lifayi2008 / hexo_blog

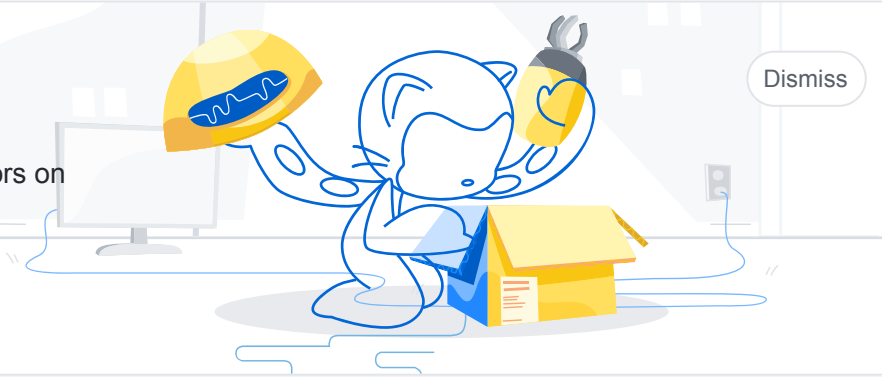
Sign up

Code Issues 0 Pull requests 0 Projects 0 Security Pulse

All your code in one place

GitHub makes it easy to scale back on context switching. Read rendered documentation, see the history of any file, and collaborate with contributors on projects across GitHub.

Sign up for free See pricing for teams and enterprises



Dismiss

Tree: 458626a814 hexo_blog / source / _posts / Nginx管理员指南-译.md

Find file Copy path

lifayi2008 new post 47383e2 on Nov 11, 2016

1 contributor

1546 lines (1128 sloc) 69 KB

Raw Blame History

title	date	tags	categories
Nginx管理员指南-译	2016-08-22 08:21:06 -0700	nginx web	基础服务

Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD

title	date	tags	categories

Nginx进程及进程管理

Master和worker进程

正常启动的nginx有一个master进程和多个worker进程。如果启动caching，则也会启动一个cache leader进程和一个cache manager进程

Master进程主要负责读入并应用配置文件，同时也管理worker进程

Worker进程真正的来处理请求。Nginx使用一个不依赖于操作系统的机制高效的将请求分发给worker进程。Worker进程的数量可以在配置文件中设置为一个固定的值或由nginx根据当前系统可用核心数来决定

Nginx控制

可以给运行着的nginx进程发送不同的信号来，执行不同的操作，形式为：

```
nginx -s signal
```

signal可以是下面的值：

- quit – Shut down gracefully
- reload – Reload the configuration file
- reopen – Reopen log files
- stop – Shut down immediately (fast shutdown)

也可以使用`kill`直接给主进程的ID发送信号（可发送的信号可以参考nginx.org），主进程id保存在`nginx.pid`文件中，这个文件一般在`/usr/local/nginx/logs/`或者`/var/run/`目录中

Nginx作为一个web服务器

Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD

Web服务器中可以配置多个virtual server，每个虚拟主机可以定义多个不同的配置实例---location，每一个location都有很多选项来控制映射到本location的一系列URI应该如何处理。每一个location都可以将请求转发或者返回一个文件（页面），而且还可以更改请求的URI，然后这些请求被重定向到另外一些location或者virtual server。另外可以返回特定的错误代码，你也可以设置这些错误代码应该返回的页面

设置虚拟主机

可以在nginx配置文件中配置一个或者多个virutal server：

```
http {
    server {
        # Server configuration
    }
}
```

`server`配置块中通常包含一个listen配置项表示这个virtual server监听的地址和端口（也可以是一个unix domain socket的地址），ipv4地址和ipv6地址都是可以的，ipv6地址需要放置在一个方括号中

```
server {
    listen 127.0.0.1:8080;
    # The rest of server configuration
}
```

如果没有冒号和地址，则表示监听在所有地址；如果没有冒号和端口，则表示监听在80端口；如果没有`listen`，则在特权用户下是80端口，非特权用户下是8000端口

如果根据地址和端口匹配到的`server`有多个，则nginx会根据请求头中的`Host`信息，来测试匹配的`server`。`server_name`配置项可以是一个完整的主机名，通配符或者是正则表达式，如果是正则表达式则需要以`~`开始

Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD

```
server {
    listen      80;
    server_name example.org www.example.org;
    ...
}
```

如果有多个`server_name`被匹配到，则nginx按照下面的顺序来决定哪一个`server`会处理这个请求：

1. Exact name
2. Longest wildcard starting with an asterisk, such as *.example.org
3. Longest wildcard ending with an asterisk, such as mail.*
4. First matching regular expression (in order of appearance in the configuration file)

如果主机名没有被任何`server_name`匹配到，则nginx会使用第一个server来处理这个请求，除非你显式的使用`default_server`配置项来指定一个默认的主机

```
server {
    listen      80 default_server;
    ...
}
```

配置Location

一个virtual server里可以有多个`location`，nginx会根据`location`配置项的值来决定哪一个`location`块中的配置项会应用到这个请求的处理中。每一个`location`中还可以嵌套另外一个或者多个`location`来分别处理这些请求

`location`配置项的值可以有两种形式：表示URI前缀路径名的字符串，或者正则表达式

Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD

下面的例子中的配置项会匹配`/some/path` `/some/path/document.html`但是不会匹配`/my/some/path`，因为配置项的值会从URI的开始来进行匹配

```
location /some/path/ {
    ...
}
```

正则表达式使用`~`来表示大小写敏感的匹配，`*~`来表示大小写不敏感的匹配，下面的例子中会匹配任何以`.html`结尾的请求

```
location ~ \.html? {
    ...
}
```

对于每一个`location`配置项，nginx会首先进行URI前缀路径匹配，如果没有找到匹配项才会进行正则表达式匹配。准确的逻辑规则如下：

1. Test the URI against all prefix strings
2. The = (equals sign) modifier defines an exact match of the URI and a prefix string. If the exact match is found, the search stops
3. If the ^~ (caret-tilde) modifier prepends the longest matching prefix string, the regular expressions are not checked.
4. Store the longest matching prefix string.
5. Test the URI against regular expressions.
6. Break on the first matching regular expression and use the corresponding location.
7. If no regular expression matches, use the location corresponding to the stored prefix string

Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD

`=`的一个典型应用是对`/`的请求，下面的配置放在`server`中的第一个会在匹配到`/`后立即停止检查，这样可以最快的处理

```
location = / {
    ...
}
```

`location`配置项的内容决定了如何处理这个请求，直接返回一个页面或者将请求转发到其他的服务器都是可以的。下面的例子中，URI以`/images/`开始的请求会直接从`/data`目录中获取，其他的请求会转发给另外一个URL来处理

```
server {
    location /images/ {
        root /data;
    }

    location / {
        proxy_pass http://www.example.com;
    }
}
```

`root`配置项告诉nginx获取静态文件的根目录，nginx返回`根目录+URI`对应的资源，如果URI是`/images/test.png`，nginx会返回`/data/images/test.png`。`porxy_pass`配置项告诉nginx将请求转发给另外一个URL来处理，将这个URL指定的主机的返回结果返回给客户端

使用变量

可以在nginx的配置文件中使用变量，变量的值到运行时才会进行计算，变量以`$`开始。有些变量值的定义依赖于客户端请求的信息；也有一些预定义的变量可以参考nginx.org文档中的描述；当然你也可以使用`set` `map` `geo`配置项自定义变量。大部分的变量都在运行时才知道实际的值，包含的信息会依赖于实际请求信息。比如`$remote_addr`表示客户端的IP地址，而`$uri`则表示请求中的URI的值

Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD

返回指定的状态码

有些情况下，客户端请求的网页不存在或者已经被暂时或者永久的移动到另外一个位置。这样的话就需要直接返回给客户端一个状态码来通知客户端。可以在配置文件中使用`return`配置项来实现：

```
location /wrong/url {
    return 404;
}
```

配置项的第一个参数是返回的状态码，可选的第一个参数可以指定错误的页面或者应该重定向的页面：

```
location /permanently/moved/url {
    return 301 http://www.example.com/moved/here;
}
```

> `return`配置项可以在`server`配置块或者`location`配置块中使用

重写请求中的**URI**

一个请求中的URI可以在请求处理过程中使用`rewrite`配置项多次进行重写，这个配置项需要两个参数，和一个可选的选项；第一个参数是一个正则表达式用来匹配请求中的URI，第二个参数指定了匹配到的URI应该被替换的内容，第三个选项则用来控制是否就此停止`rewrite`的处理（并返回内容），或者发送一个`301` `302`状态码表示重定向

```
location /users/ {
    rewrite ^/users/(.*)$ /show?user=$1 break;
}
```

