
```

13.1.12 error_page 指令

语法: `error_page code [code...] [=|answer-code] uri`

默认值: `no`

使用环境: `http, server, location, if in location`

该指令用于设置如果出现指定的 `HTTP` 错误状态码，则返回给客户端显示的对应 `URI` 地址。

示例如下：

```
error_page 404 /404.html;
```

如果遇到 404 错误状态码（客户端请求的页面不存在），则显示指定的 `/404.html` 文件内容给客户端。注意 `/404.html` 文件大小不能超过 512 字节，否则 Internet Explorer 浏览器会默认为其错误页面，而不是指定的 `/404.html` 页面。

```
error_page 502 503 504 /50x.html;
```

如果遇到 502、503、504 错误状态码（服务器错误），则显示指定的 `/50x.html` 文件内容给客户端。

```
error_page 403 http://example.com/forbidden.html;
```

如果遇到 403 错误状态码（服务器已经理解请求，但是拒绝执行它），则跳转到 `URL` 地址 `http://example.com/forbidden.html`。

此外，还可以更改 `HTTP` 的错误响应码为别的响应码，以便某些客户端浏览器能够支持，例如：

```
error_page 404 =200 /empty.gif;
```

如果遇到 404 错误状态码（客户端请求的页面不存在），将状态码改为 200，并显示指定的

/empty.gif 文件内容给客户端。

如果响应错误代码的页面是 PHP 等 FastCGI 程序，则最好在 `error_page` 中加上=号。

```
error_page 404 = /404.php;
```

如果你想返回原样的错误状态码，则去掉 `error_page` 指令中的=号。

```
error_page 404 /404.php;
```

13.1.13 if_modified_since 指令

语法: `if_modified_since [off|exact|before]`

默认值: `if_modified_since exact`

使用环境: `http, server, location`

`if-modified-since` 是由客户端浏览器往 Nginx 服务器发送的头信息。当再次请求本地存在的缓存页面时，客户端浏览器会通过 `if-modified-since` 头信息将先前 Nginx 服务器端发过来的 `Last-Modified` 最后修改时间戳发送回去，这是为了让 Nginx 服务器端进行验证，通过这个时间戳判断客户端的页面是否是最新的。如果不是最新的，则返回新的内容；如果是最新的，则返回 304 状态码告诉客户端浏览器其本地缓存的页面是最新的，于是浏览器就可以直接从本地加载页面，这样在网络上传输的数据就会大大减少，从而可加快页面的显示速度，同时也减轻服务器的负担。

`if_modified_since` 指令 (Nginx 0.7.24 以上版本支持) 用于设置如何去比较客户端请求 Header 头信息中的 “`if-modified-since`” 文件的修改时间。

- `off`——不检查 “`if-modified-since`” 请求头；
- `exact`——时间完全符合；
- `before`——文件修改时间应该早于 “`if-modified-since`” 请求头中的时间。

13.1.14 index 指令

语法: `index file [file...]`

默认值: `index index.html`

使用环境: `http, server, location`

该指令用于设置 Nginx 的默认首页文件。

例 1: 在文件名中可以使用变量，并且最后的默认首页文件可以使用绝对路径 (即 `/index.html`):

```
index index.$geo.html index.0.html /index.html;
```

例 2:

```
index index.shtml index.html index.htm index.php default.html;
```

如果某个目录下没有指定的默认首页文件,则可以开启以下指令,显示当前目录的文件列表:

```
autoindex on;
```

13.1.15 internal 指令

语法: internal

默认值: no

使用环境: location

该指令用于设置某个 location 路径只能在 Nginx 内部使用,外部无法访问。

例 1:

```
error_page 404 /404.html;  
location /404.html { internal; }
```

例 2:

```
location / {  
    root /var/www/html;  
    error_page 404 @40x;  
}  
location @40x {  
    root /var/www/errors/40x.html;  
}
```

13.1.16 keepalive_timeout 指令

语法: keepalive_timeout [time]

默认值: keepalive_timeout 75

使用环境: http, server, location

keep-alive 功能可使客户端到服务器端的连接持续有效,当出现对服务器的后继请求时,keep-alive 功能可避免建立或重新建立连接。市场上的大部分 Web 服务器,包括 IIS、Apache、Nginx、Lighttpd,都支持 HTTP keep-alive。对于提供静态内容的网站来说,这个功能通常很有用。但是,对于负担较重的网站来说,这里存在另外一个问题:虽然为客户保留打开的连接有

一定的好处，但它同样影响了性能，因为在暂停期间，本来可以释放的资源仍旧被占用。当 Web 服务器和应用服务器在同一台机器上运行时，keep-alive 功能对资源利用的影响尤其突出。此功能为 HTTP 1.1 预设的功能，HTTP 1.0 加上 keep-alive header 也可以提供 HTTP 的持续作用功能。

该指令用于设置 keep-alive 连接超时时间，使用该指令第二个参数设置的时间（秒）将显示在输出的 Header 头 keep-alive: timeout=time 中。不过，一些浏览器不支持 keep-alive。

另外，在 IE 浏览器中，浏览器自身限制 keepalive 时间为 60 秒。无论是客户端浏览器（Internet Explorer）还是 Web 服务器都具有较低的 keepalive 值，这些都将是限制因素。例如，如果客户端的超时值是两分钟，而 Web 服务器的超时值是一分钟，则最大超时值是一分钟。客户端或服务器都可以是限制因素。

13.1.17 keepalive_requests 指令

语法：keepalive_requests n

默认值：keepalive_requests 100

使用环境：http, server, location

设置一个 keep-alive 连接使用的次数。一次请求结束后，如果该连接使用的次数没有超过 keepalive_requests 指令设置的请求次数，则服务器并不立即主动断开连接，而是直到达到 keepalive_timeout 指令设置的时间，才关闭连接。

13.1.18 large_client_header_buffers 指令

语法：large_client_header_buffers number size

默认值：large_client_header_buffers 4 4k/8k

使用环境：http, server

该指令用于设置客户端请求的 Header 头缓冲区大小，默认值为 4KB。客户端请求行不能超过 large_client_header_buffers 指令设置的值，客户端请求的 Header 头信息不能大于 large_client_header_buffers 指令设置的缓冲区大小，否则会报“Request URI too large”（414）或“Bad request”（400）错误。如果客户端的 Cookie 信息较大，则须增加缓冲区大小，示例如下：

```
client_header_buffer_size 128k;  
large_client_header_buffers 4 128k;
```


13.1.19 limit_except 指令

语法: `limit_except methods {...}`

默认值: `no`

使用环境: `location`

该指令用于限制 HTTP 方法访问 `location` 中的内容, 示例如下:

```
limit_except GET {  
    allow 192.168.1.0/32;  
    deny all;  
}
```

13.1.20 limit_rate 指令

语法: `limit_rate speed`

默认值: `no`

使用环境: `http, server, location, if in location`

该指令主要用来限速, 限速单位是“字节数/秒”, 一般在提供 HTTP 下载等应用中会用到该指令。限速只对一个连接起效, 如果客户端开启两个连接下载, 下载的速度将是限速值的两倍。

例 1: 限制每个连接的下载速度为 100KB/秒:

```
limit_rate 100k;
```

例 2: 在特定条件下开启限速功能:

```
server {  
    if ($slow)  
    {  
        set $limit_rate 4k;  
    }  
}
```

13.1.21 limit_rate_after 指令

语法: `limit_rate_after time`

默认值: `limit_rate_after 1m`

使用环境: `http, server, location, if in location`

该指令可以设置一个字节数（例如 1MB），下载的字节数大于该值后，`limit_rate` 指令的限速功能将起效。对于 MP3 在线播放、HTTP 方式的 Flash FLV 视频点播等应用，使用该指令将会起到不错的效果。

例：下载的文件字节数超过 1MB 后，限速为 100KB/秒。

```
limit_rate_after 1m;  
limit_rate 100k;
```

13.1.22 listen 指令

语法：listen address:port [default [backlog=num | rcvbuf=size | sndbuf=size | accept_filter=filter | deferred | bind | ssl]

默认值：listen 80

使用环境：server

该指令用于设置虚拟主机监听的服务器地址和端口号。你可以同时设置服务器地址和端口号，也可以只指定一个 IP 地址，或者一个端口号，或者一个服务器名。如果 listen 指令只设置一个服务器名或 IP 地址，那么它的默认端口号为 80。

示例如下：

```
listen 127.0.0.1:8000;  
listen 127.0.0.1;  
listen 8000;  
listen *:8000;  
listen localhost:8000;
```

监听 IPV6 地址示例如下：

```
listen [::]:8000;  
listen [fe80::1];
```

示例：通过以下配置，可以设置一个虚拟主机同时支持 HTTP 和 HTTPS：

```
listen 80;  
listen 443 default ssl;
```

13.1.23 location 指令

语法：location [=|~|~*|^~] /uri/ { ... }

默认值：no

使用环境：server

该指令允许对不同的 URI 进行不同的配置，既可以使用字符串，也可以使用正则表达式。使用正则表达式，须使用以下前缀：

(1) ~*，表示不区分大小写的匹配。

(2) ~，表示区分大小写的匹配。

在匹配过程中，Nginx 将首先匹配字符串，然后再匹配正则表达式。匹配到第一个正则表达式后，会停止搜索。如果匹配到正则表达式，则使用正则表达式的搜索结果，如果没有匹配到正则表达式，则使用字符串的搜索结果。

可以使用前缀“^~”来禁止匹配到字符串后，再去检查正则表达式。匹配到 URI 后，将停止查询。

使用前缀“=”可以进行精确的 URI 匹配，如果找到匹配的 URI，则停止查询。例如“location =/”，只能匹配到“/”，而“/test.html”则不能被匹配。

正则表达式的匹配，按照它们在配置文件中的顺序进行，写在前面的优先。

示例如代码 13-1 所示：

代码 13-1

```
location = / {  
    # 仅仅匹配/  
    [ configuration A ]  
}  
location / {  
    # 匹配任何以/开头的查询，但是正则表达式及较长的字符串（例如/admin/）将被优先匹配。  
    [ configuration B ]  
}  
location ^~ /images/ {  
    # 匹配任何以/images/开头的字符串，并且停止搜索，所以正则表达式将不会被检查。  
    [ configuration C ]  
}  
location ~* \.(gif|jpg|jpeg)$ {  
    # 匹配以.gif、.jpg、.jpeg 结尾的任何请求。但是，/images/内的请求将使用 Configuration C 的配置。  
    [ configuration D ]  
}
```

请求处理匹配结果示例：

- / -> configuration A;
- /documents/document.html -> configuration B;
- /images/1.gif -> configuration C;

- /documents/1.jpg -> configuration D。

你可以采用不同的顺序定义这 4 个配置，但是，匹配结果是一样的。

另外，前缀 “@” 是一个命名标记，这种 location 不会用于正常的请求，它们通常只用于处理内部的重定向（例如：error_page、try_files），示例代码如 13-2 所示：

代码 13-2

```
location ~ /\.php$ {
    root    /home/www/htdocs/;
    index  index.php index.html index.htm;
    error_page 404 502 504 @fetch;
}

location @fetch {
    internal;
    proxy_pass http://backend;
    break;
}
```

13.1.24 log_not_found 指令

语法：log_not_found [on|off]

默认值：log_not_found on

使用环境：http, server, location

该指令用来启用或禁用 404 错误日志，这个指令可以用来禁止 Nginx 记录找不到 robots.txt 和 favicon.ico 这类文件的错误信息。

13.1.25 log_subrequest 指令

语法：log_subrequest [on|off]

默认值：log_subrequest off

使用环境：http, server, location

该指令用来启动或禁止在 access_log 中记录类似 rewrite rules、SSI requests 等子请求。

13.1.26 msie_padding 指令

语法：msie_padding [on|off]

默认值: msie_padding on

使用环境: http, server, location

此指令关闭或开启 MSIE 浏览器的 msie_padding 特性,若启用此选项,Nginx 会为 response 头部填满至 512 字节,这样就阻止了相关浏览器激活友好错误页面,因此不会隐藏更多的错误信息。

13.1.27 msie_refresh 指令

语法: msie_refresh [on|off]

默认值: msie_refresh off

使用环境: http, server, location

此指令允许或禁止为 MSIE 指派一个 refresh 而不是重定向。

13.1.28 open_file_cache 指令

语法: open_file_cache max = N [inactive = time] | off

默认值: open_file_cache off

使用环境: http, server, location

该指令用于设置打开文件的缓存。以下信息可以被缓存:

- 打开文件描述符的文件大小和修改时间信息。
- 存在的目录信息。
- 搜索文件的错误信息: 文件不存在,无权限读取等错误。
- max——指定缓存的最大数量,当缓存数量不够用时,使用 LRU 规则清除最早写入、最近没有被访问过的缓存。
- inactive——指定缓存的过期时间,默认值为 60 秒。
- off——关闭缓存。

示例代码如下:

