

第 14 章

Nginx 的其他 HTTP 模块

本章介绍的 HTTP 模块不会在编译 Nginx 时自动编译进来，除非使用 “./configure --with-模块名” 命令指定编译这些模块到 Nginx 中。

在本章的指令介绍中，指令的“使用环境”是指该指令可以在 Nginx 配置文件中使用的位置，例如使用环境为 “http, server, location”，则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。

14.1 HTTP Addition 模块

本模块可以在当前的 location 内容之前或之后添加其他的 location 内容。

它作为一个输出过滤器被执行，主请求的内容和到其他 location 的内容不会被完全缓冲，将仍以流处理的方式发送到客户端。因为当发送 HTTP Header 头信息时，最终响应 Body 主体的长度是未知的，所以这里 Nginx 将不会在 Header 头中提供 Content-Length 头信息，而是始终采用 HTTP Chunked 编码动态地提供 body 内容的长度。进行 Chunked 编码传输的 HTTP 响应会在 Header 头中设置：Transfer-Encoding: chunked，表示 Body 主体将用 Chunked 编码传输内容。

本模块默认是不会编译进 Nginx 的，如果你要使用该模块，则要在编译安装 Nginx 时指定：

```
./configure --with-http_addition_module
```

本模块相关指令的使用示例如下：

```
location / {  
    add_before_body    /before_action;  
    add_after_body     /after_action;  
}
```

限制项如下：

在 Nginx 0.8.17 中，如果当前的 location 为它自身的子请求提供服务，该子请求的内容将不会被添加，例如代码 14-1 示例：

代码 14-1

```
location /foo {  
    add_before_body /bar;  
}  
  
location /bar {  
    add_before_body /baz;  
}
```

访问/foo 不会使用/baz 代替子请求/bar 之前的内容。另外，在 before/after body location 中，只能使用字符串，而不能使用变量。以下配置虽然可以在 nginx.conf 配置文件检测时通过，但是不能正常工作。

```
location / {  
    set $before_action /before_action;  
    add_before_body $before_action;  
}
```

14.1.1 add_before_body 指令

语法：add_before_body uri

默认值：no

使用环境：http, server, location

该指令用于在应答主体之前添加 URI 子请求的内容结果。

14.1.2 add_after_body 指令

语法：add_after_body uri

默认值：no

使用环境：http, server, location

该指令用于在应答主体之后添加 URI 子请求的内容结果。

14.1.3 addition_types 指令

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

默认值: `text/html`

使用环境: `http, server, location`

该指令(从 Nginx 0.7.9 版本开始)允许 `location` 处理自定义的 MIME 类型(默认为“`text/html`”)。

在 Nginx 0.8.17 之前的版本,该指令在源代码中被错误地拼写为“`addtion_types`”,这个 Bug 在 0.8.17 版本中被修正。

14.2 Embedded Perl 模块

本模块允许在 Nginx 中直接执行 Perl,或者通过 SSI 调用 Perl。

本模块默认是不会编译进 Nginx 的,如果你要使用该模块,则要在编译安装 Nginx 时指定:

```
./configure --with-http_perl_module
```

另外,您的操作系统中必须安装 Perl 5.6.1 以上版本。

已知问题:

(1) 如果 Perl 模块执行长时间的操作,例如 DNS 查询、数据库查询等,运行 Perl 脚本的工作进程将一直处于阻塞状态,因此,内置的 Perl 脚本应该非常简单,执行尽可能快。

(2) Nginx 在通过“`kill -HUP <pid>`”命令重新加载配置文件时,可能会导致内存泄露。

示例如代码 14-2 所示:

代码 14-2

```
http {
    perl_modules perl/lib;
    perl_require hello.pm;

    perl_set $msie6 '
sub {
    my $r = shift;
    my $ua = $r->header_in("User-Agent");
    return "" if $ua =~ /Opera/;
    return "1" if $ua =~ / MSIE [6-9] \.\d+/;
    return "";
}
```

```
}  
';  
  
server {  
    location / {  
        perl hello::handler;  
    }  
}  
}
```

perl/lib/hello.pm 示例如代码 14-3 所示:

代码 14-3

```
package hello;  
use nginx;  
  
sub handler {  
    my $r = shift;  
    $r->send_http_header("text/html");  
    return OK if $r->header_only;  
  
    $r->print("hello!\n<br/>");  
    $r->rflush;  
  
    if (-f $r->filename or -d _) {  
        $r->print($r->uri, " exists!\n");  
    }  
  
    return OK;  
}  
  
1;  
__END__
```

14.2.1 perl 指令

语法: perl module::function | 'sub {...}'

默认值: no

使用环境: location

该指令用来指定数据 location 中须要用到的 Perl 函数。

14.2.2 perl_modules 指令

语法: perl_modules path

默认值: no

使用环境: http

该指令用于为 Perl 模块指定额外的路径。从 Nginx 0.6.7 版本开始, 该路径与 nginx.conf 配置文件所在目录有关, 而不是 Nginx 的 prefix 安装目录。

14.2.3 perl_require 指令

语法: perl_require module

默认值: no

使用环境: http

这里, 你可以使用多个 perl_require 指令。

14.2.4 perl_set 指令

语法: perl_set module::function | 'sub {...}'