本例中正则表达式`()`捕获的信息会代替第二个参数中的`$1`

Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD

可以在`server`和`location`配置块中使用多个`rewrite`。`server`配置块中的一系列`rewrite`指令只会被执行一遍，如果有匹配到的URI则按这条规则进行URI修改，使用修改过的URI进行下一条`rewrite`规则的测试，直到最后一条。`server`中的`rewirte`执行完毕后，nginx会根据新的URI来决定哪一个`location`会处理这个请求，如果`location`中也有`rewrite`规则，则继续对URI进行匹配和修改，修改后的URI要再次进行`location`的匹配，这样一直循环下去，直到匹配到的`location`中的`rewite`规则都不匹配这个新的URI，然后则由这个`location`来处理这个最终的URI请求

```
server {
    ...
    rewrite ^(/download/.*)/media/(.*)\..*$ $1/mp3/$2.mp3 last;
    rewrite ^(/download/.*)/audio/(.*)\..*$ $1/mp3/$2.ra  last;
    return  403;
    ...
}
```

上面的例子中混合了`rewrite`和`return`指令，`/download/some/media/file`的请求会被重写为`/download/some/mp3/file.mp3`，因为使用了`last`标志选项，第二条rewrite规则会被直接跳过；如果URI没有被两条rewrite规则匹配到则返回`403`状态码

有两种类型的中断`rewrite`处理的指令选项:

- **`last`** – Stops execution of the rewrite directives in the current server or location context, but NGINX Plus searches for locations that match the rewritten URI, and any rewrite directives in the new location are applied (meaning the URI can be changed again)
- **`break`** – Like the break directive, stops processing of rewrite directives in the current context and cancels the search for locations that match the new URI. The rewrite directives in the new location are not executed.

`last`会停止本server或者location中的剩余的rewrite的处理，而`break`则停止所有rewrite处理

Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD

重写HTTP相应

使用`sub_filter`配置项可以重写http响应中的某些内容。这个配置项也支持变量和链式替换，可以组合出复杂的替换规则

```
location / {
    sub_filter      /blog/ /blog-staging/;
    sub_filter_once off;
}
```

下面的配置中将响应内容中链接的`http://`替换为`http://`并且将本地地址替换为请求头中的服务器地址。而`sub_filter_once`配置项则表示连续的应用本location中的所有`sub_filter`：

```
location / {
    sub_filter     'href="http://127.0.0.1:8080/'    'href="http://$host/';
    sub_filter     'img src="http://127.0.0.1:8080/' 'img src="http://$host/';
    sub_filter_once on;
}
```

> 注意：已经被替换的内容不会再进行第二次匹配替换

处理错误

使用`error_page`配置项可以让nginx返回一个自定义的错误页面，或者替换为另外一个错误码，或者重定向到另外一个URI地址。下面的例子中如果有404错误时返回`404.html`页面：

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

Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD

上面的配置的意思并不是返回一个404状态码，而是有404状态码出现时会返回对应的页面。响应码可以来自于被代理的服务器或者是nginx本身

下面的例子表示如果有404出现时，替换为301状态码并且重定向到后面的URL地址。这在用户访问一个旧的URL时有用，301状态码表示这个地址已经被永久重定向到后面的URL地址：

```
location /old/path.html {
    error_page 404 =301 http:/example.com/new/path.html;
}
```

下面的例子表示当一个请求未找到时，重定向到另外一个URI（被代理的后端服务器）。因为这里没有指定状态码，所以返回给客户端的状态码是被代理的服务器返回的值：

```
server {
    ...
    location /images/ {
        # Set the root directory to search for the file
        root /data/www;

        # Disable logging of errors related to file existence
        open_file_cache_errors off;

        # Make an internal redirect if the file is not found
        error_page 404 = /fetch$uri;
    }

    location /fetch/ {
        proxy_pass http://backend/;
    }
}
```

Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD

`open_file_cache_errors`配置项表示当请求未找到时阻止写入一条错误消息，因为这里的错误已经被妥善处理，所以关闭这个选项

配置**nginx**提供静态内容访问

根目录和索引文件

配置文件中的`root`配置项指定了搜索文件的根目录。Nginx会将请求中的URI附加到制定的根目录路径的后面来返回对应的资源。这个配置项可以出现在http，server和location配置块中。下面例子中的server配置块中的root配置项会应用到本server中的所有没有明确指定根目录的location中：

```
server {
    root /www/data;

    location / {
    }

    location /images/ {
    }

    location ~ \.(mp3|mp4) {
        root /www/media;
    }
}
```

上面的配置中，如果URI以`/images`开始则从`/www/data/images/`目录中查找请求的资源，而以`mp3/mp4`结尾的请求则从`/www/media/`目录中查找资源

如果请求以斜线结尾，则nginx会认为要请求这个目录中的索引文件。index配置项则指定了索引的文件名（默认是index.html）。在上面的例子中如果用户请求的URI是`/images/some/path/`则nginx会将返

Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD

回`/www/data/images/some/path/index.html`文件返回。如果这个文件不存在则返回404状态码。可以使用`autoindex`配置项来将用户请求的目录中的所有文件列出，而不是返回目录中的index.html文件:

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

index配置项可以指定多个值，nginx会返回第一个找到的文件:

```
location / {
    index index.$geo.html index.htm index.html;
}
```

上面配置中的`$geo`变量是一个使用geo配置项定义的变量，这个变量的值依赖于客户端的IP地址

Nginx根据`index`配置项指定的文件进行搜索，将第一个存在的文件附加到URI路径后面，然后还会依次进行location的匹配，而最后处理这个请求的location可能是另外一个匹配到完整URI的location：

```
location / {
    root /data;
    index index.html index.php;
}

location ~ \.php {
    fastcgi_pass localhost:8000;
    ...
}
```

Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD

如果用户请求的URI是`/path/`，而`/data/path/index.html`文件不存在，但是`/data/path/index.php`文件存在，则新的请求URI会是`/path/index.php`，所以第二个location会处理这个请求

Trying Serval Options

`try_files`配置项可以用来测试特定的文件或者目录是否存在，如果不存在则进行内部重定向或者返回一个错误码。比如要检查URI对应的文件是否存在可以使用`try_files`配置项和`$uri`变量：

```
server {
    root /www/data;

    location /images/ {
        try_files $uri /images/default.gif;
    }
}
```

文件以URI的形式指定，最终要测试的文件是应用了root配置项或者alias配置项的结果。本例中nginx会检查URI对应的文件是否存在，如果不存在则返回`/www/data/images/default.gif`文件

最后一个参数可以是一个状态码（需要前置一个等号）或者是一个命名的`location`：

```
location / {
    try_files $uri $uri/ $uri.html =404;
}
location / {
    try_files $uri $uri/ @backend;
}

location @backend {
    proxy_pass http://backend.example.com;
}
```

Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD

优化静态内容访问速度

加载的速度对nginx静态内容的响应有重要的影响。下面的一些简单的优化方法可能会带来性能的提升

`启用sendfile`
默认情况下，nginx发送文件之前会将文件拷贝至buffer中。启动sendfile配置项，则让nginx直接将内容从一个文件描述符拷贝至另外一个文件描述符，不经过中间的缓存。另外为了避免一个快速的连接整个的占据一个worker进程，我们可以使用`sendfie_max_chunk`配置项限制一个`sendfile()`系统调用可以传输的数据总量：

```
location /mp3 {
    sendfile           on;
    sendfile_max_chunk 1m;
    ...
}
```

`启用tcp_nopush`
在开启`sendfile`的情况下启用`tcp_nopush`配置项可以让nginx在sendfile系统调用获取到数据后立即发送http响应头:

```
location /mp3 {
    sendfile   on;
    tcp_nopush on;
    ...
}
```

`启用tcp_nodelay`
启用`tcp_nodelay`配置项可以替换Nagle's算法，这种算法被设计用来解决慢速网络中的小数据包问题；这种算法会将多个小数据包合成一个大的数据包发送，所以在发送前等待200ms。默认情况下`tcp_nodelay`已经被开启。所以这个选项仅仅用来保持连接：

Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD

```
location /mp3  {
    tcp_nodelay       on;
    keepalive_timeout 65;
    ...
}
```