```
open_file_cache max = 1000 inactive = 20s; open_file_cache_valid 30s;  
open_file_cache_min_uses 2; open_file_cache_errors on;
```

13.1.29 open_file_cache_errors 指令

语法: `open_file_cache_errors on | off`

默认值: `open_file_cache_errors off`

使用环境: `http, server, location`

该指令设置是否开启搜索文件的缓存错误。

13.1.30 open_file_cache_min_uses 指令

语法: `open_file_cache_min_uses number`

默认值: `open_file_cache_min_uses 1`

使用环境: `http, server, location`

该指令用于指定在 `open_file_cache` 指令设置的时间内文件的最小使用数。如果打开的文件超过该数量, 则文件描述符会保持缓存中的打开状态。

13.1.31 open_file_cache_valid 指令

语法: `open_file_cache_valid time`

默认值: `open_file_cache_valid 60`

使用环境: `http, server, location`

该指令用于指定检查 `open_file_cache` 指令中条款有效性的时间, 单位为秒。

13.1.32 optimize_server_names 指令

语法: `optimize_server_names [on | off]`

默认值: `optimize_server_names on`

使用环境: `http, server`

该指令激活或停用基于域名的虚拟主机的优化。特别地, 影响了检查中使用的主机名重定向。如果打开优化选项, 以及侦听所有基于域名的虚拟主机, 监听一个 IP 地址和所有基于域名的服务器地址: 端口对有相同的配置, 然后名称不检查, 并在执行请求重定向使用的第一台服务器的名称。

如果重定向必须使用主机名并且在客户端检查通过，那么这个参数必须设置为 off。

注意：这个参数不建议在 nginx 0.7.x 版本中使用，请使用 `server_name_in_redirect`。

13.1.33 port_in_redirect 指令

语法：port_in_redirect [on|off]

默认值：port_in_redirect on

使用环境：http, server, location

该指令允许或阻止 Nginx 重定向过程中的端口操作。如果 port_in_redirect 为 on，Nginx 在请求重定向时则不会加上端口。

13.1.34 recursive_error_pages 指令

语法：recursive_error_pages [on|off]

默认值：recursive_error_pages off

使用环境：http, server, location

该指令允许或禁止除了第一条 error_page 指令之外的 error_page 指令。

13.1.35 resolver 指令

语法：resolver address

默认值：no

使用环境：http, server, location

13.1.36 resolver_timeout 指令

语法：resolver_timeout time

默认值：30

使用环境：http, server, location

该指令用于解析超时时间。

13.1.37 root 指令

语法: root path

默认值: root html

使用环境: http, server, location, if in location

该指令主要用来指定请求的文档根目录。例如, 配置内容为 location /i/ { root /spool/w3; } 时, 请求 URI 地址 “/i/top.gif” 将返回文件 “/spool/w3/i/top.gif” 的内容给客户端。

13.1.38 satisfy_any 指令

语法: satisfy_any [onoff]

默认值: satisfy_any off

使用环境: location

该指令用于检查至少一个成功的访问权限认证, 它可以在 NginxHttpAccess 模块或 NginxHttpAuthBasicModule 中使用。

```
location / {  
    satisfy_any on;  
    allow 192.168.1.0/32;  
    deny all;  
    auth_basic "closed site";  
    auth_basic_user_file conf/htpasswd;  
}
```

13.1.39 send_timeout 指令

语法: send_timeout the time

默认值: send_timeout 60

使用环境: http, server, location

该指令用于设置发送给客户端的应答超时时间。超时时间是指进行了两次 TCP 握手, 还没有转为 established 状态的时间。如果超过这个时间, 客户端没有响应, Nginx 则关闭连接。

13.1.40 sendfile 指令

语法: sendfile [onoff]

默认值: sendfile off

使用环境: http, server, location

该指令启用或禁用 `sendfile()` 函数。`sendfile()` 是作用于数据拷贝在两个文件描述符之间的操作函数, 这个拷贝操作是在内核中操作的, 所以称为“零拷贝”。`sendfile` 函数比 `read` 和 `write` 函数要高效得多, 因为 `read` 和 `write` 要把数据拷贝到用户应用层进行操作。但是, 在 Nginx 的生产环境应用中, 出现过启用 `sendfile` 比禁用 `sendfile` 效率更低的情况, 所以, 是否启用该函数须谨慎选择。

13.1.41 server 指令

语法: `server { ... }`

默认值: no

使用环境: http

该指令用于配置虚拟主机。在 `server { }` 中, 使用 `listen` 指令来为一个虚拟主机设置监听的 IP 和端口, 使用 `server_name` 指令来设置不同的虚拟主机名称。

13.1.42 server_name 指令

语法: `server_name name [...]`

默认值: `server_name hostname`

使用环境: server

该指令主要完成以下两项操作:

(1) 根据客户端请求 Header 头信息中的 Host 域名, 来匹配该请求应该由哪个虚拟主机配置 `server{...}` 来处理。

域名处理的优先级按照以下顺序执行:

- A. 全名。
- B. 以通配符开头的域名, 例如 “*.example.com”。
- C. 以通配符结尾的域名, 例如 “www.example.*”。
- D. 使用正则表达式的域名。

如果 Header 头信息中的 Host 域名在各 `server{...}` 中没有匹配项, 该请求将按照以下优先级顺

序由一个 `server{...}` 来处理:

- A. `listen` 指令被标记为 default 的 `server{...}`。
- B. 第一个出现 `listen` 指令 (或者默认为 80 端口) 的 `server{...}`。

(2) 如果 `server_name_in_redirect` 设置为 on, 设置的主机名将被用于 HTTP 重定向。

示例代码如下:

```
server {
    server_name  example.com www.example.com;
}
```

第一个名称为服务器的基础名称, 该虚拟主机的主机名 (hostname) 即为该名称。

可以使用通配符 “*” 来代替域名的前部分或后部分, 例如:

```
server {
    server_name  example.com *.example.com www.example.*;
}
```

以上两个服务器名称 (`example.com` 和 `*.example.com`) 可以写成一句:

```
server {
    server_name  .example.com;
}
```

服务器名称也可使用正则表达式, 在正则表达式加上 “~”, 例如:

```
server {
    server_name  www.example.com ~^www\d+\.example\.com$;
}
```

服务器的基础名称将被用于 HTTP 重定向, 如果客户端请求中没有 Host 头信息, 或者 Host 头信息没有匹配 `server_name`, 你可以使用 “*” 强制 Nginx 在 HTTP 重定向中使用 Host 头(注意: “*” 不能作为虚拟主机的第一个名称, 但是, 你可以使用一个伪名称 “_” 来代替第一个名称):

```
server {
    server_name example.com *;
}
server {
    server_name _ *;
}
```

注意: 在 Nginx 0.6 版本中该处配置稍有不同:

```
server {
    server_name _;
}
```

从 Nginx 0.7.12 版本开始, 支持空的服务器名称, 用来匹配那些没有指定 Host 头信息的客户端请求:

```
server {  
    server_name "";  
}
```

从 Nginx 0.8.25 版本开始，可以在 `server_name` 正则表达式中使用命名的捕获变量（named captures），而不是数字号来获取匹配的捕获变量。

```
server {  
    server_name ~^(www\.)?(?<domain>.+)$;  
    location / {  
        root /sites/$domain;  
    }  
}
```

一些老的 PCRE 正则表达式库提供了类似以下的语法，如果你在使用命名的捕获变量中遇到问题，可以尝试使用以下配置：

```
server {  
    server_name ~^(www\.)?(?P<domain>.+)$;  
    location / {  
        root /sites/$domain;  
    }  
}
```

13.1.43 server_name_in_redirect 指令

语法：server_name_in_redirect on|off

默认值：server_name_in_redirect on

使用环境：http, server, location

如果 `server_name_in_redirect` 指令设置为 on，Nginx 将使用 `server_name` 指令设置的一个名称来作重定向。如果 `server_name_in_redirect` 指令设置为 off，Nginx 将使用客户端请求中的 Host 头信息来作重定向。

13.1.44 server_names_hash_max_size 指令

语法：server_names_hash_max_size number

默认值：server_names_hash_max_size 512

使用环境：http

该指令用于指定服务器名称哈希表的最大值。

13.1.45 server_names_hash_bucket_size 指令

语法: `server_names_hash_bucket_size number`

默认值: `server_names_hash_bucket_size 32/64/128`

使用环境: `http`

该指令用于指定服务器名称哈希表的框大小。该默认值取决于 CPU 缓存。

13.1.46 server_tokens 指令

语法: `server_tokens on|off`

默认值: `server_tokens on`

使用环境: `http, server, location`

是否在错误页面或服务器 Header 头中输出 Nginx 版本号给客户端浏览器。

13.1.47 tcp_nodelay 指令

语法: `tcp_nodelay [on|off]`

默认值: `tcp_nodelay on`

使用环境: `http, server, location`

该指令允许或禁止使用套接字选项 `TCP_NODELAY`，仅适用于 `keep-alive` 连接。

默认情况下数据发送时，内核并不会马上发送，它可能等待更多的字节组成一个包，这样可以提高 IO 发送的效率。但在每次只发送很少字节的程序中，使用 `TCP_NODELAY` 时等待时间就会比较长。请根据实际情况选择是否开启。

13.1.48 tcp_nopush 指令

语法: `tcp_nopush [on|off]`

默认值: `tcp_nopush off`

使用环境: `http, server, location`

该指令允许或禁止使用 FreeBSD 上的 `TCP_NOPUSH`，或者 Linux 上的 `TCP_CORK` 套接字

选项。该选择仅在 `sendfile` 开启的时候才起作用。设置该选择的原因是 Nginx 在 Linux 和 FreeBSD 4.x 上，试图在一个包中发送它的 HTTP 应答头。

13.1.49 try_files 指令

语法: `try_files param1 [param2 ... paramN] fallback`

默认值: `none`

使用环境: `location`

该指令用于告诉 Nginx 测试每个文件是否存在，并且使用首先找到的文件作为 URI。如果没有找到文件，则调用 `location fallback`（“fallback”可以为任何名称）。`fallback` 是一个请求参数，它可以是一个命名的 `location`，也可以是任何可能的 URI。示例：

```
location / {
    try_files index.html index.htm @fallback;
}

location @fallback {
    root /var/www/error;
    index index.html;
}
```

13.1.50 types 指令

语法: `types {...}`

使用环境: `http, server, location`

该指令用于应答的 MIME-types 对应的扩展名，一个 MIME-types 可以对应多个扩展名。默认情况下，使用这些扩展名：

```
types {
    text/html    html;
    image/gif    gif;
    image/jpeg   jpg;
}
```

完整的 MIME-types 与扩展名的映射表在被包含的 `conf/mime.types` 文件中。

如果你想让某些 `location` 使用 MIME 类型: `application/octet-stream`，则可以使用以下配置：

```
location /download/ {
    types {}
    default_type application/octet-stream;
}
```

13.1.51 HTTP 核心模块中可以使用的变量

Nginx HTTP 核心模块支持一些与 Apache 变量名称相同的内置变量,例如: \$http_user_agent、\$http_cookie, 此外, 还支持一些 Nginx 特有的其他变量:

\$arg_PARAMETER

该变量包含了当查询字符串时, GET 请求可变参数的值。

\$args

这个变量等于请求行中的参数。

\$binary_remote_addr

二进制格式的客户端地址。

\$content_length

这个变量等于客户端请求头中的 content-length 值。

\$content_type

这个变量等于客户端请求头中的 content-type 值。

\$cookie_COOKIE

客户端请求 Header 头中的 cookie 变量。前缀 “\$cookie_” 加上 cookie 名称的变量, 该变量的值即为 cookie 名称的值。例如客户端请求头中某一行的内容为 “Cookie:PHPSESSID=6fd171f5c1b4e42783f2b6b9d7124065; userid=2”, 那么 Nginx 变量 \$cookie_userid 的值则为 2。

\$document_root

这个变量等于当前请求所属的 root 指令设置的文档根目录路径。

\$document_uri

这个变量与 \$uri 类似。

\$host

这个变量等于客户端请求头中的 Host 值, 如果客户端请求头中没有 Host 值, 那么这个变量等于为当前请求提供服务的服务器名称。

\$http_HEADER

客户端请求 header 头中的变量。将客户端请求头每行名称中的横线 “-” 换成下划线 “_”,

大写字母转为小写字母，前缀加上“\$http_”，即为该名称对应的变量。例如客户端请求头中 cookie 行的内容为“Accept-Encoding: gzip, deflate”，那么在 Nginx 中则有一个变量 \$http_accept-encoding，该变量的值为“gzip, deflate”。类似的变量有：\$http_user_agent、\$http_referer...

\$is_args

如果 \$args 已经设置，则该变量的值为“?”，否则为“”。

\$limit_rate

这个变量允许限制连接速率。

\$query_string

这个与 \$args 类似。

\$remote_addr

客户端的 IP 地址。

\$remote_port

客户端的端口。

\$remote_user

这个变量等于用户名，Auth Basic 模块中会使用到该变量。

\$request_filename

这个变量等于当前请求的文件路径，由 root、alias 指令及 URI 请求生成。

\$request_body

这个变量（0.7.58 以上版本支持）包含了请求的 body 主体内容。在使用 proxy_pass 或 fastcgi_pass 指令的 location 中，这个变量比较有意义。

\$request_body_file

客户端请求主体的临时文件名。

\$request_method

这个变量等于 HTTP 请求的动作，常使用的动作为 GET 或 POST。

在 0.8.20 及之前的版本中，这个变量总是等于主请求的动作名，如果当前请求是一个子请求，该变量的值并不是当前请求的动作名。

\$request_uri

这个变量等于带有参数的完整 URI。

\$scheme

HTTP 方法（例如 IE 浏览器支持的 http 和 https）。按需使用，示例如下：