默认值: no

使用环境: http

该指令用于设置一个名称为变量名的函数体。示例如代码 14-4:

代码 14-4

```
.....
http {
    perl_set $path_md5 '
        use Digest::MD5 qw(md5_hex);
        use File::stat;

        sub {
            my $r = shift;
            my $s = md5_hex($r->uri);
            my $path_md5 = join "", join("/", substr($s, 0, 1), substr($s, 1, 1),
                substr($s, 2)), ".html";
            my $filepath = "/data/www/" . $path_md5;
            if(-f $filepath) {
                my $mtime = stat($filepath)->mtime;
                if(time() - $mtime > 1800) {
                    return $path_md5 . ".new";
                }
            }
            return $path_md5;
        }
    }
}
```

```

    }';

server {
    listen 80;
    server_name 127.0.0.1;
    index index.html;
    root /data/www;
    location /images/ {
        try_files $path_md5 @fastcgi;
    }
    .....
}
}

```

14.2.5 从 SSI 调用 Perl 脚本

指令格式如下：

```
<!-- # perl sub="module::function" arg="parameter1" arg="parameter2"... >
```

请求对象的方法\$r：

\$r->args——返回请求参数的方法。

\$r->discard_request_body——告诉 Nginx 丢弃请求主体的方法。

\$r->filename——返回与 URI 请求想符合的文件名。

\$r->has_request_body (function) ——如果没有请求的主体，则返回 0。如果请求的主体存在，则建立传递的函数并返回 1。在主体处理的结尾，Nginx 将调用已经建立的处理器。用法示例如代码 14-5：

代码 14-5

```

package hello;

use nginx;

sub handler {
    my $r = shift;

    if ($r->request_method ne "POST") {
        return DECLINED;
    }

    if ($r->has_request_body(\&post)) {
        return OK;
    }

    return 400;
}

```

```

}

sub post {
    my $r = shift;
    $r->send_http_header;
    $r->print("request_body: \"", $r->request_body, "\"<br/>");
    $r->print("request_body_file: \"", $r->request_body_file, "\"<br/>\n");
    return OK;
}

1;

__END__

```

`$r->header_in (header)` ——检索一个 HTTP 请求头。

`$r->header_only` ——如果只须要返回一个应答头，则为真。

`$r->header_out (header, value)` ——设置一个应答头。

`$r->internal_redirect (uri)` ——使用内部重定向到指定的 URI。重定向只能在 Perl 脚本执行结束后发生。

`$r->print (args, ...)` ——发送数据到客户端。

`$r->request_body` ——当主体没有记录进临时文件时，返回客户端请求主体。因此，为了保证客户端请求主体保留在内存中，须要使用 `client_max_body_size` 指令限制其大小，并且使用 `client_body_buffer_size` 指令指定足够大的缓冲区。

`$r->request_body_file` ——返回存储客户端需求主体的文件名。该文件在处理完成后必须被删除。如果需要请求主体总是写入该文件，你须要设置 `client_body_in_file_only` 指令为 on。

`$r->request_method` ——返回 HTTP 请求方法。

`$r->remote_addr` ——返回客户端 IP 地址。

`$r->rflush` ——马上发送数据到客户端。

`$r->sendfile (file [, displacement [, length])` ——传输文件的内容给客户端显示。可选的参数用来设置初始位置点及传送数据的长度。严格来说，数据传输只能发生在 Perl 脚本执行完毕后。

`$r->send_http_header (type)` ——添加 Header 到应答中。参数选项 “type” 用来填写 Header 行 “Content-Type” 的值。

`$r->sleep (milliseconds, handler)` ——设置在给定请求时间内的指定处理程序，或者停止处理。在这段时间内，Nginx 将继续处理其他请求。在规定的超时时间期满后，Nginx 将运行安装处理程序。请注意，你须要传递一个标识符给函数处理程序。在处理器之间传输数据，你应该使

用 `$r->variable()`。用法示例如代码 14-6:

代码 14-6

```
package hello;

use nginx;

sub handler {
    my $r = shift;

    $r->discard_request_body;
    $r->variable("var", "OK");
    $r->sleep(1000, \&next);

    return OK;
}

sub next {
    my $r = shift;

    $r->send_http_header;
    $r->print($r->variable("var"));

    return OK;
}

1;
__END__
```

`$r->status (code)` —— 设置 HTTP 应答头。

`$r->unescape (text)` —— 使用 `unescape` 方法对以 %XX 十六进制形式编码的文本进行解码。

`$r->uri` —— 返回请求的 URI。

`$r->variable (name[, value])` —— 返回或设置指定变量的值。对每个查询来说，变量为局部变量。

14.3 Flv Stream 模块

本模块用于 Flash 播放器以 HTTP 下载方式播放远程 Web 服务器上的 FLV 视频时，支持播放进度条拖动，即支持以 `test.flv?start=12345` 的方式从指定字节位置下载文件。国内著名的视频分享网站优酷网、土豆网、新浪播客等，都支持这种拖动方式。