优化**backlog**队列

默认情况下linux待处理连接队列最大值是128:

```
sudo sysctl -a | grep somax
可以使用下面的命令更改这个值
sudo sysctl -w net.core.somaxconn=4096
```

或者在/etc/sysctl.conf文件中添加:

```
net.core.somaxconn = 4096
```

然后使用`sysctl -p`命令使其生效

nginx中可以在`listen`配置项后面添加`backlog`值:

```
server {
    listen 80 backlog 4096;
    # The rest of server configuration
}
```

Nginx作为反向代理

Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD

Nginx作为反向代理时，可以使用多种协议将请求转发给后端服务器，也可以更改请求头中的信息，或者缓存被代理服务器的响应信息。反向代理的典型应用是，将负载分配到后端的若干服务器，或者是将http请求转换为另外一种请求协议发给后端服务器

将请求转发给后端服务器

将请求转发给后端服务器可以在`location`配置块中使用`proxy_pas`配置项：

```
location /some/path/ {
    proxy_pass http://www.example.com/link/;
}
```

配置值也可以是IP地址加端口的形式：

```
location ~ \.php {
    proxy_pass http://127.0.0.1:8000;
}
```

> 注意：在第一个例子中配置值指定的地址后面有一个URI，这样的话匹配到的`/some/path/`会被这个URI代替，比如`/some/path/page.html`的请求转发到后端的服务器时会变成`http://www.example.com/link/page.html`；如果配置值中没有URI则原始的请求URI被转发到后端

Nginx支持下面几种形式的非http转发：

- **fastcgi_pass** passes a request to a FastCGI server
- **uwsgi_pass** passes a request to a uwsgi server
- **scgi_pass** passes a request to an SCGI server
- **memcached_pass** passes a request to a memcached server

Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD

> 注意：不同的请求方法可能还需要指定另外的一些参数

配置项`proxy_pass`的值可以是一个命名的主机组，这样的话请求会被分发到组中的主机，详细的信息可以参考后面的负载均衡章节

控制请求头

默认情况下，经过代理的请求头信息会有两处被修改或者添加：`Host`被改为`$proxy_host`变量，`Connection`改为`close`

要更改请求中的头信息可以使用`proxy_set_header`配置项。这个配置项一般用在location中，但也可以用在server和http配置块:

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

本例中Host被设置为`$host`变量；如果要阻止某个请求头信息转递给后端主机可以设置为空：

```
location /some/path/ {
    proxy_set_header Accept-Encoding "";
    proxy_pass http://localhost:8000;
}
```

配置**buffer**

默认情况下nginx会缓存后端服务器发送过来的响应信息，直到它收到这个完整的响应信息；这样在客户端较慢的情况下不用浪费服务器的时间

Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD

配置项`proxy_buffering`指定了这个设置，默认为on开启的状态

`proxy_buffers`配置项指定了分配给一个请求的buffer的大小和数量。后端服务器响应信息的第一部分（http响应头信息）存放在一个单独的buffer中，这个buffer的大小由`proxy_buffer_size`配置项指定，这个值通常小于其他的buffer的值（但有时候也可能需要大的头信息）：

```
location /some/path/ {
    proxy_buffers 16 4k;
    proxy_buffer_size 2k;
    proxy_pass http://localhost:8000;
}
```

如果大部分客户端需要尽快的到相应的信息，我们可以使用`proxy_buffering off`来关闭这个功能。这样就只缓存http响应头，所以只有`proxy_buffer_size`配置项有效

将连接代理的流量绑定到特定的IP

配置项`proxy_bind`可以将发送给后端服务器的数据绑定在某一个接口：

```
location /app1/ {
    proxy_bind 127.0.0.1;
    proxy_pass http://example.com/app1/;
}
location /app3/ {
    proxy_bind $server_addr;
    proxy_pass http://example.com/app3/;
}
```

响应内容压缩

Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD

压缩相应通常会显著的减少传输的数据量。但同时他也会增加运行时负载，这可能会给服务器性能带来负面影响。Nginx会在将相应内容发送给客户端之前进行压缩，但是不会压缩已经压缩过的响应内容（比如作为一个代理服务器时）

启用压缩

要启用压缩功能，只需要将gzip配置项的值指定为on即可：

```
gzip on;
```

默认情况下，Nginx仅仅压缩MIME类型为`text/html`的相应。要压缩其他类型，需要使用`gzip_types`配置项来指定其他的类型：

```
gzip_types text/plain application/xml;
```

要指定启用压缩的最小响应体大小，可以使用`gzip_min_length`配置项。默认是20字节（这里增加到1000字节）：

```
gzip_min_length 1000;
```

默认情况下，Nginx对代理服务发送的请求的响应不会进行压缩。而判断一个请求是否是代理服务器发送过来的依据是请求头中是否包含via头信息。要让Nginx也压缩此类请求的相应，则需要使用`gzip_proxied`配置项，这个配置项有几个配置参数来指定哪种被代理的请求会进行压缩。比如，我们通常只会压缩那些不会被缓存服务器缓存的响应内容。`gzip_proxied`配置项有些参数可以让Nginx检查相应内容的`Cache-Control`头字段，如果字段值是`no-cache`，`no-store`或者`private`时才启用压缩。另外也需要包含`expired`参数来让Nginx检查`Expired`头字段的值。这些参数按照下面例子中的配置方法，后面跟着auth参数可以检查响应头中是否包含`Authorization`头字段（包含这个字段的响应信息不会被缓存）

Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD

```
gzip_proxied no-cache no-store private expired auth;
```

和其他大多数配置项一样，这些启用压缩的配置项可以在http，server和location配置块中进行配置

整个启用内容压缩的配置项如下：

```
server {
    gzip on;
    gzip_types      text/plain application/xml;
    gzip_proxied    no-cache no-store private expired auth;
    gzip_min_length 1000;
    ...
}
```

启用解压缩

一些客户端不能读取使用gzip编码方法压缩的相应内容。有时候我们想存储被压缩的数据（解压缩存储）或者运行时解压缩响应数据然后将它存入缓存。为了同时满足支持和不支持压缩内容的客户端，Nginx运行时解压缩数据然后将其发送给客户端

要启用解压缩，可以使用`gunzip`配置项：

```
location /storage/ {
    gunzip on;
    ...
}
```

`gunzip`配置项可以和`gzip`配置项在同一个配置块中使用

Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD

```
server {
    gzip on;
    gzip_min_length 1000;
    gunzip on;
    ...
}
```

> 注意：本配置项在一个分开的模块中定义，默认的编译参数不能启用这个配置项

发送压缩文件

要发送一个压缩版本的文件到客户端，需要在合适的配置块中设置`gzip_static`配置项的值为on：

```
location / {
    gzip_static on;
}
```

这种情况下，如果客户端发来一个/path/to/file文件的请求，则Nginx会尝试查找并发送/path/to/file.gz文件。如果文件不存在，或者客户端不支持gzip，则发送未压缩的文件

`gzip_static`不会在运行时对文件进行压缩。它仅仅指示nginx使用已经压缩的文件响应请求

> 注意：本配置项在一个分开的模块中定义，默认的编译参数不能启用这个配置项

内容缓存

当启用内容缓存时，如果同样的请求再次到来nginx直接将缓存的内容发给客户端，而不用向后端被代理的服务器去请求资源

启用响应缓存

Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD

要启用缓存，可以在http配置块中包含`proxy_cache_path`配置项。第一个强制的参数指定了缓存内容存放的路径，而强制的参数`keys_zone`则定义了缓存zone名称和这个zone对应缓存条目元数据可使用的共享内存的大小；然后就可以在想要缓存的server或者location中使用`proxy_cache`配置项指定这个zone名称：

```
http {
    ...
    proxy_cache_path /data/nginx/cache keys_zone=one:10m;

    server {
        proxy_cache one;
        location / {
            proxy_pass http://localhost:8000;
        }
    }
}
```

注意上面定义的内存限制并不是限制总的缓存的数据；被缓存的内容和元数据会另外存储在磁盘特定的文件里，这里只是限制了可存储在内存中的数据的总量。要限制总的缓存的大小需要在`proxy_cache_path`配置项中指定`max_size`参数

缓存需要调用的进程

有两个额外的进程需要在缓存中使用到

缓存管理器：它会周期性的被激活来检查缓存的状态。如果缓存的大小超过了上面配置的限制值，则会删除那些最近最不经常访问的数据；所以在管理器两次被激活期间，缓存数据可能会超出限制值