```
rewrite ^(.+)$ $scheme://example.com$1 redirect;  
$server_addr
```

这个变量等于服务器的地址。通常，在一次系统调用后，该变量的值可以确定。如果为了避免系统调用，则要在 listen 指令中指出地址，并且使用参数 bind。

\$server_name

服务器的主机名。

\$server_port

请求到达的服务器端口。

\$server_protocol

这个变量等于请求采用的协议，通常该变量的值为 HTTP/1.0 或 HTTP/1.1。

\$uri

这个变量等于当前请求的 URI（不带参数），它可以不同于最初的值，例如可以被内部重定向所改变。

13.2 HTTP Upstream 模块

HTTP Upstream 模块是与反向代理、负载均衡相关的模块，包含 ip_hash、server、upstream 等指令。这一模块已在前面第 6 章中做了详细介绍。

13.3 HTTP Access 模块

HTTP Access 模块提供了一个简单的基于 host 名称的访问控制。通过该模块，可以允许或禁止指定的 IP 地址或 IP 段访问某些虚拟主机或目录。

示例如下：

```
location / {  
    deny    192.168.1.1;  
    allow   192.168.1.0/24;  
    allow   10.1.1.0/16;  
    deny    all;  
}
```

13.3.1 allow 指令

语法：allow [address | CIDR | all]

默认值：none

使用环境：http, server, location, limit_except

允许指定的 IP 地址或 IP 段访问某些虚拟主机或目录。

13.3.2 deny 指令

语法：deny [address | CIDR | all]

默认值：none

使用环境：http, server, location, limit_except

禁止指定的 IP 地址或 IP 段访问某些虚拟主机或目录。

13.4 HTTP Auth Basic 模块

该模块采用基于 HTTP 基本身份验证的用户名和密码登录方式，来保护你的虚拟主机或目录。

示例如下：

```
location / {  
    auth_basic            "Restricted";  
    auth_basic_user_file  httpasswd;  
}
```

13.4.1 auth_basic 指令

语法：auth_basic [textloff]

默认值：auth_basic off

使用环境: http, server, location, limit_except

该指令用于指定弹出的用户名和密码登录框中提示的名称, 例如设置 auth_basic 名称为 “Kingsoft”, 效果如图 13-1 所示:

```
location /admin/ {  
    auth_basic "Kingsoft";  
    auth_basic_user_file htpasswd;  
}
```

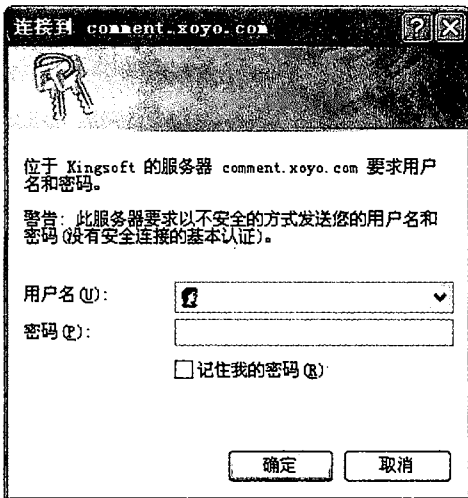


图 13-1 auth_basic 指令弹出的用户名和密码登录框

如果设置 auth_basic off, 则关闭 HTTP 基本身份验证。

13.4.2 auth_basic_user_file 指令

语法: auth_basic_user_file the_file

默认值: no

使用环境: http, server, location, limit_except

该指令用于设置 htpasswd 密码文件的名称。在 Nginx 0.6.7 版本之前, htpasswd 密码文件相对路径为 Nginx 安装目录, 在 Nginx 0.6.7 版本之后, htpasswd 密码文件的相对路径为 nginx.conf 文件所在的路径。如果您的 Nginx 安装在 /usr/local/nginx/ 路径下, 并设置了 “auth_basic_user_file htpasswd;”, 那么 Nginx 将读取 /usr/local/nginx/conf/htpasswd 文件。

htpasswd 密码文件的格式如下:

用户名:密码
用户名 2:密码 2:注释
用户名 3:密码 3¹

你也可以使用 perl 创建密码文件, pw.pl 的内容如下:

```
#!/usr/bin/perl
use strict;

my $pw=$ARGV[0] ;
print crypt($pw,$pw)."\n";
```

然后执行:

```
chmod +x pw.pl
./pw.pl password
papAq5PwY/QQM
```

papAq5PwY/QQM 就是 password 的 crypt()密码。

13.5 HTTP Autoindex 模块

该模块用于提供自动目录列表。该模块只有在找不到默认 index 文件时才启用。

13.5.1 autoindex 指令

语法: autoindex [on|off]

默认值: autoindex off

使用环境: http, server, location

该指令允许和禁止自动目录列表。

13.5.2 autoindex_exact_size 指令

语法: autoindex_exact_size [on|off]

默认值: autoindex_exact_size on

使用环境: http, server, location

该指令用于设置按照什么单位显示目录列表中文件的大小、默认字节, 或者 KB、MB、GB。

¹ 密码必须使用函数 crypt(3) 加密, 你可以通过 Apache 的 htpasswd 程序生成密码。

13.5.3 autoindex_localtime 指令

语法: autoindex_localtime [on|off]

默认值: autoindex_localtime off

使用环境: http, server, location

该指令的参数为 on 时, 采用本地时间显示文件修改时间, 为 off 时, 采用 GMT 格林尼治时间显示文件修改时间。

13.6 HTTP Browser 模块

该模块可以根据客户端 “User-agent” 请求头中的信息创建变量。

`$modern_browser`——如果浏览器被识别为新浏览器, 则等于 `modern_browser_value` 指令设置的值。

`$ancient_browser`——如果浏览器被识别为旧浏览器, 则等于 `ancient_browser_value` 指令设置的值。

`$msie`——如果浏览器被鉴定为 MSIE 浏览器, 该变量的值则为 1。

配置示例如下:

(1) 根据浏览器类型选择不同的首页文件, 代码如下:

```
modern_browser_value "modern.";
modern_browser msie 5.5;
modern_browser gecko 1.0.0;
modern_browser opera 9.0;
modern_browser safari 413;
modern_browser konqueror 3.0;
index index.${modern_browser}html index.html;
```

(2) 重定向旧的浏览器, 代码如下:

```
modern_browser msie 5.0;
modern_browser gecko 0.9.1;
modern_browser opera 8.0;
modern_browser safari 413;
modern_browser konqueror 3.0;
modern_browser unlisted;
ancient_browser Links Lynx Netscape4;
```

```
if ($ancient_browser){  
    rewrite ^ /ancient.html;  
}
```

13.6.1 ancient_browser 指令

语法: ancient_browser line [line...]

默认值: no

使用环境: http, server, location

该指令用于设置 “User-agent” 头信息中包含的旧的浏览器名称，多个浏览器名称可以用空格分隔。特殊的行可以采用正则表达式，例如 “^Mozilla/[1-4]”。

13.6.2 ancient_browser_value 指令

语法: ancient_browser_value line

默认值: ancient_browser_value 1

使用环境: http, server, location

该指令为\$ancient_browser 变量指定 Value 值。

13.6.3 modern_browser 指令

语法: modern_browser browser version|unlisted

默认值: no

使用环境: http, server, location

该指令用于设置新浏览器类型，可选的浏览器参数只能是 msie、gecko（Mozilla-based 浏览器）opera、safari、konqueror 几种。

版本可以设置为 X、X.X、X.X.X 或 X.X.X.X，它们各自的最大值分别为 4000、4000.99、4000.99.99 和 4000.99.99.99。

特殊的值 “unlisted” 表示即不属于新浏览器，也不属于旧浏览器的其他浏览器类型。

如果 “User-Agent” 头信息为空值，将会被归为新浏览器。

代码 13-3 的配置可以判断客户端浏览器是否为手机浏览器：

代码 13-3

```
modern_browser  unlisted;

ancient_browser "GoBrowser";
ancient_browser "MIDP";
ancient_browser "WAP";
ancient_browser "UP.Browser";
ancient_browser "Smartphone";
ancient_browser "Obigo";
ancient_browser "Mobile";
ancient_browser "AU.Browser";
ancient_browser "wxd.Mms";
ancient_browser "WxdB.Browser";
ancient_browser "CLDC";
ancient_browser "UP.Link";
ancient_browser "KM.Browser";
ancient_browser "UCWEB";
ancient_browser "SEMC-Browser";
ancient_browser "Mini";
ancient_browser "Symbian";
ancient_browser "Palm";
ancient_browser "Nokia";
ancient_browser "Panasonic";
ancient_browser "MOT-";
ancient_browser "SonyEricsson";
ancient_browser "NEC-";
ancient_browser "Alcatel";
ancient_browser "Ericsson";
ancient_browser "BENQ";
ancient_browser "BenQ";
ancient_browser "Amoisonic";
ancient_browser "Amoi";
ancient_browser "Capitel";
ancient_browser "PHILIPS";
ancient_browser "SAMSUNG";
ancient_browser "Lenovo";
ancient_browser "Mitsu";
ancient_browser "Motorola";
ancient_browser "SHARP";
ancient_browser "WAPPER";
ancient_browser "LG-";
ancient_browser "LG/";
ancient_browser "EG900";
ancient_browser "CECT";
ancient_browser "Compal";
ancient_browser "kejian";
ancient_browser "Bird";
ancient_browser "BIRD";
ancient_browser "G900/V1.0";
ancient_browser "Arima";
ancient_browser "CTL";
```



```
ancient_browser "TDG";
ancient_browser "Daxian";
ancient_browser "DBTEL";
ancient_browser "Eastcom";
ancient_browser "EASTCOM";
ancient_browser "PANTECH";
ancient_browser "Dopod";
ancient_browser "Haier";
ancient_browser "HAIER";
ancient_browser "KONKA";
ancient_browser "KEJIAN";
ancient_browser "LENOVO";
ancient_browser "Soutec";
ancient_browser "SOUTEC";
ancient_browser "SAGEM";
ancient_browser "SEC";
ancient_browser "SED-";
ancient_browser "EMOL";
ancient_browser "INNO55";
ancient_browser "ZTE";
ancient_browser "iPhone";
ancient_browser "Android";
ancient_browser "Windows CE";
ancient_browser "DX";
ancient_browser "TELSON";
ancient_browser "TCL";
ancient_browser "oppo";
ancient_browser "ChangHong";
ancient_browser "MALATA";
ancient_browser "KTOUCH";
ancient_browser "TIANYU";
ancient_browser "TOUCH";
ancient_browser "MAUI";
ancient_browser "J2ME";
ancient_browser "BlackBerry";
ancient_browser "yulong";
ancient_browser "coolpad";
```

```
if ( $ancient_browser )
{
    proxy_pass http://mobile.domain.com;
}
```

13.6.4 modern_browser_value 指令

语法: modern_browser_value line

默认值: modern_browser_value 1

使用环境: http, server, location

该变量为\$modern_browser 变量指定 Value 值。

13.7 HTTP Charset 模块

该模块用来添加文本编码类型到 HTTP 应答头 “Content-Type indicated”。此外，该模块还能够将服务器端网页原来的文本编码转换成另一种文本编码，输出给客户端。例如：

```
charset            windows-1251;  
source_charset     koi8-r;
```

13.7.1 charset 指令

语法: charset encoding|off

默认值: charset off

使用环境: http, server, location, if in location

该指令用来添加文本编码类型到 HTTP 应答头 “Content-Type indicated”。如果编码与 source_charset 指令设置的编码不一样，将进行重编码。参数为 “off”。

13.7.2 charset_map 指令

语法: charset_map encoding1 encoding2 {...}

默认值: no

使用环境: http, server, location

该指令用来添加字符集映射，可以将一种编码映射成另一种编码。代码符号以十六进制方式表示。该指令主要是针对俄文的，示例如下：