本模块默认是不会编译进 Nginx 的，如果你要使用该模块，则要在编译安装 Nginx 时指定：

```
./configure --with-http_flv_module
```


本模块的配置示例如下：

```
location ~ /\.flv$ {
    flv;
}
```

在前面第 11 章第 11.2.1 节中，已详细介绍了“采用 Nginx 的 Flv Stream 模块搭建 HTTP 下载方式的 FLV 视频服务器”，这里不再赘述。

14.3.1 flv 指令

语法：flv

默认值：None

使用环境：location

为当前的 location 开启针对 FLV 视频拖动播放的特殊文件处理。

14.4 HTTP Gzip Static 模块

在一个文件传送给能够接受 Gzip 压缩的客户端浏览器之前，本模块用来检查同名 location 下是否存在以“.gz”结尾的文件。这样做的意图是避免重复压缩相同的文件。

本模块从 Nginx 0.6.24 版本开始被引入，默认是不会编译进 Nginx 的，如果你要使用该模块，则要在编译安装 Nginx 时指定：

```
./configure --with-http_gzip_static_module
```

示例代码如下：

```
gzip_static on;

gzip_http_version 1.1;
gzip_proxied expired no-cache no-store private auth;
gzip_disable "MSIE [1-6]\.";
gzip_vary on;
```

14.4.1 gzip_static 指令

语法：gzip_static on|off

默认值：gzip_static off

使用环境：http, server, location

开启 HTTP Gzip Static 模块。你应该确保压缩文件和未压缩文件的时间戳一样。

14.4.2 gzip_http_version 指令

参见第 13 章第 13.11.5 节的 gzip_http_version 指令。

14.4.3 gzip_proxied 指令

参见第 13 章第 13.11.6 节的 gzip_proxied 指令。

14.5 HTTP Random Index 模块

本模块用于从目录中选择一个随机目录索引。

本模块默认是不会编译进 Nginx 的，如果你要使用该模块，则要在编译安装 Nginx 时指定：

```
./configure --with-http_random_index_module
```

配置示例代码如下：

```
location / {  
    random_index on;  
}
```

14.5.1 random_index 指令

语法：random_index [on|off]

默认值：off

使用环境：location

如果在一个指定的 location 中开启本指令，它将为每次访问扫描指定目录内的文件，并发送一个随机选取的文件代替通常的 index.html。以 “.” 开头的文件不会被选取。

14.6 HTTP Geo IP 模块

本模块基于客户端 IP 地址与 MaxMind()提供的 GeoIP 地址库进行比对，创建一些变量，用来实现地区性负载均衡。本模块适用于 Nginx 0.7.63 和 0.8.6 之后的版本。

本模块需要 Geo IP 数据库及读取该数据库的 libgeoip 类库，如代码 14-7 所示。

代码 14-7

```
#下载免费的 geo 城市 IP 数据库
wget http://geolite.maxmind.com/download/geoip/database/GeoLiteCity.dat.gz
#下载免费的 geo 国家 IP 数据库
wget http://geolite.maxmind.com/download/geoip/database/GeoLiteCountry/
GeoIP.dat.gz
#下载 libgeoip，在 Debian Linux 操作系统，你可以使用以下方式安装：
sudo apt-get install libgeoip-dev
#在其他 Linux 操作系统，你可以下载源码自己编译安装
wget http://geolite.maxmind.com/download/geoip/api/c/GeoIP.tar.gz
```

在 CentOS Linux 操作系统，你可以使用 yum 命令安装 libgeoip：

```
yum install geoip-devel
```

本模块默认是不会编译进 Nginx 的，如果你要使用该模块，则要在编译安装 Nginx 时指定：

```
./configure --with-http_geoip_module
```

配置示例代码如下：

```
http {
    geoip_country GeoIP.dat;
    geoip_city    GeoLiteCity.dat;
    ...
}
```

14.6.1 geoip_country 指令

语法：geoip_country path/to/db.dat;

默认值：none

使用环境：http

该指令用于指定判断访问者 IP 地址所属国家的.dat 数据库文件路径。通过该指令的设置，GeoIP 模块会创建以下可用的变量：

\$geoip_country_code;——两个字母的国家代码，例如“RU”，“US”。

\$geoip_country_code3;——三个字母的国家代码，例如“RUS”，“USA”。

\$geoip_country_name;——国家名称，例如：“Russian Federation”，“United States”。

如果你只需要国家名称，则只设置 geoip_country 数据库（1.1MB）即可，因为 geoip_city 数据库（43MB）要大得多，并且所有的数据库将被缓存在内存中。

14.6.2 geoip_city 指令

语法: `geoip_city path/to/db.dat;`

默认值: `none`

使用环境: `http`

该指令用于指定判断访问者 IP 地址所属国家、地区、城市的 .dat 数据库文件路径。通过该指令的设置, GeoIP 模块会创建以下可用的变量:

`$geoip_city_country_code`——两个字母的国家代码, 例如 “RU”, “US”。

`$geoip_city_country_code3`——三个字母的国家代码, 例如 “RUS”, “USA”。

`$geoip_city_country_name`——国家名称, 例如: “Russian Federation”, “United States” (if available)。