缓存导入器：只会在nginx刚刚启动时运行一次。会导入之前缓存数据的元数据到共享内存区域。导入整个缓存的元数据可能会消耗大量资源，可能在nginx刚刚运行的短暂时间内降低nginx的性能。要避免这个问题可以使用下面的配置项来分段导入：

Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD

- **loader_threshold** – Duration of an iteration, in milliseconds (by default, 200)
- **loader_files** – Maximum number of items loaded during one iteration (by default, 100)
- **loader_sleeps** – Delay between iterations, in milliseconds (by default, 50)
-

下面的例子中每次迭代时间为300毫秒，每次导入200个条目：

```
proxy_cache_path /data/nginx/cache keys_zone=one:10m loader_threshold=300 loader_files=200;
```

指定缓存哪些请求

默认情况下nginx会缓存所有的HTTP GET和HEAD请求的相应数据，并且会在第一次请求时就缓存响应数据。Nginx默认使用请求字符串作为缓存内容的key。如果一个请求的key可以在缓存中找到则直接使用缓存的内容响应客户端。你可以在http、server和location配置块中使用多种配置项来控制缓存

要改变缓存key的计算方法，可以使用`proxy_cache_key`配置项：

```
proxy_cache_key "$host$request_uri$cookie_user";
```

可以使用`proxy_cache_min_uses`配置项定义同样的key被请求最少多少次时才会缓存：

```
proxy_cache_min_uses 5;
```

要缓存除GET HEAD请求方法之外的其他请求方法可以使用`proxy_cache_method`配置项：

```
proxy_cache_methods GET HEAD POST;
```

Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD

限制或者绕过缓存

默认情况下，缓存中的内容会被无限期保存直到超出配置的最大限制，然后会删除最不经常被访问的缓存内容。你可以在http server和location配置块中来设置缓存过期条件

针对响应码来设置过期时间可以使用`proxy_cache_valid`配置项：

```
proxy_cache_valid 200 302 10m;
proxy_cache_valid 404      1m;
```

上例中响应码是200和302过期时间为10分钟，而响应码是404的则一分钟过期，要指定所有的响应码可以使用

要设置在什么条件下会将缓存内容发送给客户端可以使用`proxy_cache_bypass`配置项，配置项的每一个参数指定了一个条件，参数中可以包含多个变量。如果有一个参数非空或者不等于`0`，则nginx不会返回缓存中的内容，而是将请求转发给后端服务器：

```
proxy_cache_bypass $cookie_nocache $arg_nocache$arg_comment;
```

使用`proxy_no_cache`配置项可以定义在什么条件下nginx不缓存，对条件的检查规则和上面一样：

```
proxy_no_cache $http_pragma $http_authorization;
```

混合配置示例

```
http {
    ...
    proxy_cache_path /data/nginx/cache keys_zone=one:10m loader_threshold=300
                     loader_files=200 max_size=200m;
```

Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD

```
    server {
        listen 8080;
        proxy_cache one;

        location / {
            proxy_pass http://backend1;
        }

        location /some/path {
            proxy_pass http://backend2;
            proxy_cache_valid any 1m;
            proxy_cache_min_uses 3;
            proxy_cache_bypass $cookie_nocache $arg_nocache$arg_comment;
        }
    }
}
```

管理**SSL**连接

设置一个**HTTPS**服务器

要设置https服务，只需要在配置文件的server配置块中listen配置项后面加上ssl选项值，然后指定服务器证书和私钥文件的路径即可：

```
server {
    listen              443 ssl;
    server_name         www.example.com;
    ssl_certificate     www.example.com.crt;
    ssl_certificate_key www.example.com.key;
    ssl_protocols       TLSv1 TLSv1.1 TLSv1.2;
    ssl_ciphers         HIGH:!aNULL:!MD5;
    ...
}
```

Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD

服务器证书就是公钥，会发送给所有连接到本服务器的客户端。私钥应该放置在一个访问受限的安全的位置，只允许服务器master进程读取私钥文件。服务器证书和私钥也可以存储与同一个文件中：

```
ssl_certificate www.example.com.cert;
ssl_certificate_key www.example.com.cert;
```

这种情况下这个文件的访问要受到限制。尽管服务器证书和私钥放置在同一个文件中，但是客户端连接时Nginx只会把公钥发送给客户端

`ssl_protocols` `ssl_ciphers`配置项可以用来限制只能使用特定版本的安全协议和安全套件

1.0.5版本的Nginx默认使用`ssl_protocols SSLv3 TLSv1 和 ssl_ciphers HIGH:!aNULL:!MD5`；从版本1.1.13和1.0.12，默认使用`ssl_protocols SSLv3 TLSv1 TLSv1.1 TLSv1.2`

旧的安全套件设计中常常有一些缺陷，明智的做法是在现在的Nginx配置中禁用他们（但是可能会有一些需要向后兼容的需求让你不太那么容易使用最新的安全套件）。请注意CBC-mode安全套件可能遭受到一些安全攻击，目前使用SSLv3是避免这些攻击的最好办法

HTTPS服务器优化

SSL加密服务会额外消耗一些CPU资源。最耗资源的是SSL协议的握手阶段。下面两种方法可以用来减少这些操作：

- 启用长连接来在同一个连接中发送多个资源
- 重用SSL会话的参数可以在并发和后续连接建立时减少SSL握手的操作

可以使用`ssl_session_cache`配置项启用ssl会话缓存功能，Sessions被存储在SSL会话缓存中并且被所有worker进程共享。1M的缓存大概可以存储4000个Sessions。默认的缓存超时时间是5分钟，这个时间可以使用`ssl_session_timeout`配置项来更改。下面是一个优化配置的示例：

Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD

```
worker_processes auto;

http {
    ssl_session_cache   shared:SSL:10m;
    ssl_session_timeout 10m;

    server {
        listen              443 ssl;
        server_name         www.example.com;
        keepalive_timeout   70;

        ssl_certificate     www.example.com.crt;
        ssl_certificate_key www.example.com.key;
        ssl_protocols       TLSv1 TLSv1.1 TLSv1.2;
        ssl_ciphers         HIGH:!aNULL:!MD5;
        ...
    }
}
```

SSL证书链

虽然我们的证书已经被权威证书机构签名，但是还有有一些浏览器提示不能验证证书的签名。这可能是因为给我们证书签名的权威证书机构使用的是他们签过名的中间证书，而这些中间证书可能并没有内置到浏览器中。这种情况下我们应该将权威证书机构提供的一系列中间证书连接到我们的服务器证书文件中。我们的服务器证书必须出现在中间证书之前：

```
$ cat www.example.com.crt bundle.crt > www.example.com.chained.crt
```

然后使用这个证书来替代之前的服务器证书：

Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD

```
server {
    listen              443 ssl;
    server_name         www.example.com;
    ssl_certificate     www.example.com.chained.crt;
    ssl_certificate_key www.example.com.key;
    ...
}
```

浏览器通常会缓存这些接收到并且已经被内置权威证书验证过的中间证书。然后我们的服务器证书可以被这些中间证书验证，这样浏览器就不会报警告了。要确保服务器会发送完整的证书链到客户端我们可以使用openssl客户端工具进行验证：

```
$ openssl s_client -connect www.godaddy.com:443
...
Certificate chain
 0 s:/C=US/ST=Arizona/L=Scottsdale/1.3.6.1.4.1.311.60.2.1.3=US
     /1.3.6.1.4.1.311.60.2.1.2=AZ/O=GoDaddy.com, Inc
     /OU=MIS Department/CN=www.GoDaddy.com
     /serialNumber=0796928-7/2.5.4.15=V1.0, Clause 5.(b)
   i:/C=US/ST=Arizona/L=Scottsdale/O=GoDaddy.com, Inc.
     /OU=http://certificates.godaddy.com/repository
     /CN=Go Daddy Secure Certification Authority
     /serialNumber=07969287
 1 s:/C=US/ST=Arizona/L=Scottsdale/O=GoDaddy.com, Inc.
     /OU=http://certificates.godaddy.com/repository
     /CN=Go Daddy Secure Certification Authority
     /serialNumber=07969287
   i:/C=US/O=The Go Daddy Group, Inc.
     /OU=Go Daddy Class 2 Certification Authority
 2 s:/C=US/O=The Go Daddy Group, Inc.
     /OU=Go Daddy Class 2 Certification Authority
   i:/L=ValiCert Validation Network/O=ValiCert, Inc.
```

Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD

```
     /OU=ValiCert Class 2 Policy Validation Authority
     /CN=http://www.valicert.com//emailAddress=info@valicert.com
...
```

本例中主体为`s`的`www.godaddy.com`的服务器证书`#0`是由证书发行者`i`签名的，而`i`又是`#1`证书的主体。证书`#1`的签发者又是证书`#2`的主体。而证书`#2`是由权威证书发行商`ValliCert Inc`签名的，后者的证书是内置于我们的浏览器中

如果没有将中间证书添加到服务器证书中，则只有服务证书`#0`被显示

A Single HTTP/HTTPS Server

我们可以将含有`ssl`配置项值的`listen`配置项加入到一个`server`配置块中来让这个server可以同时处理HTTP和HTTPS的请求：

```
server {
    listen              80;
    listen              443 ssl;
    server_name         www.example.com;
    ssl_certificate     www.example.com.crt;
    ssl_certificate_key www.example.com.key;
    ...
}
```

在0.7.13版本之前的nginx需要使用`ssl`配置项来启用server的HTTPS处理能力，所以被配置`ssl`配置项的server配置块就不能处理非HTTPS的请求；现在可以用在`listen`配置项后面加上`ssl`配置值的方法来启用HTTPS所以可以使用上面的配置方法

基于主机名的**HTTPS**服务器

如果有多个使用HTTPS的servers需要在同一个IP地址接收请求的话就会有问题：

Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD

```
server {
    listen          443 ssl;
    server_name     www.example.com;
    ssl_certificate www.example.com.crt;
    ...
}

server {
    listen          443 ssl;
    server_name     www.example.org;
    ssl_certificate www.example.org.crt;
    ...
}
```

如果使用上面的配置方法，则客户端会收到默认的服务器证书，在本例中是`www.example.com`，即使你请求的是第二个server的主机名。这是由于SSL协议特定导致的，因为在nginx收到客户端的http请求之前（能拿到客户都请求中的HOST字段值），首先要建立SSL连接，这时候就需要服务器端证书了。所以nginx只能提供默认的服务器证书

解决这个问题最好的方法是给每一个启用HTTPS的server使用不同的IP地址：

```
server {
    listen          192.168.1.1:443 ssl;
    server_name     www.example.com;
    ssl_certificate www.example.com.crt;
    ...
}

server {
    listen          192.168.1.2:443 ssl;
    server_name     www.example.org;
    ssl_certificate www.example.org.crt;
```

Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD

```
    ...
}
```

注意还有一些关于HTTPS upstreams的特殊的设置（`proxy_ssl_ciphers /proxy_ssl_protocols proxy_ssl_session_reuse`）可以用来调优Nginx和上游服务器之间的SSl传输

多个主机名使用同一个证书

有另外一些方法来让多个HTTPS servers使用同一个IP地址，但他们都有一些不足的地方。一种方法是在证书的`SubjectAltName`字段使用多个主机名，比如`www.example.org` `www.example.com`。但是要注意的是`SubjectAltName`字段的长度是有限制的

另外一种方式是使用通配型的证书，比如`*.example.org`。这样这个证书就可以验证所有`example.org`的子域名（主机名）；但是这里的通配符`*`只能指代一级，即不能用这个证书验证`www.subdomain.example.org`主机名。证书的`SubjectAltName`字段值可以混合精确的主机名和通配的主机名

这样我们可以将指定证书的配置项放在`http`配置块中：

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

server {
    listen          443 ssl;
    server_name     www.example.com;
    ...
}

server {
    listen          443 ssl;
    server_name     www.example.org;
    ...
}
```

Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD

Server Name Indication

另外一在同一个IP地址上支持多个HTTPS servers的方法是使用TLS的SNI扩展，但是目前支持这个扩展的浏览器比较少

兼容性提示

- The SNI support status has been shown by the “-V” switch since versions 0.8.21 and 0.7.62.
- The ssl parameter of the listen directive has been supported since version 0.7.14. Prior to version 0.8.21 it could only be specified along with the default parameter.
- SNI has been supported since version 0.5.32.
- The shared SSL session cache has been supported since version 0.5.6.
- From version 0.7.65, 0.8.19 and later the default SSL protocols are SSLv3, TLSv1, TLSv1.1, and TLSv1.2 (if supported by the OpenSSL library).
- From version 0.7.64, 0.8.18 and earlier the default SSL protocols are SSLv2, SSLv3, and TLSv1.
- From version 1.0.5 and later the default SSL ciphers are HIGH:!aNULL:!MD5
- From version 0.7.65, 0.8.20 and later the default SSL ciphers are HIGH:!ADH:!MD5
- From version 0.8.19 the default SSL ciphers are ALL:!ADH:RC4+RSA:+HIGH:+MEDIUM
- From version 0.7.64, 0.8.18 and earlier the default SSL ciphers are ALL:!ADH:RC4+RSA:+HIGH:+MEDIUM:+LOW:+SSLv2:+EXP

Nginx作为一个**SSL**代理前端服务器（**TCP**转发）

Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD

本节设置需要Nginx plus R6以上版本

当Nginx作为前端代理服务器时，我们可以在Nginx上启用SSL来接收用户的HTTPS连接请求。Nginx会为后端应用服务器解密用户请求数据，并且加密后端返回的响应数据；而Nginx和后端服务器则可以使用非SSL连接

配置示例：

```
stream {
    upstream stream_backend {
         server backend1.example.com:12345;
         server backend2.example.com:12345;
         server backend3.example.com:12345;
    }

    server {
        listen                  12345 ssl;
        proxy_pass              stream_backend;

        ssl_certificate         /etc/ssl/certs/server.crt;
        ssl_certificate_key     /etc/ssl/certs/server.key;
        ssl_protocols           SSLv3 TLSv1 TLSv1.1 TLSv1.2;
        ssl_ciphers             HIGH:!aNULL:!MD5;
        ssl_session_cache       shared:SSL:20m;
        ssl_session_timeout     4h;
        ssl_handshake_timeout 30s;
    …
     }
}
```

`ssl_handshake_timeout`指定了ssl握手阶段交换的数据的缓存时间，默认是60s这里改为30s

还可以启用`ssl_session_tickets`功能将session会话缓存到客户端

Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD

HTTP负载均衡

在多种部署场景中nginx都可以作为一种非常高效的HTTP负载均衡应用

将流量分发到一组服务器

需要先在`http`配置块中使用`upstream`配置项定义一个服务器组，然后可以在`location`中使用这个主机组

`upstream`配置项后面跟主机组名，然后是一对大括号，在大括号中使用`server`配置项来定义主机的地址（可以是ip或者域名），每个配置项后面还可以跟一个或者多个参数：

```
http {
    upstream backend {
        server backend1.example.com weight=5;
        server backend2.example.com;
        server 192.0.0.1 backup;
    }
}
```

要将请求分发到主机组，需要在`proxy_pass`配置项（依赖于使用的协议，可以是`fastcgi_pass`,`memcached_pass`,`uwsgi_pass`,`scgi_pass`）：

```
server {
    location / {
        proxy_pass http://backend;
    }
}
```

选择一种负载均衡方法

Nginx支持如下四种负载均衡调度方法：

Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD

1. **`round-robin`方法**：根据权重将请求均衡的分发到组中所有服务器。这是默认方法，不需要任何配置项就启用这种方法
2. **`least_conn`方法**：根据权重将请求分配给当前活动连接数量最少的服务器。需要在`upstream`中使用`least_conn`;配置项
3. **`ip_hash`方法**：根据客户端ip地址决定请求分发服务器。一般使用ipv4地址的前三个8位值或者ipv6的整个地址值来计算hash值。这种方法保证从一个ip地址来的请求一定会转发到后端的一台服务器，除非不可用。需要在`upstream`中使用`ip_hash`;配置项
4. **`generic_hash`方法**：根据用户自定义的key来决定请求如何转发，key可以是文本或者变量或者它们的组合。比如key可以是ip地址和端口或者URI：

```
upstream backend {
    hash $request_uri consistent;

    server backend1.example.com;
    server backend2.example.com;
}
```

可选的`consistent`参数启用ketama一致hash负载均衡。请求会被按照用户自定义的hashed key均匀的分发到`upstrem`中的服务器。如果upstream中增加或者移除一台服务器，则只有很少一部分keys需要重新映射

服务器权重

默认情况下nginx会使用`round-robin`方法并且根据权重将请求转发到组中的服务器。server配置参数中的`weight`参数指定了每个服务器的权重，默认是`1`：