```
charset_map koi8-r windows-1251 {  
    C0 FE ; # small yu  
    C1 E0 ; # small a  
    C2 E1 ; # small b  
    C3 F6 ; # small ts  
    # ...  
}
```

以上示例是为了将 KOI8-R 编码转换成 Windows-1251 编码。

美国英语使用的编码是 ASCII（American National Standard Code）编码，它是一种 7 位（128 个字符）编码，只包含拉丁字母和一些符号，因此西里尔字符编码采用 7 位是不够的。KOI8-R

(Kod Obmena Informatsii, 俄语 К о д О б м е н а И н ф а р м а ц и и 的拉丁写法, 缩写为 К О И 8) 是一种官方的 Internet 标准编码, 它是一种包含西里尔字母的 8 位编码, 它是由 KOI8 派生的, KOI8-R 中的 8 就是指 8 位的意思, R 指俄语 (KOI8-U 使用于乌克兰语), 这在 Cyrillic Character Set (RFC, Request For Comments, 请求注解, Internet 标准草案, 1489) 的注册纪录中有详细描述。它同时也是事实上的 e-mail 和 NNTP (Network News Transfer Protocol, 网络新闻传输协议) 的西里尔字符标准。另外, 它还是 Unix 系统的标准编码, 如图 13-2 所示。

NUL	SOH	STX	ETX	EOT	ENQ	ACK	BEL	BS	HT	LF	VT	FF	CR	SO	SI
DLE	DC1	DC2	DC3	DC4	NAK	SYN	ETB	CAN	EM	SUB	ESC	FS	GS	RS	US
SPC	!	"	#	\$	%	&	'	()	*	+	,	-	.	/
0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?
@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
P	Q	R	S	T	U	V	W	X	Y	Z	[\]	^	_
`	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
p	q	r	s	t	u	v	w	x	y	z	{		}	~	DEL
—		г	г	л	л	т	т	т	т	+	■	■	■	■	■
■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■
=		ё	ё	п	п	п	п	п	п	п	п	п	п	п	п
		ё	ё	п	п	п	п	п	п	п	п	п	п	п	п
ю	а	б	ц	д	е	ф	г	х	и	SPC	к	л	м	н	о
п	я	р	с	т	у	ж	в	ь	ы	з	ш	э	щ	ч	ъ
н	я	р	с	т	у	ж	в	ь	ы	з	ш	э	щ	ч	ъ
п	я	р	с	т	у	ж	в	ь	ы	з	ш	э	щ	ч	ъ

图 13-2 KOI8-R 编码

CP1251 是 MS Widows 编码页 1251, 它是 Microsoft 西里尔字符的标准编码, 它也恰恰是 MS Windows 平台事实上的西里尔字符标准, 如图 13-3 所示。

NUL	SOH	STX	ETX	EOT	ENQ	ACK	BEL	BS	HT	LF	VT	FF	CR	SO	SI
DLE	DC1	DC2	DC3	DC4	NAK	SYN	ETB	CAN	EM	SUB	ESC	FS	GS	RS	US
SPC	!	"	#	\$	%	&	'	()	*	+	,	-	.	/
0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?
@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
P	Q	R	S	T	U	V	W	X	Y	Z	[\]	^	_
`	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
p	q	r	s	t	u	v	w	x	y	z	{		}	~	DEL
Ъ	Г	г	„	„	„	„	„	„	„	„	„	„	„	„	„
Ъ	Г	г	„	„	„	„	„	„	„	„	„	„	„	„	„
У	у	у	у	у	у	у	у	у	у	у	у	у	у	у	у
„	„	„	„	„	„	„	„	„	„	„	„	„	„	„	„
А	Б	В	Г	Д	Е	Ж	З	И	Й	SPC	Л	М	Н	О	П
Р	С	Т	У	Ф	Х	Ц	Ч	Ш	Щ	Ъ	Ы	Ь	Э	Ю	Я
а	б	в	г	д	е	ж	з	и	й	к	л	м	н	о	п
р	с	т	у	ф	х	ц	ч	ш	щ	ъ	ы	ь	э	ю	я

图 13-3 Widows-CP1251 编码

13.7.3 override_charset 指令

语法: `override_charset on|off`

默认值: `override_charset off`

使用环境: `http, server, location, if in location`

当该指令开启时, 如果后端的 FASTCGI 服务器响应头带有 “Content-Type” 头信息, 将开启编码转换。

13.7.4 source_charset 指令

语法: `source_charset encoding`

默认值: `no`

使用环境: `http, server, location, if in location`

该指令用于设定原始编码, 如果编码与 `charset` 指令设置的编码不一样, 将进行重编码。

13.8 HTTP Empty Gif 模块

该指令可以保持一个 1×1 像素的透明 GIF 图片在内存中, 当请求该图片时, 能够得到非常快的响应速度。示例如下:

```
location = /_gif {  
    empty_gif;  
}
```

13.8.1 empty_gif 指令

语法: `empty_gif`

默认值: `n/a`

使用环境: `location`

13.9 HTTP Fcgi 模块

该模块用于设置 Nginx 与 FastCGI 进程交互, 并通过传递参数来控制 FastCGI 进程工作。

示例如代码 13-4 所示:

代码 13-4

```
location / {
    fastcgi_pass    localhost:9000;
    fastcgi_index   index.php;

    fastcgi_param   SCRIPT_FILENAME    /home/www/scripts/php$fastcgi_script_name;
    fastcgi_param   QUERY_STRING       $query_string;
    fastcgi_param   REQUEST_METHOD     $request_method;
    fastcgi_param   CONTENT_TYPE       $content_type;
    fastcgi_param   CONTENT_LENGTH     $content_length;
}
```

缓冲示例如代码 13-5 所示:

代码 13-5

```
http {
    fastcgi_cache_path    /path/to/cache levels=1:2
                          keys_zone=NAME:10m
                          inactive=5m    clean_time=2h00m;

    server {
        location / {
            fastcgi_pass    http://127.0.0.1;
            fastcgi_cache    NAME;
            fastcgi_cache_valid    200 302 1h;
            fastcgi_cache_valid    301    1d;
            fastcgi_cache_valid    any    1m;
            fastcgi_cache_min_uses    1;
            fastcgi_cache_use_stale error timeout invalid_header http_500;
        }
    }
}
```

13.9.1 fastcgi_buffers 指令

语法: fastcgi_buffers the_number is_size;

默认值: fastcgi_buffers 8 4k/8k;

使用环境: http, server, location

该指令设置了读取 FastCGI 进程返回信息的缓冲区数量和大小。默认情况下, 一个缓冲区的大小应该跟操作系统的页大小一样, 默认为 4k/8k 自动选择。

你可以通过以下命令查看系统页大小:

```
getconf PAGESIZE
```

或

```
getconf PAGE_SIZE
```

或者编译以下 C 代码来查看：

```
#include <unistd.h>
#include <stdio.h>
int main()
{
    printf("Page size on your system = %i bytes\n", getpagesize());
    return 0;
}
```

13.9.2 fastcgi_buffer_size 指令

语法：fastcgi_buffer_size the_size

默认值：fastcgi_buffer_size 4k/8k

使用环境：http, server, location

该指令设置 FastCGI 服务器相应头部的缓冲区大小。通常情况下，该缓冲区的大小设置等于 fastcgi_buffers 指令设置的一个缓冲区的大小。默认为 4k/8k 自动选择。

13.9.3 fastcgi_cache 指令

语法：fastcgi_cache zone;

默认值：off

使用环境：http, server, location

该指令用来设置缓存在共享内存中的名称。一款区域可能被几个地方引用。

13.9.4 fastcgi_cache_key 指令

语法：fastcgi_cache_key line;

默认值：none

使用环境：http, server, location

该指令用来设置被缓存的 key，示例如下：

```
fastcgi_cache_key localhost: 9000 $ request_uri;
```

13.9.5 fastcgi_cache_methods 指令

语法: `fastcgi_cache_methods [GET HEAD POST];`

默认值: `fastcgi_cache_methods GET HEAD;`

使用环境: `main, http, location`

该指令用于设置哪些 HTTP 请求可以被缓存。参数可选值为 GET、HEAD、POST，默认值为 GET、HEAD。另外，如果你只设置“`fastcgi_cache_methods POST;`”，GET/HEAD 的缓存不会被禁止。

13.9.6 fastcgi_index 指令

语法: `fastcgi_index file`

默认值: `none`

使用环境: `http, server, location`

如果请求的 FastCGI URI 以“/”斜线结束，该指令设置的文件会被附加到 URI 的后面并保存在变量 `$fastcgi_script_name` 中。

13.9.7 fastcgi_hide_header 指令

语法: `fastcgi_hide_header name`

使用环境: `http, server, location`

默认情况下，Nginx 不会将 FastCGI 进程返回的“Status”、“X-Accel-...” Header 头信息返回给客户端。该指令可以被用来隐藏其他的 Header 头信息。

如果需要“Status”、“X-Accel-...”消息头，就须要使用 `fastcgi_pass_header` 指令让 FastCGI 强制发送消息头给客户端。

13.9.8 fastcgi_ignore_client_abort 指令

语法: `fastcgi_ignore_client_abort on|off`

默认值: `fastcgi_ignore_client_abort off`

使用环境: http, server, location

如果客户端中断对服务器的请求, 则该指令可决定当前对 FastCGI 的请求是否该中断。

13.9.9 fastcgi_intercept_errors 指令

语法: fastcgi_intercept_errors on|off

默认值: fastcgi_intercept_errors off

使用环境: http, server, location

该指令用来决定是否要把客户端转向 4xx 和 5xx 错误页, 或者允许 Nginx 自动指定错误页。

注意: 你需要在此明确错误页, 它才是有用的。Igor 曾说: “如果没有定制的处理机制, Nginx 不会拦截一个没有默认页的错误。Nginx 只会拦截一些小的错误, 放过一些其他错误。

13.9.10 fastcgi_max_temp_file_size 指令

语法: fastcgi_max_temp_file_size 0

该指令用于关闭磁盘缓冲。

13.9.11 fastcgi_param 指令

语法: fastcgi_param parameter value

默认值: none

使用环境: http, server, location

该指令指定的参数, 将被传递给 FastCGI-server。

它可能使用字符串、变量及其它们的组合来作为参数值。如果不在此制定参数, 它就会继承外层设置; 如果在此设置了参数, 将清除外层相关设置, 仅启用本层设置。

下面是一个例子, 对 PHP 来说的最精简的必要参数:

```
fastcgi_param SCRIPT_FILENAME /home/www/scripts/php$fastcgi_script_name;
fastcgi_param QUERY_STRING $query_string;
```

参数 SCRIPT_FILENAME 是 PHP 用来确定执行脚本的名字, 而参数 QUERY_STRING 是它的一个子参数。

如果要处理 POST，那么这三个附加参数是必要的：

```
fastcgi_param REQUEST_METHOD $request_method;
fastcgi_param CONTENT_TYPE $content_type;
fastcgi_param CONTENT_LENGTH $content_length;
```

如果 PHP 在编译时使用了——enable-force-cgi-redirect 选项，设置参数 REDIRECT_STATUS 的值为 200 就是必须的了。

```
fastcgi_param REDIRECT_STATUS 200;
```

13.9.12 fastcgi_pass 指令

语法：fastcgi_pass fastcgi-server

默认值：none

使用环境：location, if in location

该指令用于指定 FastCGI 服务器监听的端口或 Unix 套接字。

示例如下：

```
fastcgi_pass localhost:9000;
```

使用 Unix 套接字的示例如下：

```
fastcgi_pass unix:/tmp/fastcgi.socket;
```

如果有多台 FastCGI 服务器，你也可以使用 upstream 指令指定 FastCGI 服务器，示例如下：

```
upstream backend {
    server localhost:1234;
}
fastcgi_pass backend;
```

13.9.13 fastcgi_pass_header 指令

语法：fastcgi_pass_header name

默认值：http, server, location

13.9.14 fastcgi_read_timeout 指令

语法：fastcgi_read_timeout time



默认值: 60

使用环境: http, server, location

该指令用于设置 upstream 模块等待 FastCGI 进程发送数据的超时时间, 默认值为 60 秒, 如果你有执行时间较长的 FastCGI 程序, 则须更改此值。

13.9.15 fastcgi_redirect_errors 指令

语法: fastcgi_redirect_errors on|off

该指令用于开启或关闭 FastCGI 错误重定向。

13.9.16 fastcgi_split_path_info 指令

语法: fastcgi_split_path_info regex

使用环境: location

Nginx 版本要求: >= 0.7.31

```
location ~ ^(.+\.php)(.*)$ {  
    ...  
    fastcgi_split_path_info ^(.+\.php)(.*)$;  
    fastcgi_param SCRIPT_FILENAME /path/to/php$fastcgi_script_name;  
    fastcgi_param PATH_INFO $fastcgi_path_info;  
    fastcgi_param PATH_TRANSLATED $document_root$fastcgi_path_info;  
    ...  
}
```

13.10 geo 模块

geo 模块主要用于做全局负载均衡, 可以根据不同的客户端 IP 访问到不同的服务器。一些针对不同地区的客户, 使用不同的服务器去处理的需求, 可以使用 geo 模块。

geo 简单的配置示例如下:

```
geo $geo {  
    default 0;  
    127.0.0.1/32 2;  
    192.168.1.0/24 1;  
    10.1.0.0/16 1;  
}
```

下面, 我们来看 geo 全局负载均衡与 upstream 反向代理的配合使用, 示例如代码 13-6 所示:

代码 13-6

```
user www www;

worker_processes 10;

error_log /data1/logs/nginx_error.log crit;

pid /tmp/nginx.pid;

worker_rlimit_nofile 51200;

events
{
    use epoll;

    worker_connections 51200;
}

http
{
    include conf/mime.types;
    default_type application/octet-stream;

    geo $geo {
        default default;
        218.30.115.0/24 china_telecom;
        202.106.182.0/24 china_unicom;
        202.205.3.0/24 cernet;
    }

    upstream default.server {
        server 192.168.0.2;
    }

    upstream china_telecom.server {
        server 192.168.0.3;
    }

    upstream china_unicom.server {
        server 192.168.0.4;
    }

    upstream cernet.server {
        server 192.168.0.5;
    }

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

```
        proxy_redirect off;
        proxy_set_header Host $host;
        proxy_set_header X-Real-IP $remote_addr;
        proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
        proxy_pass http://$geo.server$request_uri;
    }
}
```