`$geoip_region`——地区名称(省, 地区, 州, 大行政区, 联邦政府国土等)例如: “Moscow City”, “DC” (如果存在)。

`$geoip_city`——城市名称, 例如: “Moscow”, “Washington” (如果存在)。

`$geoip_postal_code`——邮政编码(如果存在)。

`$geoip_city_continent_code`——洲(指欧、亚、非、南北美、澳、南极洲之一)代码(如果存在)。

`$geoip_latitude`——纬度(如果存在)。

`$geoip_longitude`——经度(如果存在)。

14.7 HTTP RealIP 模块

本模块可以修改客户端请求头中的客户端 IP 地址, 例如 X-Real-IP 和 X-Forwarded-For。如果 Nginx 位于 Squid 等代理服务器之后, 或者位于 F5 Big-IP、NetScaler 等七层负载均衡交换机之后, 本模块将非常有用。

本模块默认是不会编译进 Nginx 的, 如果你要使用该模块, 则要在编译安装 Nginx 时指定:
`./configure --with-http_realip_module`

假设一台 Nginx 位于 Squid 的后端, Squid 设置了 X-Forwarded-For 头信息, 记录了用户的真实 IP, 但是, 如果存在多级代理的情况, X-Forwarded-For 头信息的内容值将变成以逗号分隔的

多个 IP 地址。

```
X-Forwarded-For: 202.108.1.1, 192.168.1.5, 192.168.1.6, 192.168.2.1
```

通过本模块，你可以设置一个可信的代理服务器 IP 列表，Header 头中第一个不受信任的 IP 将作为客户端 IP。例如以上的 X-Forwarded-For 信息中，假设 202.108.1.1 为客户端 IP，其他均为代理服务器 IP，就可以使用以下配置：

```
set_real_ip_from 192.168.1.0/24;  
set_real_ip_from 192.168.2.1;  
real_ip_header X-Forwarded-For;
```

这样，X-Forwarded-For 中的 IP 信息就只剩下客户端的 IP 地址 202.108.1.1 了。

14.7.1 set_real_ip_from 指令

语法：set_real_ip_from [the address|CIDR]

默认值：none

使用环境：http, server, location

该指令用于设置可信的代理服务器 IP 地址，这些 IP 地址将在请求转发时被从 Header 头信息中去掉。

14.7.2 real_ip_header 指令

语法：real_ip_header [X-Real-IP|X-Forwarded-For]

默认值：real_ip_header X-Real-IP

使用环境：http, server, location

该指令用于设置转发客户端 IP 地址的 Header 头名称。

14.8 HTTP SSL 模块

本模块用于 HTTPS 支持。

它支持使用以下两个限制检查客户端证书：

- (1) 不允许指定过期证书的列表。
- (2) 如果你有一个证书链文件，你无须像 Apache 那样指定每个证书文件。例如金山逍遥网

用户中心使用的中国互联网信息中心 (CNNIC) SSL 证书, 则是由三个证书链文件构成的。根证书由 Entrust.net 颁发给自己, 中级根证书由 Entrust.net 颁发给 CNNIC, 域名 my.xoyo.com 的 CRT 证书文件由 CNNIC 颁发, 如图 14-1 所示。

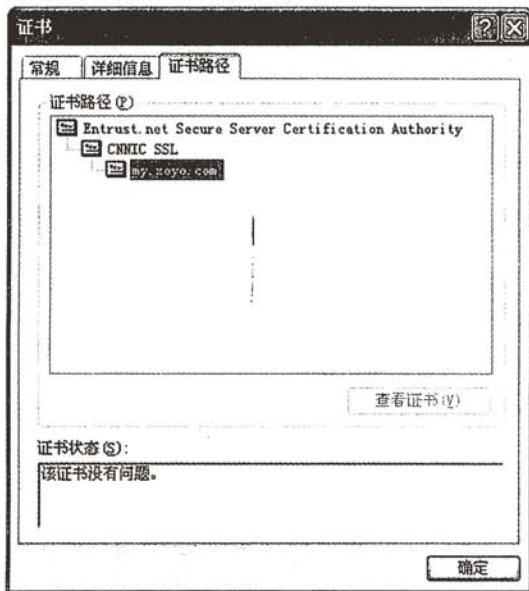


图 14-1 Windows 上查看证书链文件

将三个证书文件的内容 (文本格式) 按照低级别证书在前的方式, 拷贝到一个证书链文件 (文本格式) 中, 即可构成证书链文件, Nginx 配置文件中只须要指定这个证书链文件即可。证书链格式如代码 14-8 所示:

代码 14-8

```
-----BEGIN CERTIFICATE-----  
根证书内容  
-----END CERTIFICATE-----  
-----BEGIN CERTIFICATE-----  
中级根证书内容  
-----END CERTIFICATE-----  
-----BEGIN CERTIFICATE-----  
my.xoyo.com 域名证书内容  
-----END CERTIFICATE-----
```

本模块默认是不会编译进 Nginx 的, 如果你要使用该模块, 则要在编译安装 Nginx 时指定:
`./configure --with-http_ssl_module`

配置示例如代码 14-9:

代码 14-9

```
http {
    server {
        listen          443;
        ssl              on;
        ssl_certificate  /usr/local/nginx/conf/cert.pem;
        ssl_certificate_key /usr/local/nginx/conf/cert.key;
        keepalive_timeout 70;
    }
}
```

从 Nginx 0.7.14 版本开始，通常习惯在 `listen` 指令中使用“`ssl`”参数：

```
server {
    listen 443 ssl;
    ssl_certificate      /usr/local/nginx/conf/cert.pem;
    ssl_certificate_key  /usr/local/nginx/conf/cert.key;
    ...
}
```

生成证书：

通常情况下，你可以按照以下步骤生成一个自行颁发的证书，如代码 14-10 所示：

代码 14-10

```
cd /usr/local/nginx/conf
openssl genrsa -des3 -out server.key 1024
openssl req -new -key server.key -out server.csr
cp server.key server.key.org
openssl rsa -in server.key.org -out server.key
openssl x509 -req -days 365 -in server.csr -signkey server.key -out server.crt
```

在 Nginx 配置文件中配置新证书，如代码 14-11：

代码 14-11

```
server {
    server_name YOUR_DOMAINNAME_HERE;
    listen 443;
    ssl on;
    ssl_certificate /usr/local/nginx/conf/server.crt;
    ssl_certificate_key /usr/local/nginx/conf/server.key;
}
```

重启 Nginx，然后访问：

`https://YOUR_DOMAINNAME_HERE`

在第 11 章，已介绍了使用 HTTPS（SSL）构建一个安全的 Nginx Web 服务器实例。

14.8.1 在多个 server{.....}虚拟主机中使用通配符 SSL 证书

如果你有一个通配符 SSL 证书,例如颁发给*.nginx.org 域名的 SSL 证书,你可以在 http{.....} 中指定证书文件和私钥文件,那么在各虚拟主机中就可以继承相同的证书。配置示例如代码 14-12:

代码 14-12

```
ssl_certificate      common.crt;
ssl_certificate_key  common.key;

server {
    listen            80;
    server_name      www.nginx.org;
    ...
}

server {
    listen            443 ssl;
    server_name      secure.nginx.org;
    ...
}

server {
    listen            80;
    listen            443;
    server_name      images.nginx.org;
    ...
}
```

14.8.2 ssl 指令

语法: ssl [on|off]

默认值: ssl off

使用环境: main, server

为一个 server{.....}虚拟主机开启 HTTPS (SSL) 支持。

14.8.3 ssl_certificate 指令

语法: ssl_certificate file

默认值: ssl_certificate cert.pem

使用环境: main, server

为当前的虚拟主机指定 PEM 格式的证书文件。一个文件中可以包含多个证书文件(证书链), 同样, 密钥也必须为 PEM 格式。从 Nginx 0.6.7 版本开始, 证书文件的相对路径为 nginx.conf 配置文件所在的目录, 而不是 Nginx 安装目录。

14.8.4 ssl_certificate_key 指令

语法: ssl_certificate_key file

默认值: ssl_certificate_key cert.pem

使用环境: main, server

为当前的虚拟主机指定 PEM 格式的私钥文件。从 Nginx 0.6.7 版本开始, 证书文件的相对路径为 nginx.conf 配置文件所在的目录, 而不是 Nginx 安装目录。

14.8.5 ssl_client_certificate 指令

语法: ssl_client_certificate file

默认值: none

使用环境: main, server

指定 PEM 格式的 CA 证书, 用于检查客户端证书。

14.8.6 ssl_dhparam 指令

语法: ssl_dhparam file

默认值: none

使用环境: main, server

指定 PEM 格式的含有 Diffie-Hellman 参数的文件, 用于 TLS 会话键。

14.8.7 ssl_ciphers 指令

语法: ssl_ciphers file

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

使用环境: main, server

指定许可密码的描述。密码以 OpenSSL 支持的格式指定, 例如:

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

使用以下命令可以查看 OpenSSL 支持的完整格式列表:

```
openssl ciphers
```

14.8.8 ssl_crl 指令

语法: ssl_crl file

默认值: none

使用环境: http, server

该指令 (Nginx 0.8.7 以上版本开始支持) 用于指定一个 PEM 格式的证书吊销文件, 用于检查客户端证书。

14.8.9 ssl_prefer_server_ciphers 指令

语法: ssl_prefer_server_ciphers [on|off]

默认值: ssl_prefer_server_ciphers off

使用环境: main, server

对 SSLv3 和 TLSv1 协议的服务器端密码需求优先级高于客户端密码。

14.8.10 ssl_protocols 指令

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

默认值: ssl_protocols SSLv2 SSLv3 TLSv1

使用环境: main, server

该指令用于指定使用的 SSL 协议。

14.8.11 ssl_verify_client 指令

语法: ssl_verify_client on|off|ask

默认值: `ssl_verify_client off`

使用环境: `main, server`

该指令用于开启客户端证书验证。参数“ask”在客户端主动提出检查证书时,对客户端证书进行检查。

14.8.12 `ssl_verify_depth` 指令

语法: `ssl_verify_depth number`

默认值: `ssl_verify_depth 1`

使用环境: `main, server`

设置客户端证书链的深度。

14.8.13 `ssl_session_cache` 指令

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

默认值: `ssl_session_cache off`

使用环境: `main, server`

该指令设置用来存储 SSL 会话的缓存类型和大小。缓存类型为:

`off`——硬关闭: Nginx 明确告诉客户端这个会话不可用。

`none`——软关闭: Nginx 告诉客户端会话能够被重用,但是 Nginx 实际上不会重用它们。这是为某些邮件客户端使用的一种变通方法,可以被用于邮件代理和 HTTP 服务器中。

`builtin`——OpenSSL 内置缓存,仅可用于一个工作进程。缓存大小用会话数来指定。注意:使用该指令会导致内存碎片,当使用此功能时须要考虑。

`shared`——位于所有工作进程的共享缓存。缓存的大小用字节数指定,1MB 缓存能够容纳大约 4 000 会话。每个共享缓存必须拥有自己的名称。同名的缓存可以用于多个虚拟主机。

你也可以同时使用 `builtin` 和 `shared`, 示例如下:

```
ssl_session_cache builtin:1000 shared:SSL:10m;
```

然而,只使用共享内存而不使用 `builtin` 缓存,将更有效。

14.8.14 ssl_session_timeout 指令

语法: ssl_session_timeout time

默认值: ssl_session_timeout 5m

使用环境: main, server

设置客户端能够重复使用存储在缓存中的会话参数时间。

该模块支持一些非标准的错误代码，可以借助 error_page 指令来做 debug 调试：

495——检查客户端证书时发生错误。

496——客户端不允许必须的证书。

497——正常的请求发送到 HTTPS。

在调试完成后，可以取得一些变量，例如 \$request_uri、\$uri、\$arg 等。Nginx 的 Http SSL 模块支持这些内置变量：

\$ssl_cipher: 返回既定 SSL 连接中使用的密码行。

\$ssl_client_serial: 返回既定 SSL 连接的客户端证书的序列号。

\$ssl_client_s_dn: 返回既定 SSL 连接的客户端证书的 DN 主题行。

\$ssl_client_i_dn: 返回既定 SSL 连接的客户端证书的 DN 发行站行。

\$ssl_protocol: 返回既定 SSL 连接的协议。

\$ssl_session_id: 在 Nginx 0.8.20 以上版本支持该变量。

\$ssl_client_cert

\$ssl_client_raw_cert

\$ssl_verify "SUCCESS": 如果客户端证书验证通过，该变量的值为“SUCCESS”。

14.8.15 ssl_engine 指令

语法: ssl_engine

该指令指定允许去使用的 OpenSSL 引擎，例如 Padlock。须要安装一个近期发布的 OpenSSL 版本。

14.9 HTTP Stub Status 模块

本模块主要用于查看 Nginx 的一些状态信息。

本模块默认是不会编译进 Nginx 的，如果你要使用该模块，则要在编译安装 Nginx 时指定：

```
./configure --with-http_stub_status_module
```

配置示例如代码 14-13：

代码 14-13

```
server
{
    listen 80;
    server_name status.yourdomain.com;

    location / {
        stub_status on;
        access_log off;
        allow 192.168.1.2;
        deny all;
    }
}
```

14.9.1 stub_status 指令

语法：stub_status on

默认值：None

使用环境：location

该指令用于开启 Nginx 状态信息。

访问以上示例中配置的 `http://status.yourdomain.com/`，显示的 Nginx 状态信息示例如下：

```
Active connections: 17580
server accepts handled requests
 38810620 38810620 298655730
Reading: 68 Writing: 1219 Waiting: 16293
```

Active connections——对后端发起的活动连接数。

Server accepts handled requests——Nginx 总共处理了 43 629 083 个连接，成功创建 43 629 083 次握手（证明中间没有失败的），总共处理了 259 552 136 个请求。

Reading——Nginx 读取到客户端的 Header 信息数。

Writing——Nginx 返回给客户端的 Header 信息数。

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

所以，在访问效率高，请求很快被处理完毕的情况下，Waiting 数比较多是正常的。如果 reading + writing 数较多，则说明并发访问量非常大，正在处理过程中。

14.10 HTTP Sub 模块

本模块主要用来搜索并替换 Nginx 应答内容中的文本。

本模块默认是不会编译进 Nginx 的，如果你要使用该模块，则要在编译安装 Nginx 时指定：
`./configure --with-http_sub_module`

示例如下：

```
location / {  
    sub_filter      </head>  
'</head><script language="javascript" src="$script"></script>';  
    sub_filter_once on;  
}
```

14.10.1 sub_filter 指令

语法：sub_filter text substitution

默认值：none

使用环境：http, server, location

该指令允许使用一些其他的文本替换 Nginx 应答内容中的一些文本。匹配不区分大小写，替换的文本可以包含变量。每个 location 只能指定一个替换规则。

14.10.2 sub_filter_once 指令

语法：sub_filter_once on|off

默认值：sub_filter_once on

使用环境：http, server, location

该指令的值设置为 off 时，允许搜索并替换所有匹配行。该指令的默认值为 on，只替换第一个匹配项。



14.10.3 sub_filter_types 指令

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

默认值: sub_filter_types text/html

使用环境: http, server, location

该指令用于指定 sub_filter 指令将检查哪一内容类型，默认值为 text/html。

14.11 HTTP Dav 模块

本模块用来设置允许 HTTP 和 WebDAV 方法: PUT、DELETE、MKCOL、COPY、MOVE。

本模块默认是不会编译进 Nginx 的，如果你要使用该模块，则要在编译安装 Nginx 时指定：

```
./configure --with-http_dav_module
```

示例如代码 14-14:

代码 14-14

```
location / {
    root    /data/www;
    client_body_temp_path /data/client_temp;

    dav_methods PUT DELETE MKCOL COPY MOVE;

    create_full_put_path on;
    dav_access      group:rw all:r;

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

14.11.1 dav_access 指令

语法: dav_access user:permissions [users:permissions] ...

默认值: dav_access user:rw

使用环境：http, server, location

该指令用于指定文件和目录的访问权限，示例如下：

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

如果指定任何允许的 groups 或 all，则无须为 user 指定权限：

```
dav_access group:rw all:r;
```

14.11.2 dav_methods 指令

语法：dav_methods [off|put|delete|mkcoll|copy|move] ...

默认值：dav_methods off

使用环境：http, server, location

该指令用来设置允许指定的 HTTP 和 WebDAV 方法。如果设置为 off，所有的方法将被禁止，并且忽略剩余的参数。

PUT 方法的目的文件必须在相同分区的临时文件存储目录中存在，使用 client_body_temp_path 指令在 section 区域中设置。

当一个文件被使用 PUT 方法创建时，将使用 Date 头信息作为该文件的修改时间。

14.11.3 create_full_put_path 指令

语法：create_full_put_path on|off

默认值：create_full_put_path off

使用环境：http, server, location

默认情况（off）下，PUT 方法只能在已经存在的目录中创建文件。该指令允许 Nginx 创建目录。

14.12 Google Perftools 模块

本模块允许使用 Google 公司开发的性能优化工具 Google Performance Tools (<http://code.google.com/p/google-perftools/>)。该模块可在 Nginx 0.6.29 之后的版本中使用。

Google Performance Tools 的安装步骤如代码 14-15 所示：

代码 14-15

```
#64 位操作系统须要先安装 libunwind
wget http://ftp.tware.net/Unix/NonGNU/libunwind/libunwind-0.99.tar.gz
tar zxvf libunwind-0.99.tar.gz
cd libunwind-0.99/
CFLAGS=-fPIC ./configure
make CFLAGS=-fPIC
make CFLAGS=-fPIC install
cd ../

wget http://google-perftools.googlecode.com/files/google-perftools-1.4.tar.gz
tar zxvf google-perftools-1.4.tar.gz
cd google-perftools-1.4/
./configure --prefix=/usr
make && make install
cd ../
/sbin/ldconfig
```

本模块默认是不会编译进 Nginx 的，如果你要使用该模块，则要在编译安装 Nginx 时指定：

```
./configure --with-google_perftools_module
```

示例如下：

```
google_perftools_profiles /path/to/profile;
```

Profile 将以/path/to/profile.<worker_pid>格式存储。

14.12.1 google_perftools_profiles 指令

语法：google_perftools_profiles path

默认值：none

该指令用于指定 profiles 基础文件名，工作进程的 PID 将被添加到指定的文件中。

14.13 HTTP XSLT 模块

XSLT 是一种用于将 XML 文档转换为 XHTML 文档或其他 XML 文档的语言。本模块是通过一个或多个 XSLT 模板转换 XML 应答内容的过滤器。本模块适用于 Nginx 0.7.8 以后版本。

本模块默认是不会编译进 Nginx 的，如果你要使用该模块，则要在编译安装 Nginx 时指定：

```
./configure --with-http_xslt_module
```

配置示例如下：

```
location / {  
    xml_entities      /site/dtd/entities.dtd;  
    xslt_stylesheet   /site/xslt/one.xslt  param=value;  
    xslt_stylesheet   /site/xslt/two.xslt;  
}
```



14.13.1 xslt_entities 指令

语法: `xslt_entities <path>`

默认值: `no`

使用环境: `http, server, location`

用于指定 DTD 描述文件 (XML 实体)。DTD 实际上可以看作一个或多个 XML 文件的模板, 这些 XML 文件中的元素、元素的属性、元素的排列方式/顺序、元素能够包含的内容等, 都必须符合 DTD 中的定义。此文件在配置阶段被编译。由于技术原因, 不能对正在处理的 XML 文件指定实体, 但是会用一个特别指定的文件来代替。在这个文件中, 没有必要来描述处理 XML 的结构, 只须说明必须的符号元素即可, 例如:

```
<! ENTITY of nbsp " ">
```

14.13.2 xslt_stylesheet 指令

语法: `xslt_stylesheet template [parameter[[parameter...]]`

默认值: `no`

使用环境: `http, server, location`

用于指定 XSLT 模板的参数。模板在配置阶段被编译。参数按照以下方法指定:

```
param=value
```

你可以为每行指定任何参数, 或者使用 “:” 分割多个参数。如果参数自身包含字符 “:”, 请使用 “%3A” 转义。此外, 如果参数包含非字母数字的字符, `libxslt` 要求字符串参数应该用单引号或双引号引用。示例如下:

```
param1='http%3A/www.example.com': param2=value2
```

它可以使用变量作为参数, 例如, 参数值可以用一个变量来取代:

```
location / {  
    xslt_stylesheet /site/xslt/one.xslt  
    $arg_xslt_params
```

```
param1='$value1': param2=value2
param3=value3;
}
```

可以指定多个模板，在这种情况下，将按照它们的声明顺序链接在一起。

14.13.3 xslt_types 指令

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

默认值: `xslt_types text/xml`

使用环境: `http, server, location`

允许处理除了“text/xml”之外的指定 MIME 类型。如果 XSLT 输出模式为 HTML，应答的 MIME 类型可以更改为“text/HTML”。

14.14 HTTP Secure Link 模块

本模块计算和请求需要安全性令牌的 URL 地址。在 Nginx 0.7.18 以上版本中可以使用本模块。

本模块默认是不会编译进 Nginx 的，如果你要使用该模块，则要在编译安装 Nginx 时指定：

```
./configure --with-http_secure_link_module
```

配置示例代码如下：

```
location /prefix/ {
    secure_link_secret secret_word;

    if ($secure_link = "") {
        return 403;
    }
}
```

14.14.1 secure_link_secret 指令

语法: `secure_link_secret secret_word`

默认值: `none`

使用环境: `location`

该指令用于指定一个密码去校验请求。一个被保护链接的完整 URL 如下：

```
/prefix/hash/reference
```

该指令将按照以下方式进行哈希计算：

```
md5 (reference, secret_word);
```

`prefix` 为 `location` 区块的位置范围，不能为 `/`。`secure_link` 只能用于非 `root` 路径。

配置示例代码如 14-16：

代码 14-16

```
server
{
    listen      80;
    server_name www.yourdomain.com;
    index index.html index.htm;
    root /data0/htdocs/www;

    location /download {
        secure_link_secret password;
        if ($secure_link = "") {
            return 403;
        }
        rewrite (.*?) /download/$secure_link break;
    }
}
```

假设有两个文件 `/data0/htdocs/www/download/demo.mp4` 和 `/data0/htdocs/www/download/video/demo.mp4`，可通过 PHP 程序生成一个 md5 串：

```
<?php
echo md5("demo.mp4"."password")."\n";
echo md5("video/demo.mp4"."password")."\n";
?>
```

执行该 PHP 程序得出：

```
626e08fd9d6822090b3d684a8ef325d7
ceab52da294cb1a77671d315000bf6ef
```

通过以下 URL 访问 Nginx，就可以下载这两个文件：

```
http://www.yourdomain.com/download/626e08fd9d6822090b3d684a8ef325d7/demo.mp4
http://www.yourdomain.com/download/626e08fd9d6822090b3d684a8ef325d7/video/demo.mp4
```

14.14.2 \$secure_link 变量

自动设置 URL 的基准部分，并与 `prefix` 和 `hash` 分开。如果 `hash` 错误，变量 `$secure_link` 的值将为空。

14.15 HTTP Image Filter 模块

本模块是一个转换 JPEG、GIF、PNG 图片的过滤器。

本模块默认是不会编译进 Nginx 的，如果你要使用该模块，则要在编译安装 Nginx 时指定：

```
./configure --with-http_image_filter_module
```

本模块需要操作系统安装 libgd 类库。

配置示例如代码 14-17：

代码 14-17

```
location /img/ {
    proxy_pass    http://backend;
    image_filter  resize 150 100;
    error_page    415    = /empty;
}

location = /empty {
    empty_gif;
}
```

14.15.1 image_filter 指令

语法：image_filter (testsize|resize width height|crop width height)

默认值：none

使用环境：location

它指定适用于图片的转换类型。

test: 检查应答是否确实为一个图片格式 JPEG、GIF、PNG，如果不是，返回 415。

size: 以 JSON 格式给出图片的信息，示例如下：

```
{ "img" : { "width": 100, "height": 100, "type": "gif" } }
```

或者当一个错误发生时，返回：

```
{ }
```

resize: 按比例缩小图片到指定的大小。

crop: 按比例缩小图片到指定的大小，并裁减掉超过指定大小的图片区域。

14.15.2 image_filter_buffer 指令

语法: `image_filter_buffer size`

默认值: 1M

使用环境: http, server, location

设置读取图片的最大值。

14.15.3 image_filter_jpeg_quality 指令

语法: `image_filter_jpeg_quality [0...100]`

默认值: 75

使用环境: http, server, location

设置处理 JPEG 图片的质量损耗比率。最大推荐者为 95。

14.15.4 image_filter_transparency 指令

语法: `image_filter_transparency on|off`

默认值: on

使用环境: http, server, location

该指令允许你在 GIF 和基于调色板的 PNG 禁止使用图片透明，从而提高图片质量。

不管这个指令如何设置，真彩色 PNG alpha-channels 总是存在。

注意：灰度 PNG 未经测验，但是能够被当成真彩色来处理。