```
upstream backend {
    server backend1.example.com weight=5;
    server backend2.example.com;
```

Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD

```
    server 192.0.0.1 backup;
}
```

上面的例子中backend1的权重是5，而backend2的权重是1，所以如果有6个请求第一个服务器会分配到5个儿第二个服务器分配到一个；另外还有一个backup服务器在上面两个服务器又任何一个不可用时nginx会将请求也转发到这台服务器

服务器慢启动

满启动可以避免将请求大量的转发到一个刚刚恢复的服务器，有可能导致服务器再次宕机。可以在`server`配置项中添加`slow_start`配置参数来启用慢启动服务：

```
upstream backend {
    server backend1.example.com slow_start=30s;
    server backend2.example.com;
    server 192.0.0.1 backup;
}
```

时间指定了服务器会在多久时间内恢复权重 如果主机组中只有一台服务器，则`max_fails`， `fail_timeout`和`slow_start`参数都会被忽略

启用**Session**保持

Session保持意味着nginx可以识别用户session并且将同一个session的请求转发到服务器组中的同一台服务器。nginx支持3种session保持的方法，在upstream配置块中使用`sticky`配置项指定

`Sticky cookie``方法`：使用这种方法，nginx会在第一个响应信息中加入session cookie，并且标识出这个响应来自于服务器组中的哪一台。当这个客户端再次发出请求时会在请求头中带上这个cookie值，nginx会将这个请求发送至上一次处理这个请求的服务器：

Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD

```
upstream backend {
    server backend1.example.com;
    server backend2.example.com;

    sticky cookie srv_id expires=1h domain=.example.com path=/;
}
```

本例中的srv_id参数值指定了要设置和检查的cookie名，可选的expires参数指定了浏览器应该保存这个cookie的时间，可选的domain参数指定了cookie的域，可选的`path`参数指定了cookie设置的路径；这是最简单的一种session保持方法

`Sticky route`方法：使用这种方法，nginx会给发来请求的客户端分配一个`route`。这个route会和客户端的cookie或者URI关联，下一请求到来时会根据这个关联的信息获取到route，然后将请求发送给对应的服务器：

```
upstream backend {
    server backend1.example.com route=a;
    server backend2.example.com route=b;

    sticky route $route_cookie $route_uri;
}
```

`Cookie learn`方法：使用这种方法，nginx会检查请求和响应中的`session`标识。然后nginx会记住这个标识的请求和响应来自于哪个服务器。通常这种标识也是通过`HTTP cookie`的方法传递的：

```
upstream backend {
   server backend1.example.com;
   server backend2.example.com;

   sticky learn
       create=$upstream_cookie_examplecookie
```

Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD

```
        lookup=$cookie_examplecookie
        zone=client_sessions:1m
        timeout=1h;
}
```

上面例子中，nginx会通过在响应的信息中设置一个`EXAMPLECOOKIE` cookie的方法来创建一个session

必选参数`create`指定了创建session时依赖的变量。上例中是根据upstream服务器发过来的`EXAMPLECOOKIE` cookie创建的session

必选参数`lookup`指定了如何查找是否存在这个session。上例中是根据客户端发来的`EXAMPLECOOKIE` cookie来查找

必选参数`zone`指定了用来保存sticky session信息的共享内存区域。上例中这个区域名是`client_session`并且大小限制为1M

这是一种比较复杂的session保持方法，不需要给客户端增加新的cookie，所有的数据都保存在nginx

限制连接数量

使用nginx，我们可以在server配置项后面使用`max_conns`参数来限制允许的连接数量

如果已经到达最大的连接数，则nginx会将其他的连接请求放入一个队列，这个队列可以在`upstream`配置块中使用`queue`配置项来指定：

```
upstream backend {
    server backend1.example.com  max_conns=3;
    server backend2.example.com;

    queue 100 timeout=70;
}
```

Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD

`queue`配置项后面的参数表示队列中的请求的最大数量，`timeout`参数指定了如果在队列中70秒还未得到处理则给客户端返回一个错误

Note that the max_conns limit will be ignored if there are idle keepalive connections opened in other worker processes. As a result, the total number of connections to the server may exceed the max_conns value in a configuration where the memory is shared with multiple worker processes.

被动的健康监控

如果nginx认为一个服务器不可达，则不会将请求转发到这台服务器，直到它认为这台服务器已经恢复。下面的参数可以放在server配置项后面来设置服务器不可达的条件

`max_fails`参数指定了要确定服务器确实处于不可达状态应该发送的请求的数量

`fail_timeout`服务器会在这个时间间隔内发送上一个参数指定的次数的请求，所以从上次检测到服务器处于不可达后在这个时间内nginx认为服务器处于上次监测的状态，也就不会将请求转发至这台服务器 ？？？？

默认情况下nginx会在10秒内进行一次检测，所以如果一次请求未得到相应，则在10秒内nginx认为这台服务器处于不可达状态：

```
upstream backend {
    server backend1.example.com;
    server backend2.example.com max_fails=3 fail_timeout=30s;
    server backend3.example.com max_fails=2;
}
```

主动的健康监控

Nginx可以周期性的发送特定的请求到后端服务器，然后检查响应是否符合条件来确定服务器状态

Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD

要启用这种类型的检查，需要在使用这个`upstream`的`location`中使用`health_check`配置项。而且在服务器组中需要指定`zone`配置项：

```
http {
    upstream backend {
        zone backend 64k;

        server backend1.example.com;
        server backend2.example.com;
        server backend3.example.com;
        server backend4.example.com;
    }

    server {
        location / {
            proxy_pass http://backend;
            health_check;
        }
    }
}
```

本例中启用了主动健康检查，每5秒nginx会发送一个`/`请求到后端的每一台服务器；如果有任何错误发生则nginx不会将请求再转发到这台服务器，知道下次检测成功

`zone`配置项定义了一个在worker进程之间共享的内存区域，用来存储组的配置信息。这可以让worker进程使用相同设置的计数器来跟踪组中服务器的响应。zone配置项也使这个服务器组可动态配置（dynamically configurable）

`health_check`配置项也可以使用下面的参数覆盖默认配置：

```
location / {
    proxy_pass http://backend;
```

Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD

```
    health_check interval=10 fails=3 passes=2;
}
```

上面配置中，两次检测的时间设置为10秒，连续三次检查失败则认为服务器不可达。连续两次检查成功则认为服务器恢复

也可以指定检查的URI：

```
location / {
    proxy_pass http://backend;
    health_check uri=/some/path;
}
```

指定的URI会被附加到本server指定的主机名或者IP地址后面来组成完整的URL

另外，也可以自己指定nginx认为服务器正常的条件。用`match`配置块来指定条件，然后在`health_check`配置项中来使用这个条件：

```
http {
    ...

    match server_ok {
        status 200-399;
        body !~ "maintenance mode";
    }

    server {
        ...

        location / {
            proxy_pass http://backend;
```

Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD

```
            health_check match=server_ok;
        }
    }
}
```

上例中，如果返回码是`200-399`并且body中不含`maintenance mode`则认为服务器是正常的

match配置块中可以指定检查状态码，响应头和响应体中的内容；使用这个配置项我们可以检查状态码是否在一定范围内，响应头中是否包含特定头域，响应体是否匹配正则表达式；在`match`块中可以指定一个返回码检查，一个响应体检查和多个相应头域检查。

下面是另外一些例子：