13.10.1 geo 指令

语法: geo [\$ip_variable] \$variable { ... }

默认值: none

使用环境: http

geo 指令默认使用客户端的 IP 地址 (变量 \$remote_addr 的值) 来进行全局负载均衡, 从 Nginx 0.7.27 版本开始, 可以支持使用指定的变量来做全局负载均衡, 例如:

```
geo $arg_remote_addr $geo {
    ...;
}
```

geo 指令支持以下描述:

default: 任何 IP 地址, 相当于 0.0.0.0/0。

include: 可以引用一个文本文件, 里面包含 geo 的配置文件。当 geo 配置的 IP 段较多时, 可以将 IP 段写在另一个配置文件中, 并引用进来。

ranges: 从 Nginx 0.7.23 版本开始, 支持使用区间形式来指定 IP 段。该指令必须写在 geo 配置环境的第一行。

Default、include 描述示例代码如 13-7 所示:

代码 13-7

```
geo $country {
    default          no;
    include          conf/geo.conf;
    127.0.0.0/24     us;
    127.0.0.1/32     ru;
    10.1.0.0/16      ru;
    192.168.1.0/24   uk;
}
```

引用的配置文件 conf/geo.conf 的内容如下:

```
10.2.0.0/16      ru;
192.168.2.0/24   ru;
```

当指定的 IP 段区间存在包含关系时, 取最精确的配值。例如在以下示例中, IP 地址 127.0.0.1 取得的 value 值为 ru, 而不是 us。

Ranges 区间形式指定 IP 段示例如代码 13-8 所示:

代码 13-8

```
geo $country {
    ranges;
    default                no;
    127.0.0.0-127.0.0.0    us;
    127.0.0.1-127.0.0.1    ru;
    127.0.0.1-127.0.0.255  us;
    10.1.0.0-10.1.255.255  ru;
    192.168.1.0-192.168.1.255 uk;
}
```

13.11 Gzip 模块

Gzip 模块主要用于对返回给客户端的网页采用 gzip 进行压缩输出。

目前, 90%的浏览器都支持 gzip 和 deflate 两种压缩格式。如果浏览器支持 gzip 压缩, 就会在 HTTP 请求头中发送一行 “Accept-Encoding: gzip, deflate”, 这时候 Nginx 服务器可以输出经过 gzip 压缩后的页面给浏览器, 浏览器再解压。这种方式可以将网络线路上传输的大量数据消减 60% 以上, 不仅节省了服务器带宽, 同时加速了用户的下载速度和体验。

一个典型的 IE7 浏览器 HTTP GET 请求头的信息如代码 13-9 所示:

代码 13-9

```
GET / HTTP/1.1
Accept: image/gif, image/jpeg, image/pjpeg, image/pjpeg,
application/x-shockwave-flash, application/vnd.ms-excel,
application/vnd.ms-powerpoint, application/msword, application/x-ms-application,
application/x-ms-xbap, application/vnd.ms-xpsdocument, application/xaml+xml, */*
Accept-Language: zh-cn
User-Agent: Mozilla/4.0 (compatible; MSIE 7.0; Windows NT 5.1; Trident/4.0;
Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1; SV1) ; baiduds; .NET CLR
2.0.50727; .NET CLR 3.0.4506.2152; .NET CLR 3.5.30729; CIBA)
Accept-Encoding: gzip, deflate
Host: www.xoyo.com
Connection: Keep-Alive
```

13.11.1 gzip 指令

语法: gzip on|off

默认值: gzip off

使用环境: http, server, location, if (x) location

该指令用于开启或关闭 gzip 模块。

13.11.2 gzip_buffers 指令

语法: gzip_buffers number size

默认值: gzip_buffers 4 4k/8k

使用环境: http, server, location

设置系统获取几个单位的缓存用于存储 gzip 的压缩结果数据流。例如 4 4k 代表以 4k 为单位, 按照原始数据大小以 4k 为单位的 4 倍申请内存。4 8k 代表以 8k 为单位, 按照原始数据大小以 8k 为单位的 4 倍申请内存。

如果没有设置, 默认值是申请跟原始数据相同大小的内存空间去存储 gzip 压缩结果。

13.11.3 gzip_comp_level 指令

语法: gzip_comp_level 1..9

默认值: zip_comp_level 1

使用环境: http, server, location

gzip 压缩比, 1 压缩比最小处理速度最快, 9 压缩比最大但处理速度最慢 (传输快但比较消耗 cpu)。

13.11.4 gzip_min_length 指令

语法: gzip_min_length length

默认值: gzip_min_length 0

使用环境: http, server, location

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 中进行获取。默认值是 0，不管页面多大都压缩。建议设置成大于 1k 的字节数，小于 1k 可能会越压越大。即：
gzip_min_length 1024。

13.11.5 gzip_http_version 指令

语法：gzip_http_version 1.0|1.1

默认值：gzip_http_version 1.1

使用环境：http, server, location

识别 http 的协议版本。由于早期的一些浏览器或 http 客户端，可能不支持 gzip 自解压，用户就会看到乱码，所以做一些判断还是有必要的²。

13.11.6 gzip_proxied 指令

语法：gzip_proxied [off|expired|no-cache|no-store|private|no_last_modified|no_etag|auth|any] ...

默认值：gzip_proxied off

使用环境：http, server, location

Nginx 作为反向代理的时候启用，开启或关闭后端服务器返回的结果，匹配的前提是后端服务器必须要返回包含“Via”的 header 头。

off——关闭所有的代理结果数据的压缩。

expired——启用压缩，如果 header 头中包含“Expires”头信息。

no-cache——启用压缩，如果 header 头中包含“Cache-Control:no-cache”头信息。

no-store——启用压缩，如果 header 头中包含“Cache-Control:no-store”头信息。

private——启用压缩，如果 header 头中包含“Cache-Control:private”头信息。

no_last_modified——启用压缩，如果 header 头中不包含“Last-Modified”头信息。

no_etag——启用压缩，如果 header 头中不包含“ETag”头信息。

auth——启用压缩，如果 header 头中包含“Authorization”头信息。

² 21 世纪都来了，现在除了类似于百度的蜘蛛之类的东西不支持自解压，99.99% 的浏览器基本上都支持 gzip 解压，所以可以不用设这个值，保持系统默认即可。

any——无条件启用压缩。

13.11.7 gzip_types 指令

语法: `gzip_types mime-type [mime-type ...]`

默认值: `gzip_types text/html`

使用环境: `http, server, location`

匹配 mime 类型进行压缩, (无论是否指定) “text/html” 类型总是会被压缩的。

注意: 如果作为 http server 来使用, 主配置文件中要包含文件类型配置文件。

```
http
{
    include      conf/mime.types;
    .....
}
```

如果你希望压缩常规的文件类型, 可以写成代码 13-10 这样。

代码 13-10

```
http
{
    include      conf/mime.types;

    gzip on;
    gzip_min_length 1000;
    gzip_buffers   4 8k;
    gzip_http_version 1.1;
    gzip_types     text/plain application/x-javascript text/css text/html
        application/xml;

    .....
}
```

13.12 HTTP Headers 模块

这一组指令主要用来设置 Nginx 返回网页内容给用户时, 附加的 Header 头信息。示例:

```
expires      24h;
expires      0;
expires      -1;
expires      epoch;
add_header   Cache-Control private;
```

13.12.1 add_header 指令

语法: `add_header name value`

默认值: none

使用环境: http, server, location

当 HTTP 应答状态码为 200、204、301、302 或 304 的时候, 增加指定的 Header 头, 其中 Header 头的值可以使用变量。

13.12.2 expires 指令

语法: `expires [time|epoch|max|off]`

默认值: expires off

使用环境: http, server, location

使用本指令可以控制 HTTP 应答中的 “Expires” 和 “Cache-Control” Header 头信息, 起到控制浏览器端页面缓存的作用。

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。

注意: expires 指令同样也只能工作于 200、204、301、302 或 304 应答状态。

13.13 HTTP Index 模块

该模块可以用于指定虚拟主机目录下的默认首页文件名称。例如:

```
index index.html; # 指定默认首页文件名称，如果指定了多个文件，那么将按照从左到右的顺序逐个查找。
```

也可以指定多个默认首页文件名称，如果您指定了多个文件，那么将按照从左到右的顺序逐个查找。可以在列表末尾加上一个绝对路径名的文件。例如：

```
index index.html index.htm; # 指定默认首页文件名称，如果指定了多个文件，那么将按照从左到右的顺序逐个查找。
```

13.13.1 index 指令

语法：index file-path [file-path [...]];

默认值：no

使用环境：server, location

该指令用来指定做默认文档的文件名，可以在文件名处使用变量。如果您指定了多个文件，那么将按照您指定的顺序逐个查找。可以在列表末尾加上一个绝对路径名的文件。

示例如下：

```
index index.$geo.html index.0.html /index.html; # 指定默认首页文件名称，如果指定了多个文件，那么将按照从左到右的顺序逐个查找。
```

13.14 HTTP Referer 模块

HTTP Referer 是 Header 的一部分，当浏览器向 Web 服务器发送请求的时候，一般会带上 Referer，告诉服务器我是从哪个页面链接过来的，服务器借此可以获得一些信息用于处理，例如防止未经允许的网站盗链图片、文件等。因为 HTTP Referer 头信息是可以通程序来伪装生成的，所以，通过 Referer 信息防盗链并非 100%可靠，但是，它能够限制大部分的盗链情况。示例如代码 13-11 所示：

代码 13-11

```
location /photos/ {
    valid_referers none blocked www.mydomain.com mydomain.com;

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

13.14.1 valid_referers 指令

语法：valid_referers [none|blocked|server_names] ...

默认值: none

使用环境: server, location

该指令会根据 Referer Header 头的内容分配一个值 0 或 1 给变量 \$invalid_referer。如果 Referer Header 头不符合 valid_referers 指令设置的有效 Referer, 变量 \$invalid_referer 将被设置为 1 (详情请见代码 13-11 示例)。

该指令的参数可以为以下内容:

none: 表示无 Referer 值的情况;

blocked: 表示 Referer 值被防火墙进行伪装, 例如: “Referer: XXXXXXXX”;

server_names: 表示一个或多个主机名称。从 Nginx 0.5.33 版本开始, server_names 中可以使用通配符 “*” 号。

13.15 HTTP Limit Zone 模块

该模块用于针对条件, 进行会话的并发连接数控制, 例如限制每个 IP 的并发连接数等。配置示例如代码 13-12 所示:

代码 13-12

```
http {
    limit_zone one $binary_remote_addr 10m;

    server {
        location /download/ {
            limit_conn one 1;
        }
    }
}
```

13.15.1 limit_zone 指令

语法: limit_zone zone_name \$variable memory_max_size

默认值: no

使用环境: http

该指令定义了一个数据区, 其中记录会话状态信息。\$variable 定义判断会话的变量; memory_max_size 定义内存记录区的总容量。示例如下:

```
limit_zone one $binary_remote_addr 10m;
```

在以上示例中，定义一个叫“one”的记录区，总容量为 10MB，以变量 `$binary_remote_addr` 作为会话的判断基准（即一个地址一个会话）。

您可能注意到了，在这里使用的是 `$binary_remote_addr` 而不是 `$remote_addr`。

`$remote_addr` 的长度为 7 至 15 bytes，会话信息的长度为 32 或 64 bytes。而 `$binary_remote_addr` 的长度为 4 bytes，会话信息的长度为 32 bytes。

当记录区的大小为 1MB 的时候，大约可以记录 32 000 个会话信息（一个会话占用 32 bytes）。

13.15.2 limit_conn 指令

语法：limit_conn zone_name max_clients_per_ip

默认值：no

使用环境：http, server, location

该指令用于指定一个会话最大的并发连接数。当超过指定的最大并发连接数时，服务器将返回“Service unavailable”（503）。

示例：

```
limit_zone one $binary_remote_addr 10m;

server {
    location /download/ {
        limit_conn one 1;
    }
}
```

以上示例中，定义一个叫“one”的记录区，总容量为 10MB，以变量 `$binary_remote_addr` 作为会话的判断基准（即一个地址一个会话）。限制 `/download/` 目录下，一个会话只能进行一个连接。简单来说，就是限制 `/download/` 目录下，一个 IP 只能发起一个连接，多于一个，一律返回“Service unavailable”（503）状态。

13.16 HTTP Limit Req 模块

该模块允许你对 Session 会话、单个客户端 IP 地址，限制指定单位时间内的并发请求数。你可以在一定程度上减轻对应用服务器的 DOS 恶意攻击。配置示例如代码 13-13 所示：

代码 13-13

```
http {
    limit_req_zone $binary_remote_addr zone=one:10m rate=1r/s;

    ...

    server {

        ...

        location /search/ {
            limit_req zone=one burst=5;
        }
    }
}
```

13.16.1 limit_req_zone 指令

语法: `limit_req_zone $session_variable zone=name_of_zone:size rate=rate`

默认值: none

使用环境: http

该指令用于定义一块内存存储区，用来存储 Session 会话的状态，Session 的变量由指定的变量构成，通常可以设置为存储客户端 IP 的变量 `$binary_remote_addr`。以下是一个示例：

```
limit_req_zone $binary_remote_addr zone=one:10m rate=1r/s;
```

在本例中，为 Session 会话状态分配了一个名为 `one` 的 10MB 内存存储区，限制了每秒只接受一个 IP 的一次请求（1 Request/Sec）。这里使用变量 `$binary_remote_addr` 代替 `$remote_addr`，可以为每个会话减少 64 字节的内存存储空间，从而使得 1MB 内存可以存储大概 16 000 个会话状态。

速度限制的条件可以设置为“请求数/秒（r/s）”或“请求数/分钟（r/m）”，比率必须为整数，如果你要设置少于每秒 1 个请求数的值，例如“1 个请求/2 秒”，你应该设置为“30r/m”。

13.16.2 limit_req 指令

语法: `limit_req zone=zone burst=burst [nodelay]`

默认值: none

使用环境: http, server, location

该指令用于指定使用的内存存储区（zone）名称，以及最大的突发请求数（burst）。如果请

求的速率超过 `limit_req_zone` 指令中设置的速率，这些请求将被延迟处理，以便能够按照 `limit_req_zone` 指令中设定的速度处理请求。多余的请求将被延迟处理，直到这些请求的数量不超过 `burst` 参数中规定的数量。在这种情况下，请求将获得服务不可用信息：“Service unavailable”（503 错误代码）。`Burst` 的值默认为 0。

示例如下：

```
limit_req_zone $binary_remote_addr zone=one:10m rate=1r/s;

server {
    location /search/ {
        limit_req zone=one burst=5;
    }
}
```

在该示例中，允许一个用户平均每秒不超过 1 个请求，同时处理的查询数量最多不超过 5 个。如果查询数小于 `burst` 值，延迟不是必须的，你可以使用 `nodelay` 参数设置为非延迟：

```
limit_req zone=one burst=5 nodelay;
```

13.17 HTTP Log 模块

该指令用于控制 Nginx 的日志格式。示例如下：

```
log_format gzip '$remote_addr - $remote_user [$time_local] '
                '$request' $status $bytes_sent '
                '$http_referer' '$http_user_agent' '$gzip_ratio';

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

13.17.1 access_log 指令

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

默认值：`access_log log/access.log combined`

使用环境：`http, server, location`

该指令用于设置日志文件的路径、格式和缓冲区大小。使用“off”作为唯一参数，将不记录日志文件。如果没有指定日志格式，将默认采用“combined”格式。缓冲区的大小必须小于写入磁盘文件的原子记录大小。这个大小对于 FreeBSD 3.0-6.0 系统无限制。

从 Nginx 0.7.4 版本开始，日志文件的路径可以包含变量，但是有一些限制：

Nginx 的 Worker 用户必须有权限去创建文件；

缓冲将不会工作；

每个请求到来时，日志文件将被打开，在日志写入后，又将立即关闭日志文件。频繁使用的文件描述符将被保存到 `open_log_file_cache` 指令设置的缓冲区中，关于日志的轮换，必须随着时间的推移被记住（在 `open_log_file_cache` 指令的参数中设置），日志仍然继续记录在旧的文件中。

Nginx 支持为每个 `location` 环境设置强大的日志格式。相同的访问可以被同时记录在一个或多个日志文件中。

13.17.2 log_format 指令

语法: `log_format name format [format ...]`

默认值: `log_format combined "..."`

使用环境: `http, server`

该指令用来描述日志格式。在日志的格式中，可以使用 Nginx 的大多数通用变量，以及一些仅在写日志时存在的变量，如下：

`$body_bytes_sent`——减去应答头之后的传输给客户端的字节数。

`$bytes_sent`——传输给客户端的字节数。

`$connection`——连接数。

`$msec`——正在写日志的当前时间（精确到微秒级，即百万分之一秒）。

`$pipe`——如果请求是 `pipelined` 管道。

`$request_length`——请求的主体（body）长度。

`$request_time`——请求在 Nginx 开始处理之前所消耗的时间（包括 TCP 连接时间、客户端的 HTTP 发送时间等），单位为毫秒（在 Nginx 0.5.19 版本之前，单位为秒）。

`$status`——应答状态。

`$time_local`——写入日志的本地服务器时间。

`$http_x_forwarded_for`——上级反向代理服务器中通常用于记录用户真实 IP 的 X-Forward 信息。

13.17.3 log_format_combined 指令

Nginx 有一个名为“combined”的预定义（内部已经定义的）指令，相当于以下设置：

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

13.17.4 open_log_file_cache 指令

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

默认值: `open_log_file_cache off`

使用环境: `http server location`

该指令用于为带有变量的日志文件路径所频繁使用的文件描述符设置缓存。指令选项如下:

`max`——缓存中可以存放的最大文件描述符。缓存满之后, 根据 LRU 算法删除最早缓存, 并且最近没有被使用的文件描述符。

`inactive`——设置一个时间, 在该时间范围内没有被使用的文件描述符将被移除。默认时间为 10 秒。

`min_uses`——设置一个文件描述符在 `inactive` 参数规定的单位时间范围内, 最少使用的次数。满足最少使用次数的文件描述符才会被放入缓存中。默认的最少使用次数为 1 次。

`valid`——设置检查同名文件是否存在的时间周期, 默认为 60 秒。

`off`——禁用缓存。

13.18 HTTP Map 模块

该模块允许你去分类, 或者映射一组值到一组不同的值, 并将这些值存储在一个变量中。示例如代码 13-14 所示:

代码 13-14

```
map $http_host $name {
    hostnames;

    default          0;

    example.com      1;
    *.example.com    1;
    test.com         2;
    *.test.com       2;
    .site.com        3;
}
```

我们可以使用本模块的映射, 来代替写许多的 `server/location` 指令或 `redirect` 重定向规则, 如代码 13-15 所示:

代码 13-15

```
map $uri $new {
    default      http://www.domain.com/home/;

    /aa          http://aa.domain.com/;
    /bb          http://bb.domain.com/;
    /john        http://my.domain.com/users/john/;
}

server {
    server_name  www.domain.com;
    rewrite ^    $new redirect;
}
```

13.18.1 map 指令

语法: map \$var1 \$var2 { ... }

默认值: none

使用环境: http

map 指令定义了被用来设置变量的映射表, 该指令有 3 个特殊的参数:

default——指定无匹配内容时的默认值;

hostnames——支持近似域名查找。示例如下:

```
*.example.com 1;
```

另外, 以 “.” 开头的字符串, 将用来精确匹配以该字符串结尾的主机名 (域名)。例如, 以下两台记录:

```
example.com 1;
*.example.com 1;
```

可以用一条记录来代替:

```
.example.com 1;
```

include——包含一个含有映射信息的文件。它可以包含多个文件。

13.18.2 map_hash_max_size 指令

语法: map_hash_max_size number

默认值: map_hash_max_size 2048

使用环境: http

该指令用于设置保存变量映射关系的哈希表的最大值。

13.18.3 map_hash_bucket_size 指令

语法: map_hash_bucket_size n

默认值: map_hash_bucket_size 32/64/128

使用环境: http

该指令用于设置一张哈希表中映射变量的最大值。默认值取决于处理器的缓存大小。

13.19 HTTP Memcached 模块

Memcached 是一个高性能的分布式内存对象缓存系统,用于动态 Web 应用以减轻数据库负载。它通过在内存中缓存数据和对象来减少读取数据库的次数,从而提供动态、数据库驱动网站的速度。Memcached 基于一个存储键/值对的 hashmap。其守护进程(daemon)是用 C 写的,但是客户端可以用任何语言来编写,并通过 Memcached 协议与守护进程通信,且它并不提供冗余(例如,复制其 hashmap 条目);当某个服务器停止运行或崩溃了,所有存放在该服务器上的键/值对都将丢失。

Memcached(官方网站: <http://www.memcached.org/>) 由 Danga Interactive 开发,用于提升 LiveJournal.com 访问速度。LJ 每秒动态页面访问量几千次,用户 700 万。Memcached 将数据库负载大幅降低,以便更好地分配资源,更快地访问。

Memcached 服务器端的编译安装如代码 13-16 所示:

代码 13-16

```
wget http://www.monkey.org/~provos/libevent-1.4.13-stable.tar.gz
tar zxvf libevent-1.4.13-stable.tar.gz
cd libevent-1.4.13-stable/
./configure --prefix=/usr
make && make install
cd ../

wget http://memcached.googlecode.com/files/memcached-1.4.4.tar.gz
tar zxvf memcached-1.4.4.tar.gz
cd memcached-1.4.4/
./configure --with-libevent=/usr
make && make install
cd ../
```


启动 Memcached 服务器端：

```
memcached -d -m 2048 -l 192.168.1.2 -p 11211
```

Nginx 中的 Memcached 模块，相当于 Memcached 的客户端软件，使用本模块可以完成一些简单的缓存任务。Nginx 官方计划在不久的将来继续完善这一模块。

配置示例如代码 13-17 所示：

代码 13-17

```
server {  
    location / {  
        set $memcached_key $uri;  
        memcached_pass    name:11211;  
        default_type      text/html;  
        error_page        404 = /fallback;  
    }  
  
    location = /fallback {  
        proxy_pass backend;  
    }  
}
```

或者在 Nginx 0.7.x 版本中使用如代码 13-18 所示的内容：

代码 13-18

```
server {  
    location / {  
        set $memcached_key $uri;  
        memcached_pass    name:11211;  
        default_type      text/html;  
        error_page        404 @fallback;  
    }  
  
    location @fallback {  
        proxy_pass backend;  
    }  
}
```

接下来，我们结合 PHP，使用 Memcached 模块做缓存。在代码 13-19 所示示例中，如果用户访问以“/cache/”开始的 URI，则读取以 URI 为 key 的 Memcached 缓存内存，显示给用户。如果 key 不存在，则重定向到“/write_memcached.php”程序文件，该 PHP 程序会往 Memcached 内写入以 URI 为 key 的缓存数据，缓存时间为 120 秒。

代码 13-19

```
location /cache/  
{  
    memcached_buffer_size 10240;
```



```
set $memcached_key $request_uri;
memcached_pass 127.0.0.1:11211;
error_page 404 =200 /write_memcached.php;
}

location ~ .*\. (php|php5)?$
{
    #fastcgi_pass unix:/tmp/php-cgi.sock;
    fastcgi_pass 127.0.0.1:9000;
    fastcgi_index index.php;
    include fcgi.conf;
    fastcgi_param MEMCACHED_KEY      $memcached_key;
}
```

write_memcached.php 文件的内容如代码 13-20 所示:

代码 13-20

```
<?php
$key = $_SERVER['MEMCACHED_KEY'];
$value = "<html><head><title>标题</title></head><body>".date("Y-m-d H:i:s")."
<BR><BR></body></html>";

$memcache = new Memcache;
$memcache->connect('127.0.0.1', 11211);
$memcache->set($key, $value, false, 120); //缓存时间为 120 秒
$memcache->close();

echo $value;

//另输出一段信息, 用来检验 Memcached 缓存是否有效。如果 Memcached 无缓存、缓存过期, 则会显示下行
信息, 否则不会。
echo "这是动态 PHP 页面";
?>
```

13.19.1 memcached_pass 指令

语法: memcached_pass [name:port]

默认值: none

使用环境: http, server, location

该指令用于设置 Memcached 服务器的地址和端口, Memcached 的 key 为 “/uri?args”。

从 Nginx 0.5.9 版本开始, Memcached 的 key 存储在 \$memcached_key 变量中。

13.19.2 memcached_connect_timeout 指令

语法: `memcached_connect_timeout [time]`

默认值: 60000

使用环境: http, server, location

该指令用于设置连接 Memcached 服务器的超时时间, 单位为毫秒。

13.19.3 memcached_read_timeout 指令

语法: `memcached_read_timeout [time]`

默认值: 60000

使用环境: http, server, location

该指令用于设置从 Memcached 服务器读取数据的超时时间, 单位为毫秒。

13.19.4 memcached_send_timeout 指令

语法: `memcached_send_timeout [time]`

默认值: 60000

使用环境: http, server, location

该指令用于设置向 Memcached 服务器发送数据的超时时间, 单位为毫秒。

13.19.5 memcached_buffer_size 指令

语法: `memcached_buffer_size [size]`

默认值: see `getpagesize (2)`

使用环境: http, server, location

该指令用于设置接收、发送数据的缓冲区大小, 单位为字节。

13.19.6 memcached_next_upstream 指令

语法: `memcached_next_upstream [error | timeout | invalid_response | not_found | off]`

默认值: error timeout

使用环境: http, server, location

该指令用于指定在何种错误条件下, 将请求转发到 upstream 负载均衡服务器池中的另一台服务器。仅适用于当 upstream 中的 memcached_pass 指令拥有两个或两个以上后端服务器的情况。

13.19.7 HTTP Memcached 模块中的变量

`$memcached_key` 该变量的内容为 Memcached 的 key 值。通常情况下, 以当前请求的 URI 作为 Memcached 的

key, 可以设置变量为:

```
set $memcached_key $uri;
```

13.19.8 第三方的 Memcached 模块

除了 Nginx 官方的 Memcached 模块之外, 还有一个第三方的 Memcached 模块, 几乎支持 Memcached 所有的功能。但是, 考虑到第三方模块通常缺乏稳定性与兼容性, 这里就不再详细介绍, 感兴趣的朋友可以访问: <http://wiki.nginx.org/NginxHttpMemcModule>, 获取更多的信息。

13.20 HTTP Proxy 模块

该模块用于转发请求到其他的服务器。Keep-alive 是指在 HTTP/1.1 协议中, 同一个连接中发出和接收多次 HTTP 请求, 节省了创建 TCP 连接过程的时间开销。而 HTTP/1.0 协议不具备 keep-alive 请求的能力。因此, 在 HTTP/1.0 协议中, 每一个到后端的请求都会创建一个连接, 传输完成后会删除这个连接。Nginx 采用 HTTP/1.1 协议与浏览器进行通信, 采用 HTTP/1.0 协议与后端服务器进行通信。

示例如下:

```
location / {
    proxy_pass      http://localhost:8000;
    proxy_set_header X-Real-IP $remote_addr;
}
```

注意: 当使用 HTTP 代理模块时 (或者当使用 FastCGI 时), 整个客户端请求在传递给后端服务器之前, 将被 Nginx 缓存, 数据传输的进度测量将不准确。

13.20.1 proxy_buffer_size 指令

语法: proxy_buffer_size the_size

默认值: proxy_buffer_size 4k/8k

使用环境: http, server, location

该指令用于设置从被代理服务器获取的第一部分应答信息的缓冲区大小, 这个缓存区间会保存用户的头信息以供 Nginx 进行规则处理, 一般只要能保存下头信息即可。但是, 也可以将该值设置得较小。

13.20.2 proxy_buffering 指令

语法: proxy_buffering on|off

默认值: proxy_buffering on

使用环境: http, server, location

该指令用于开启对被代理的服务器的应答缓冲。

如果缓冲开启, Nginx 会假设被代理服务器的应答尽可能地快, 并将其保存到缓冲区。可以使用指令 proxy_buffer_size 和 proxy_buffers 来配置缓冲区信息。

如果应答内容无法完全放在内存中, 那么应答内容的一部分将被写入在磁盘。

如果缓存关闭, 从后端服务器接收到的应答内容, 将被立即同步传送到客户端。

Nginx 不会尝试统计被代理服务器的全部应答数、数据的最大值, Nginx 能够接受 proxy_buffer_size 指令设置的值。

对于长轮询的计算机信息传输应用, 应该尽可能地设置 proxy_buffering 为 off, 否则异步地应答将被缓存, 计算机信息传输将不工作。

13.20.3 proxy_buffers 指令

语法: proxy_buffers the_number is_size;

默认值: proxy_buffers 8 4k/8k;

使用环境: http, server, location

该指令用于设置从被代理服务器读取应答信息的缓冲区数目和大小。通常情况下，一个缓冲区的大小相当于网页的大小。该指令的默认缓冲区大小取决于操作系统平台，不是 4k 就是 8k。

假设你的网页平均大小都在 32k 以下，则可以设置：

```
proxy_buffers 4 32k;
```

13.20.4 proxy_busy_buffers_size 指令

语法：proxy_busy_buffers_size size;

默认值：proxy_busy_buffers_size ["#proxy buffer size"] * 2;

使用环境：http, server, location, if

系统很忙的时候可以申请更大的 proxy_buffers 缓冲区，官方推荐为 proxy_buffers 缓冲区的大小*2，例如：

```
proxy_busy_buffers_size 64k;
```

13.20.5 proxy_cache 相关指令集

proxy_cache 相关指令集，包含 proxy_cache、proxy_cache_path、proxy_cache_methods、proxy_cache_min_uses、proxy_cache_valid、proxy_cache_use_stale 等指令，在前面第 9 章中已经有详细说明，这里就不再重复介绍。

13.20.6 proxy_connect_timeout 指令

语法：proxy_connect_timeout timeout_in_seconds

使用环境：http, server, location

该指令用于设置跟后端服务器连接的超时时间。该时间不是服务器返回页面的时间，而是发起握手等候响应的超时时间。

13.20.7 proxy_headers_hash_bucket_size 指令

语法：proxy_headers_hash_bucket_size size;

默认值：proxy_headers_hash_bucket_size 64;

使用环境：http, server, location, if

该指令用于设置哈希表的存储桶数量。

13.20.8 proxy_headers_hash_max_size 指令

语法: proxy_headers_hash_max_size size;

默认值: proxy_headers_hash_max_size 512;

使用环境: http, server, location, if

该指令用于设置哈希表的最大值。

13.20.9 proxy_hide_header 指令

语法: proxy_hide_header the_header

默认值: http, server, location

Nginx 不会传输从被代理服务器应答内容获取的"Date"、"Server"、"X-Pad"和"X-Accel-..."等 Header 行。proxy_hide_header 指令允许隐藏一些额外的 Header 行。除了必须被传递的 Header 行, 其他 Header 行都可以使用 proxy_pass_header 指令来隐藏。例如你可以按照以下方法隐藏 MS-OfficeWebserver 和 AspNet-Version:

```
location / {  
    proxy_hide_header X-AspNet-Version;  
    proxy_hide_header MicrosoftOfficeWebServer;  
}
```

13.20.10 proxy_ignore_client_abort 指令

语法: proxy_ignore_client_abort [on|off]

默认值: proxy_ignore_client_abort off

使用环境: http, server, location

如果客户端自身终止请求, 防止中断代理请求。

13.20.11 proxy_ignore_headers 指令

语法: proxy_ignore_headers name [name ...]

默认值: none

使用环境: http, server, location

Nginx 0.7.54 以上版本开始支持该指令。该指令可以禁止处理从代理服务器返回的 Header 行。它可以指定字符串, 例如: "X-Accel-Redirect"、"X-Accel-Expires"、"Expires" 或 "Cache-Control"。

13.20.12 proxy_intercept_errors 指令

语法: proxy_intercept_errors [on|off]

默认值: proxy_intercept_errors off

使用环境: http, server, location

该指令判断 Nginx 是否会拦截 HTTP 状态码为 400 及以上代码的应答。

默认情况下, 来自被代理服务器的所有应答将照常发送。

如果设定 proxy_intercept_errors on, Nginx 将会拦截 error_page 指令明确指定的错误状态码。如果来自被代理服务器的应答状态码不匹配一个 error_page 指令, 应答将被照常发送。

13.20.13 proxy_max_temp_file_size 指令

语法: proxy_max_temp_file_size size;

默认值: proxy_max_temp_file_size 1G;

使用环境: http, server, location, if

该指令用于设置当网页内容大于代理内存缓冲区的时候, 临时文件大小的最大值。如果文件大于这个值, 它将从 upstream 服务器同步地传递请求, 而不是缓冲到磁盘。

13.20.14 proxy_method 指令

语法: proxy_method [method]

默认值: None

使用环境: http, server, location

该指令允许代理额外的 HTTP 方法。HTTP 协议除了常用的 GET、POST 方法外, 还有一些其他的方法, 例如 Squid 缓存服务器使用的自定义 PURGE 方法。

注意: 当使用本指令时, Nginx 只允许单个 HTTP 方法参数, 所以目前还不能代理类似

Subversion 版本控制服务的请求。

proxy_method 指令使用示例：

```
proxy_method PROPFIND;
```

13.20.15 proxy_next_upstream 指令

语法：proxy_next_upstream

[error|timeout|invalid_header|http_500|http_502|http_503|http_504| http_404|off]

默认值：proxy_next_upstream error timeout

使用环境：http, server, location

该指令用于设置当在哪种情况下，将请求转发到下一台服务器。在 upstream 负载均衡代理服务器池中，假设后端的一台服务器无法访问或返回指定错误响应代码时，可以使用该指令将请求转发到池中的下一台服务器。

可以设置的错误响应参数如下：

error——如果连接服务器时、发送请求时、读取应答信息时发送错误。

timeout——如果连接服务器时、传递请求时、读取后端服务器应答信息时超时。

invalid_header——后端服务器返回一个空的或错误的应答时。

http_500——后端服务器返回 500 应答状态码时。500 状态码一般是 Nginx 配置、FastCGI 程序出错导致。

http_502——后端服务器返回 500 应答状态码时。502 状态码一般是后端服务器程序执行超时、无法访问导致。

http_503——后端服务器返回 503 应答状态码时。

http_504——后端服务器返回 504 应答状态码时。

http_404——后端服务器返回 500 应答状态码时。404 状态码一般是文件不存在导致。

off——禁止将请求转发到下一台后端服务器。

当没有任何数据传送到客户端时，将请求转发到下一台后端服务器才会生效。确切地说，如果在传送应答内容给客户端的过程中出错或超时，将不会尝试重新请求一个不同的后端服务器。

13.20.16 proxy_pass 指令

语法: proxy_pass URL

默认值: no

使用环境: location, if in location

该指令用于设置被代理服务器端口或套接字, 以及 URI。

可以使用域名 (IP 地址) 和端口来指定被代理的服务器, 例如:

```
proxy_pass http://localhost:8000/uri;
```

也可以使用 Unix 套接字来指定被代理的服务器, 例如:

```
proxy_pass unix:/path/to/backend.socket;
```

请求转发给后端服务器的 URI 部分, 取自 Nginx 代理服务器的 location 后面的 URI 路径部分。例如在 Nginx 代理服务器的以下配置中, 访问 /name/test/test.html 将反向代理访问 http://192.168.1.2/test/test.html 的内容:

```
location /name/ {  
    proxy_pass http://192.168.1.2;  
}
```

但是, 对于以上的规则有以下两个例外:

(1) 当以正则表达式的方式指定 location 时。

(2) 在 location 中使用 rewrite 规则改变 URI 时, 使用这个配置可以能够更精确地处理请求 (break):

```
location /name/ {  
    rewrite /name/([^/] +) /users?name=$1 break;  
    proxy_pass http://127.0.0.1;  
}
```

在这些 URI 的传递案例中没有使用映射。

此外, 须指明, 以下 URI 将使用同样的格式转发。在工作过程中:

两个或多个连续的斜线将被转换成一个斜线: "//" -- "/";

引用的当前目录会被删除: "/./" -- "/";

引用的主目录会被删除: "/dir/./" -- "/".

如果一个服务器上要传送未经处理的格式, 那么 proxy_pass 的参数则要使用不含 URI 的 URL, 例如:

```
location /some/path/ {
    proxy_pass http://127.0.0.1;
}
```

在 proxy_pass 指令中使用变量是一个特例，也可以使用 rewrite 和 proxy_pass 的组合：

```
location / {
    rewrite ^(.*)$ /VirtualHostBase/https/$server_name:443/some/path/
    VirtualHostRoot/$1 break;
    proxy_pass http://127.0.0.1:8080;
}
```

13.20.17 proxy_pass_header 指令

语法：proxy_pass_header the_name

使用环境：http, server, location

该指令允许转发为应答转发 Header 行。

```
location / {
    proxy_pass_header X-Accel-Redirect;
}
```

13.20.18 proxy_pass_request_body 指令

语法：proxy_pass_request_body [on | off] ;

默认值：proxy_pass_request_body on;

使用环境：http, server, location

13.20.19 proxy_pass_request_headers 指令

语法：proxy_pass_request_headers [on | off] ;

默认值：proxy_pass_request_headers on;

使用环境：http, server, location

13.20.20 proxy_redirect 指令

语法：proxy_redirect [default|off|redirect replacement]

默认值：proxy_redirect default

使用环境: http, server, location

该指令用于更改被代理服务器的应答 Header 头中的"Location"和"Refresh"。

我们假设被代理服务器返回的行是: Location: *http://localhost:8000/two/some/uri/*, 那么, 以下指令:

```
proxy_redirect http://localhost:8000/two/ http://frontend/one/;
```

会将此行重写为 Location: *http://frontend/one/some/uri/*。

在代替的行中可以不用写服务器的名称, 只用写相对路径就可以了, 例如:

```
proxy_redirect http://localhost:8000/two/ /;
```

这样, 就设置了服务器的基本名称和端口, 即使它不是 80 端口。

默认参数 "default", 取决于 location 和 proxy_pass 指令的值。

例如以下两个配置的效果是一样的, 这里的参数 "default" 的值, 取决于 proxy_pass 指令的参数 "*http://upstream:port/two/*" 和 location 指令的参数 "*/one/*" :

```
location /one/ {
    proxy_pass      http://upstream:port/two/;
    proxy_redirect  default;
}

location /one/ {
    proxy_pass      http://upstream:port/two/;
    proxy_redirect  http://upstream:port/two/ /one/;
}
```

在代替的行中, 可以使用一些 Nginx 变量, 例如:

```
proxy_redirect http://localhost:8000/ http://$host:$server_port/;
```

该指令有时候可以重复:

```
proxy_redirect default;
proxy_redirect http://localhost:8000/ /;
proxy_redirect http://www.example.com/ /;
```

参数 off 在这个使用环境 (例如 location /one/ {.....}) 中禁止所有的 proxy_redirect 指令:

```
proxy_redirect off;
proxy_redirect default;
proxy_redirect http://localhost:8000/ /;
proxy_redirect http://www.example.com/ /;
```

以下这条语句可以帮助被代理服务器为相对重定向添加主机名:

```
proxy_redirect / /;
```

13.20.21 proxy_read_timeout 指令

语法: proxy_read_timeout the_time

默认值: proxy_read_timeout 60

使用环境: http, server, location

该指令用于设置从后端被代理服务器读取应答内容的超时时间。它决定 Nginx 等待多长时间来取得一个请求的应答。超时时间是指完成 TCP 两次握手, 到 TCP 状态为 ESTABLISHED 时(数据正在传输)的时间, 而不是整个时间。

注意不要将此时间设置得太低, 如果被代理服务器超过此时间, 仍然没有传输数据, Nginx 将关闭该次连接。

13.20.22 proxy_redirect_errors 指令

目前, 已经不赞成使用该指令。请使用 proxy_intercept_errors 指令。

13.20.23 proxy_send_lowat 指令

语法: proxy_send_lowat [on | off]

默认值: proxy_send_lowat off;

使用环境: http, server, location, if

该指令用于设置是否使用 SO_SNDLOWAT, 该指令只可以在 FreeBSD 操作系统中使用。

13.20.24 proxy_send_timeout 指令

语法: proxy_send_timeout time_in_seconds

默认值: proxy_send_timeout 60

使用环境: http, server, location

该指令用于指定代理服务器转发请求的超时时间。超时时间是指完成 TCP 两次握手, 到 TCP 状态为 ESTABLISHED 时(数据正在传输)的时间, 而不是整个时间。如果代理服务器超过此时间, 仍然没有转发数据到后端服务器, Nginx 将关闭该次连接。

13.20.25 proxy_set_body 指令

语法: proxy_set_body [on | off]

默认值: proxy_set_body off;

使用环境: http, server, location, if

从 Nginx 0.3.10 版本开始, 可以使用该指令。

13.20.26 proxy_set_header 指令

语法: proxy_set_header header value

默认值: Host and Connection

使用环境: http, server, location

该指令允许重新定义或添加 Header 行到转发给被代理服务器的请求信息中, 它的值可以是文本, 也可以是变量, 或者是文本和变量的组合。

当 proxy_set_header 指令没有对指定的 Header 行进行定义时, 将从客户端请求的 Header 行中继承这些行的信息。默认情况下, 只有以下两行可以被重新定义:

```
proxy_set_header Host $proxy_host;  
proxy_set_header Connection Close;
```

未修改的 Header 头 “Host” 能够按照如下方式发送:

```
proxy_set_header Host $http_host;
```

然而, 如果客户端请求中没有这个 Header 头, 将没有 “Host” 行的数据发送给被代理服务器。在这种情况下, 最好使用 \$host 变量, 它的值相当于服务器的主机名 (如果使用域名访问, 则该值为域名; 如果使用 IP 访问, 则该值为 IP):

```
proxy_set_header Host $host;
```

此外, 可以将主机名和被代理服务器的端口一起传递:

```
proxy_set_header Host $host:$proxy_port;
```

如果设置指定 Header 行的值为空, 该 Header 行将不会被发送到 upstream。例如, 以下示例可以用来在 upstream 中禁止后端服务器使用 gzip 压缩:

```
proxy_set_header Accept-Encoding "";
```

13.20.27 proxy_store 指令

语法: proxy_store [on | off | path]

默认值: proxy_store off

使用环境: http, server, location

该指令可以设置哪些从后端服务器传送过来的文件被 Nginx 存储。参数“on”保持文件与 alias 或 root 指令设置的目录一致，参数“off”不存储文件。此外，路径名称中可以使用变量，例如：

```
proxy_store /data/www$original_uri;
```

文件的最后修改时间将被在应答数据的“Last-Modified”Header 头中设定。为了文件的安全，可以使用 proxy_temp_path 指令设置一个临时目录。

该指令可以为不经常修改的文件建立一份本地镜像，从而减轻后端被代理服务器的压力。使用示例如代码 13-21 所示：

代码 13-21

```
location /images/ {
    root          /data/www;
    error_page    404 = /fetch$uri;
}

location /fetch {
    internal;
    proxy_pass    http://backend;
    proxy_store   on;
    proxy_store_access user:rw group:rw all:r;
    proxy_temp_path /data/temp;
    alias         /data/www;
}
```

或者代码 13-22 中的内容：

代码 13-22

```
location /images/ {
    root          /data/www;
    error_page    404 = @fetch;
}

location @fetch {
    internal;

    proxy_pass    http://backend;
    proxy_store   on;
```

```
proxy_store_access user:rw group:rw all:r;  
proxy_temp_path    /data/temp;  
  
root                /data/www;  
}
```

注意: `proxy_store` 不是缓存, 存储的文件不会自动过期, 它相当于一个镜像。

13.20.28 proxy_store_access 指令

语法: `proxy_store_access users:permissions [users:permission ...]`

默认值: `proxy_store_access user:rw`

使用环境: `http, server, location`

该指令用于指定创建文件和目录的权限, 例如:

```
proxy_store_access user:rw group:rw all:r;
```

如果指定的用户组是正确的, 那么, 不需要指定用户的权限, 例如:

```
proxy_store_access group:rw all:r;
```

13.20.29 proxy_temp_file_write_size 指令

语法: `proxy_temp_file_write_size size;`

默认值: `proxy_temp_file_write_size ["#proxy buffer size"] * 2;`

使用环境: `http, server, location, if`

设置写入 `proxy_temp_path` 临时目录的数据大小。它可以防止一个工作进程阻塞太长时间。

13.20.30 proxy_temp_path 指令

语法: `proxy_temp_path dir-path [level1 [level2 [level3]] ;`

默认值: `$NGX_PREFIX/proxy_temp` controlled by `--http-proxy-temp-path` at `./configure` stage

使用环境: `http, server, location`

该指令工作方式类似 `client_body_temp_path`, 用于指定一个本地目录来缓冲较大的代理请求。



13.20.31 proxy_upstream_fail_timeout 指令

从 Nginx 0.5.0 版本开始，不赞成使用该指令。请使用 upstream 模块中的 fail_timeout 参数来代替它。

13.20.32 proxy_upstream_max_fails 指令

从 Nginx 0.5.0 版本开始，不赞成使用该指令。请使用 upstream 模块中的 max_fails 参数来代替它。

13.20.33 HTTP Proxy 模块的变量

HTTP Proxy 模块包含一些内置变量，可以用于 proxy_set_header 指令：

`$proxy_host`

被代理服务器的主机名和端口。

`$proxy_port`

被代理服务器的端口。

`$proxy_add_x_forwarded_for`

包含以半角逗号分割的客户端请求头“X-Forwarded-For”和\$remote_addr 的内容。如果没有请求头“X-Forwarded-For”，该变量的值等于\$remote_addr 的值。

13.21 HTTP Rewrite 模块

该模块及相关指令已经在第 7 章中有详细说明，这里就不再重复介绍。

13.22 HTTP SSI 模块

SSI（Server Side Include），通常称为服务器端嵌入。

SSI 的工作原理：将内容发送到浏览器之前，可以使用“服务器端包含（SSI）”指令将文本、图形或应用程序信息包含到网页中。例如，可以使用 SSI 包含时间/日期戳、版权声明或供客户填写并返回的表单。对于在多个文件中重复出现的文本或图形，使用包含文件是一种简便的方法。将内容存入一个包含文件中即可，而不必将内容输入所有文件。通过一个非常简单的语句

即可调用包含文件，此语句指示 Web 服务器将内容插入适当网页。而且，使用包含文件时，对内容的所有更改只需在一个地方就能完成。

因为包含 SSI 指令的文件要求特殊处理，所以必须为所有 SSI 文件赋予 SSI 文件扩展名。默认扩展名是“.shtml”。例如以下的网址，则使用了 SSI：

`http://finance.sina.com.cn/g/20091222/16047138654.shtml`

Nginx 支持 SSI，但是，目前功能并不是十分完善。

13.22.1 ssi 指令

语法：ssi [on | off]

默认值：ssi off

使用环境：http, server, location, if in location

该指令用于开启或关闭 SSI 处理。

13.22.2 ssi_silent_errors 指令

语法：ssi_silent_errors [on|off]

默认值：ssi_silent_errors off

使用环境：http, server, location

该指令设置为 off 时，当处理 SSI 时发生错误，不输出 “[an error occurred while processing the directive]” 错误信息。

13.22.3 ssi_types 指令

语法：ssi_types mime-type [mime-type ...]

默认值：ssi_types text/html

使用环境：http, server, location

允许 SSI 处理其他 MIME 类型。SSI 默认可以处理 “text/html” 类型。

Enables SSI processing for MIME-types in addition to "text/html" types.

13.22.4 ssi_value_length 指令

语法: `ssi_value_length length`

默认值: `ssi_value_length 256`

使用环境: `http, server, location`

定义 SSI 中允许使用的参数长度。

13.22.5 SSI 命令

指令格式如下:

```
<!--# command parameter1=value parameter2=value...-->
```

支持的 SSI 命令列表如下:

1. block

该命令允许创建一个块, 在块中可以使用 SSI 命令:

name——块的名称。示例:

```
<!--# block name="one" -->
the silencer
<!--# endblock -->
```

2. config

为 SSI 配置一些参数。

- `errmsg`——SSI 处理过程中的错误行, 默认字符串为: `[an error occurred while processing the directive]`。
- `timefmt`——时间字符串的格式, `strftime(3)` 函数使用, 默认为:

```
"%A, %d-%b-%Y %H:%M:%S %Z"
```

时间中的秒可以使用 `"%s"`。

3. echo

打印一个变量。

- `var`——变量的名称。
- `default`——如果变量为空, 则显示这个字符串, 默认值为 `"none"`。示例:

```
<!--# echo var="name" default="no" -->
```

与以下写的功能相似:

```
<!--# if expr="$name" --><!--# echo var="name" --><!--# else -->no<!--# endif -->
```

4. if/elif/else/endif

使用方法:

```
<!--# if expr="..." -->
...
<!--# elif expr="..." -->
...
<!--# else -->
...
<!--# endif -->
```

以上示例只允许一层嵌套, 不允许多层嵌套。

- **expr**——匹配一个表达式, 可以是变量:

```
<!--# if expr="$name" -->
```

字符串比较:

```
<!--# if expr="$name = text" -->
<!--# if expr="$name != text" -->
```

正则匹配:

```
<!--# if expr="$name = /text/" -->
<!--# if expr="$name != /text/" -->
```

如果使用变量, 则用它们的值代替。

5. include

包含一个其他来源的文档。

- **file**——包含一个文件, 例如:

```
<!--# include file="footer.html" -->
```

- **virtual**——包含一个请求, 例如:

```
<!--# include virtual="/remote/body.php?argument=value" -->
```

多个请求将并行执行, 如果要按顺序执行, 请使用 “wait” 选项。

- **stub**——如果请求为空或返回一个错误后使用的默认块。

```
<!--# block name="one" --> <!--# endblock -->
<!--# include virtual="/remote/body.php?argument=value" stub="one" -->
```

- **wait**——当设置为 yes 时, 在当前的请求完成之前, 剩余的 SSI 不会判定, 示例:

```
<!--# include virtual="/remote/body.php?argument=value" wait="yes" -->
```

6. set

设置一个变量。

- var——变量。
- value——包含变量名变量的值，它们将被匹配。

13.22.6 SSI 变量

SSI 变量代码格式如下：

```
$date_local
```

本地时区的当前时间，选项 "timefmt" 控制格式。

```
$date_gmt
```

当前的格林尼治时间，选项 "timefmt" 控制格式。

13.23 HTTP Userid 模块

该模块相当于 Apache 的 mod_uid 模块，主要用于做客户端的身份标识。它主要使用 \$uid_get 和 \$uid_set 变量。

配置示例：

```
userid      on;
userid_name  uid;
userid_domain example.com;
userid_path  /;
userid_expires 365d;
userid_p3p    'policyref="/w3c/p3p.xml", CP="CUR ADM OUR NOR STA NID"';
```

13.23.1 userid 指令

语法：userid [only|log|loff]

默认值：userid off

使用环境：http, server, location

该指令允许或禁止发布 cookie 及记录请求的 cookie：

on——允许使用版本 2 的 cookie 及记录它们。

v1——允许使用版本 1 的 cookie 及记录它们。

log——不发送 cookie，但是写下 cookie 来记录。

off——禁止发送及写 cookie。

13.23.2 userid_domain 指令

语法: `userid_domain [name | none]`

默认值: `userid_domain none`

使用环境: `http, server, location`

指定 cookie 的域名。参数 “none” 不为 cookie 指定域名。

13.23.3 userid_expires 指令

语法: `userid_expires [time | max]`

默认值: `none`

使用环境: `http, server, location`

设置 cookie 的过期时间。

参数设置并发出浏览器 cookie 的过期时间。参数值 “max” 指定过期时间为 “2037 年 12 月 31 日 23:55:55 GMT”，这是一些老的浏览器能够识别的最大时间。

13.23.4 userid_name 指令

语法: `userid_name name`

默认值: `userid_name uid`

使用环境: `http, server, location`

该指令用于指定 cookie 的名称。

13.23.5 userid_p3p 指令

语法: `userid_p3p line`

默认值: none

使用环境: http, server, location

Header 头 “P3P” 的值, 将和 cookie 一起传递。

13.23.6 userid_path 指令

语法: userid_path path

默认值: userid_path /

使用环境: http, server, location

设置 cookie 的路径。

13.23.7 userid_service 指令

语法: userid_service number

默认值: userid_service address

使用环境: http, server, location

该指令用于设置发出 cookie 的服务器 IP 地址。如果不设置。版本 1 的 cookie 设置为 0, 版本 2 的 cookie 设置为服务器的 IP。