```
match welcome {
    status 200;
    header Content-Type = text/html;
    body ~ "Welcome to nginx!";
}
match not_redirect {
    status ! 301-303 307;
    header ! Refresh;
}
```

健康检查也可以在非http协议中使用，比如`FastCGI`，`uwsgi`，`SCGI`和`memcached`

在多个**worker**进程间共享数据

如果upstream中不包含`zone`配置项，则每一个worker进程都有一份服务器组配置信息和一系列计数器的维护信息。计数器包括服务器组中每台服务器的连接数和请求一个服务器的失败尝试次数。所以服务器组配置信息是不可变的

Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD

如果upstream中包含`zone`配置项，则服务器组的配置信息被放置在所有worker进程可访问的共享内存区域。这样服务器组就变的可配置，因为所有worker进程获取到相同的信息并且使用相同的计数器

在服务器组的健康检查和运行时配置中zone是必须的。当前其他的配置也可以从中获益

比如，如果服务器组的配置信息没有共享，每一个worker进程都会维护一份自己的关于服务器失败已尝试次数的计数器。这样的话这个请求的处理会依赖于这个worker进程。如果这个worker进程因为某种原因退出，由另外一个进程来处理这个请求，那之前维护的计数器信息都会丢失。这样nginx可能要花费更多的资源来校正这些信息

如果没有使用zone配置项的话，`least_conn`负载均衡方法可能不会正常工作

设置**zone**的大小 因为不同的应用模式所以zone也没有精确的设置。每种特性，比如sticky cookie/route/learn负载均衡，健康检查或者re-resolving会影响zone的大小

比如，256kb的zone用在sticky_route会话保持方法和单个健康检查可以容纳：

- 128 servers (adding a single peer by specifying IP:port);
- 88 servers (adding a single peer by specifying hostname:port, hostname resolves to single IP);
- 12 servers (adding multiple peers by specifying hostname:port, hostname resolves to many IPs).

在线配置

使用特定的HTTP接口，我们可以在线配置服务器组的信息。可以使用配置命令查看组中的所有服务器或者特定的服务器，更改一个特定服务器的参数，或者添加和移除一个服务器

`设置允许在线配置`

1. 在upstream中包含zone配置项。zone配置项启用一个共享内存区域，并且设置zone名和大小。服务器组的配置信息保存在这里，这样所有的worker进程都可以获取到同样的信息：

Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD

```
http {
    ...
    upstream appservers {
        zone appservers 64k;
        server appserv1.example.com      weight=5;
        server appserv2.example.com:8080 fail_timeout=5s;
        server reserve1.example.com:8080 backup;
        server reserve2.example.com:8080 backup;
    }
}
```

2. 在单个的location中设置upstream_conf配置项：

```
server {
    location /upstream_conf {
        upstream_conf;
        allow 127.0.0.1;
        deny  all;
    }
}
```

> 强烈建议你增加访问限制

一个完整的配置

```
http {
    ...
    # Configuration of the server group
    upstream appservers {
        zone appservers 64k;
```

Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD

```
        server appserv1.example.com      weight=5;
        server appserv2.example.com:8080 fail_timeout=5s;

        server reserve1.example.com:8080 backup;
        server reserve2.example.com:8080 backup;
    }

    server {
        # Location that proxies requests to the group
        location / {
            proxy_pass http://appservers;
            health_check;
        }

        # Location for configuration requests
        location /upstream_conf {
            upstream_conf;
            allow 127.0.0.1;
            deny  all;
        }
    }
}
```

在线配置

配置命令是以http请求的方式发出的。URL需要能访问到`upstream_conf`配置项所在的location。请求还需要加上一些参数来指定要配置的服务器组

比如，要查看一个服务器组中的所有backup server： http://127.0.0.1/upstream_conf?upstream=appservers&backup=

要添加一个新的服务器到这个组中： http://127.0.0.1/upstream_conf?add=&upstream=appservers&server=appserv3.example.com:8080&weight=2&max_fails=3

Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD

要移除一个服务器需要指定这个服务器的id： http://127.0.0.1/upstream_conf?remove=&upstream=appservers&id=2

要更改特定的服务的配置信息： http://127.0.0.1/upstream_conf?upstream=appservers&id=2&down=

其他的配置方法可以参考nginx.org

> 注意：从nginx1.9开始nginx支持TCP负载均衡，详细信息可以参考nginx.org

使用代理协议（**PROXY protocol**）

PROXY protocol可以让nginx接收到通过代理服务器或者负载均衡设备传过来的客户端连接信息

经过PROXY protocol传过来的信息一般是客户端的IP、代理服务器的IP以及端口。后端服务器在某些情况下需要这些信息。使用 `PROXY protocol nginx` 可以从 `SSL HTTP/2 SPDY Websocket` 和 `TCP` 协议中获取最初的客户端信息

启用代理协议

要让nginx在 `SSL HTTP/2 SPDY WebSocket` 协议中接收 `PROXY protocol` 需要进行下面的配置

1. 配置nginx接受 `PROXY protocol` 头，需要在 `listen` 配置项中添加 `proxy_protocol` 参数：

```
server {
    listen 80   proxy_protocol;
    listen 443  ssl proxy_protocol;
    ...
}
```

2. 在 `set_real_ip_from` 配置项中指定代理或者负载均衡服务器所在的网段：

```
server {
    ...
```

Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD

```
    set_real_ip_from 192.168.1.0/24;
    ...
}
```

3. 在`real_ip_header`配置项中增加`proxy_protocol`参数来保存客户端ip和端口：

```
server {
    ...
    real_ip_header proxy_protocol;
}
```

4. 使用`proxy_set_header`配置项将客户的IP地址从nginx发送至后端服务器：

```
proxy_set_header X-Real-IP       $proxy_protocol_addr;
proxy_set_header X-Forwarded-For $proxy_protocol_addr;
```

5. 在`log_format`配置项中增加`$proxy_protocol_addr`变量：

```
http {
    ...
    log_format combined '$proxy_protocol_addr - $remote_user [$time_local] '
                        '"$request" $status $body_bytes_sent '
                        '"$http_referer" "$http_user_agent"';
}
```

限制访问被代理的**HTTP**资源

限制访问（**Restricting**）

Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD

Nginx可以根据客户的IP地址来决定是否允许访问特定的资源，也可以使用基本的HTTP验证

使用`allow`和`deny`配置项可以限制或者允许特定的IP或者一个网段的IP的访问

```
location / {
    deny  192.168.1.2;
    allow 192.168.1.1/24;
    allow 127.0.0.1;
    deny  all;
}
```

要启用验证可以使用`auth_basic`配置项，这样用户需要使用合法的用户名和密码来访问特定资源。合法的用户名和密码所在的文件使用`auth_basic_user_file`配置项指定：

```
server {
    ...
    auth_basic "closed website";
    auth_basic_user_file conf/htpasswd;
}
```

可以在特定的`location`中添加`auth_basic`来设置这个`location`的访问不需要验证：

```
server {
    ...
    auth_basic "closed website";
    auth_basic_user_file conf/htpasswd;

    location /public/ {
        auth_basic off;
    }
}
```

Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD

要混合IP地址限制和认证需要使用`satisfy`配置项。默认情况下设置为`all`，客户端需要满足所有的条件才可以访问；如果设置为`any`则表示至少需要满足一个条件才允许访问：

```
location / {
    satisfy any;

    allow 192.168.1.0/24;
    deny  all;

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

限制访问（**Limiting**）

Nginx可以限制：

1. 每一个指定key value的连接数（比如每个IP地址）
2. 每一个指定key value的请求速率（特定时间内可以处理的请求数量）
3. 每一个连接的下载速度

> 注意：因为可能有的多个客户端使用同一个IP地址连接所以限制IP的方法并不很明智

`限制连接数`

要限制连接数，首先需要使用`limit_conn_zone`配置项定义一个`key`并设置共享内存区域参数（所有的worker进程使用这个共享内存区域获取相同的计数器）。第一个参数是一个表达式被计算后作为key。第二个参数指定了`zone`名字和大小：

Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD

```
limit_conn_zone $binary_remote_address zone=addr:10m;
```

然后在一个`location`中使用`limit_conn`配置项指定这个限制，也可以在`server`或者整个`http`配置块中使用。指定共享内存区域名作为第一个参数，每一个key允许的连接数作为第二参数：

```
location /download/ {
    limit_conn addr 1;
}
```

上面的例子中以每一个IP地址作为限制，因为使用`$binary_reomte_address`作为第一个参数。对于一个服务器允许的连接数限制可以使用`$server_name`变量：

```
http {
    limit_conn_zone $server_name zone=servers:10m;

    server {
        limit_conn servers 1000;
    }
}
```

限制请求速率

要限制请求的速率，首先使用`limit_req_zone`配置参数设置key和共享内存区域来保存计数器信息：

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

key的指定和`limit_conn_zone`的方法一样。`rate`参数可以指定单位时间的请求数，比如每秒`（r/s）`或者每分钟`（r/m）`。当没秒请求数少于一个时候需要使用每分钟的单位

Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD

定义了共享内存区域后，就可以使用`limit_req`配置参数来限制一个`location`或者`server`或者全局的请求速率：

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

上面的配置中，nginx每秒只处理一个请求。如果请求数量大于限制值，则这些请求会被放置在一个队列中并且被延后处理。突发请求数超过burst设置的话会返回`503`错误

如果不想超过设置的请求被延后可以使用`nodelay`参数：

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

`限制带宽`

要限制每一个连接可使用的带宽，可以使用`limit_rate`配置参数：

```
location /download/ {
    limit_rate 50k;
}
```

上面的设置会限制单个的连接的最大带宽是50k。但是用户可以打开多个连接来突破这个限制。所以要限制每个用户的下载速度的话，每个用户的连接数也需要限制，比如（需要在前面定义共享区域addr）：

```
location /download/ {
    limit_conn addr 1;
    limit_rate 50k;
}
```

Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD

使用`limit_rate_after`配置参数可以让用户在下载完多少字节内容有才应用限制（这样可以让用户快速下载完一个文件头）：

```
limit_rate_after 500k;
limit_rate 20k;
```

下面是一个混合配置示例：

```
http {
    limit_conn_zone $binary_remote_address zone=addr:10m

    server {
        root /www/data;
        limit_conn addr 5;

        location / {
        }

        location /download/ {
            limit_conn addr 1;
            limit_rate 1m;
            limit_rate 50k;
        }
    }
}
```

日志和监控

设置错误日志

Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD

Nginx会将运行过程中产生的不同级别的问题信息写入错误日志。`error_log`配置项可以控制将日志信息写入文件，标准错误输出或者系统日志；并且可以指定什么级别的日志信息才会被记录。默认情况下，错误日志放在`logs/error.log`（相对于nginx安装根目录），并且会记录指定的日志等级以上的信息

下面的配置将要记录的日志的最小等级由`error`改为`warn`：

```
error_log logs/error.log warn;
```

本例中所有的`warn`、`error`、`crit`、`alert`和`emerg`等级的日志会被记录

默认的错误日志的设置对全局起作用。要改变这个默认设置`error_log`配置项需要放置在`main`配置块中。`main`配置块中的设置也总是会被其他配置块继承。`error_log`配置项可以放置在`http`、`stream`、`server`和`location`配置块中来取消从高层次继承过来的设置。错误日志只会被记录在最内层指定的文件中；但是如果在同一个级别的配置块中指定多个`error_log`配置项则日志会被写入多个指定的目标中

> 注意：在同一层级配置块中使用多个`error_log`的配置在`1.5.2`之后的nginx才支持

设置访问日志

Nginx会在请求被处理之后立刻将客户端的请求信息记录至访问日志中。默认情况下访问日志放置在`logs/access.log`，并且会按配置项`combined`指定的格式来写入日志信息。要改变默认设置，使用`log_format`配置项指定要记录日志的格式，使用`access_log`配置项指定日志的位置和要写入的格式（`log_format`定义的），日志格式的定义使用变量

下面的配置中在默认的日志格式中增加了响应信息的压缩率，将这个日志格式命名为`compression`；然后在一个server中使用这个日志格式：

```
http {
    log_format compression '$remote_addr - $remote_user [$time_local] '
                           '"$request" $status $body_bytes_sent '
```

Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD

```
                          '"$http_referer" "$http_user_agent" "$gzip_ratio"';

    server {
        gzip on;
        access_log /spool/logs/nginx-access.log compression;
        ...
    }
}
```

下面例子的日志格式跟踪nginx和后端服务器的不同的时间值，可以在后端响应比较慢的情况下来帮助分析问题。下面是可使用的时间值：

- the time spent on establishing a connection with an upstream server. This can be done with the $upstream_connect_time variable
- the time between establishing a connection and receiving the first byte of the response header from the upstream server. This can be done with the $upstream_header_time variable,
- the time between establishing a connection and receiving the last byte of the response body from the upstream server This can be done with the $upstream_response_time variable
- full time spent on processing a request with the $request_time variable

> 所有的值都是秒和毫秒的粒度

```
http {
    log_format upstream_time '$remote_addr - $remote_user [$time_local] '
                             '"$request" $status $body_bytes_sent '
                             '"$http_referer" "$http_user_agent"'
                             'rt=$request_time uct="$upstream_connect_time" uht="$upstream_header_t

    server {
        access_log /spool/logs/nginx-access.log upstream_time;
        ...
```

Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD

```
    }
}
```

下面是理解这些值的规则：

- 如果一个请求是由几个服务器处理的，则这个变量是用逗号分开的几个值
- 如果有一个内部重定向将请求从一个后端服务器组转发到另外一个组，则变量的值用分号分隔
- 如果一个请求不能到达后端服务器或者没有收到一个完整的响应头，则变量包含“0”
- 如果连接后端服务器时发生内部错误或者返回的信息是从缓存中获取的，则变量包含`-`

可以为日志信息启用缓存或者缓存那些名字中包含变量的频繁使用的文件描述符（打开日志文件的）来优化日志记录。要启用缓存需要在`access_log`配置项后面使用`buffer`参数设置缓存的大小。这样缓存中的日志信息会在缓存不足以放置接下来的一条日志记录时被写入日志文件（也可以使用参数进行控制cases）。要启用日志文件描述符缓存，需要使用`open_log_file_cache`配置项

和`error_log`一样，`access_log`配置也会由小的配置块中的设置继承或者覆盖大的配置块的设置；如果在同一配置层级指定多个`access_log`配置项则日志被写入多个目标

`使用条件日志`
条件日志可以从所有要记录日志信息中排除不重要的日志。Nginx中在`access_log`配置项中使用`if`参数可以启用条件日志

比如，下面的配置可以排除那些返回`2xx`或者`3xx`状态码的请求的记录：

```
map $status $loggable {
    ~^[23]  0;
    default 1;
}
```

Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD

```
access_log /path/to/access.log combined if=$loggable;
```

记录到**syslog**

Syslog是一个标准的系统信息记录系统，可以将不同的设备的日志信息记录到单个的日志服务器。可以使用syslog配置参数加上前置的`error.log`或者`access.log`来启用syslog日志记录

日志信息被发送至`server=`指定的值中，这个值可以是一个主机名、一个IP地址、或者一个Unix-domain套接字中。主机名和IP地址可以指定一个端口，默认端口是514；Unix-domain套接字路径需要加`unix:`前缀：

```
error_log server=unix:/var/log/nginx.sock debug;
access_log syslog:server=[2001:db8::1]:1234,facility=local7,tag=nginx,severity=info;
```

上面的例子中`debug`以上级别的错误日志信息会被写入一个unix套接字中；而访问日志信息被发送至由ipv6地址和端口指定的服务器

`facility=`参数指定了要记录日志信息的应用类型，默认是local7。其他的值可以是：`auth, authpriv, daemon, cron, ftp, lpr, kern, mail, news, syslog, user, uucp, local0 … local7`

`tag=`参数会增加一个自定义的`tag`到日志信息中，本例中tag为nginx

`severity=`参数指定了访问日志的系统日志严重性级别。可能的值按严重性增加依次是：`debug, info, notice, warn, error (default), crit, alert, and emerg`

实时活跃性监控

Nginx提供了一个可以监控实时系统活跃性的接口，通过这个接口我们可以查看HTTP和TCP后端服务器的负载和性能信息；还可以查看到下面这些信息：

Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD

- Nginx版本、启动时间和标识
- 累计的和当前的连接数量和请求数量
- 每一个`status_zone`的请求和响应的计数
- 每一个`dynamically configured group`中的每一个服务器的请求和响应的计数，以及健康状况和启动时间统计信息
- 每一个服务器的统计信息和当前状态和服务器数量（不可用状态的数量）
- 每一个命名的`cache zone`的统计信息

> 完整的信息列表可以查看here.

统计信息可以从Nginx Plus包中的status.html页面查看

使用简单的RESTful JSON接口，可以很容易将这些信息连接到一个实时面板和第三方的监控工具中

`启用实时活跃性监控`
要启用实时监控和JSON接口，需要指定一个`location`，这个`location`的URI为`/status`并且在配置块中包含`status`配置项。要使用`status.html`页面，还需要指定另外一个`location`让nginx可以定位到这个静态页面：

```
server {
    listen 127.0.0.1;
    root /usr/share/nginx/html;

    location /status {
        status;
    }

    location = /status.html {
    }
}
```

Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD

使用**RESTful JSON**接口

如果你访问`/status`，nginx会返回一个包含实时数据的JSON文档

JSON文档中的任何元素表示的状态信息都可以使用不同的URL请求来获取：

```
http://127.0.0.1/status
http://127.0.0.1/status/nginx_versionhttp://127.0.0.1/status/caches/cache_backendhttp://127.0.0.1/s
```

原文地址：**Nginx admin guide and tutorial**

© 2019 GitHub, Inc. Terms Privacy Security Status Help

Contact GitHub Pricing API Training Blog About

Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD