Web Server 有限责任公司, OGRN 1227700436578, INN 9704151517
俄罗斯，莫斯科，维亚茨卡亚街 27 号 7 栋，127015

用户指南

版本 *1.10.0*

Web Server 有限责任公司

2025 年 07 月 07 日

目录

1 注释 **2**

2 一般信息 **3**

3 配置 **5**

3.1 通用信息 . . . 5
3.1.1 配置文件 . . . 5
3.1.2 运行时控制 . . . 8
3.1.3 连接、会话、请求、日志 . . . 11
3.2 参考和索引 . . . 20
3.2.1 内置模块 . . . 20
3.2.2 内置变量 . . . 485
3.2.3 快速访问 Angie 指令和变量 . . . 489
3.3 操作指南 . . . 492
3.3.1 如何从 nginx 迁移到 Angie . . . 492
3.3.2 ACME 配置 . . . 496
3.3.3 SSL 配置 . . . 506
3.3.4 Console Light 网络监控面板 . . . 511
3.3.5 配置 Prometheus 仪表板 . . . 526

4 故障排除 **528**

4.1 调试日志 . . . 528
4.1.1 指令位置 . . . 530
4.1.2 记录特定地址 . . . 532
4.1.3 循环内存缓冲区 . . . 532
4.2 核心转储 . . . 532
4.2.1 Linux: systemd . . . 532
4.2.2 Linux: 手动配置 . . . 533
4.2.3 FreeBSD . . . 534

5 知识产权 **535**

Web Server 有限责任公司, OGRN 1227700436578, INN 9704151517
俄罗斯，莫斯科，维亚茨卡亚街 27 号 7 栋，127015

索引 536

og:description

操作 Angie PRO 软件的信息

Web Server 有限责任公司, OGRN 1227700436578, INN 9704151517
俄罗斯，莫斯科，维亚茨卡亚街 27 号 7 栋，127015

章节 1

注释

本文档包含操作 Angie PRO 软件所需的信息。

章节 2

一般信息

Angie PRO 是唯一在俄罗斯开发和本地化的商业 Web 服务器。

Web 服务器是一类软件，通过 HTTP 协议为最终用户提供对网络资源的访问。例如，Angie PRO 可用于运营网站、移动应用程序、地铁自助服务终端以及长途列车上的多媒体系统。每当用户打开网站、使用移动应用程序、与地铁里的自助服务终端互动，或甚至与" 萨普桑" 列车上的多媒体系统交互时，用户的请求都可以由 Angie PRO 处理。

Angie PRO 是:

- **通用 Web 服务器**。用 C 语言编写。
- **L4-L7 负载均衡器**。允许在 TCP/UDP 协议和 HTTP 之间进行负载均衡。
- **代理和缓存服务器**。通过灵活的缓存机制加快 Web 服务的运行。
- **可在所有流行平台上使用**。在 Alpine、Debian、Oracle、RED OS、Rocky 和 Ubuntu 上编译和测试。
- **高性能**。世界上最有效的 Web 服务器之一。

选择 Angie PRO 的理由:

- **与 NGINX OSS 兼容**。 Angie PRO 完全兼容 Nginx，允许任何现有的 Nginx 用户在没有重大成本或服务停机的情况下过渡到 Angie PRO。
- **增强的统计信息和实时监控**。 Angie PRO 提供完整的实时服务器负载监控，能够根据负载配置文件动态管理配置，确保服务的完全可用性。
- **动态配置代理服务器组**。允许通过方便的 REST 接口管理代理服务器组的设置，无需服务中断。
- **缓存元素移除**。提供通过用户友好的 API 移除缓存元素的能力，无需服务停机。

- **对代理服务器的主动健康检查**。检查" 存活状态"，并仅代理到根据指定算法响应的代理服务器组。
- **动态键值存储**。通过 HTTP API 动态管理 Angie PRO 配置变量。
- **动态 DNS 更新**。
- **会话亲和代理**。
- **具有动态第三方模块的仓库**。 Angie PRO 支持大多数 NGINX 第三方模块，并允许无缝安装，确保功能和支持。
- **共享内存区域同步**。能够在 Angie PRO 集群中使用缓存区域、limit_req 等功能。
- **在响应头中隐藏或个性化服务器名称**。能够更改或隐藏 Web 服务器的名称和版本，防止用户看到。

与 Angie PRO 具有类似功能特性的外部软件列表包括 Nginx、Nginx Plus、Apache、Envoy、利用 NGINX 解决方案的产品（OpenResty、Tengine、Cloudflare）以及 Yandex 的云解决方案。

章节 3

配置

本页面包含有关配置 Angie 的文章、参考、索引和说明。

3.1 通用信息

这些文章涵盖了 Angie 的安装和配置，启动和停止 Web 服务器，管理它，以及请求处理和与其他服务器交互的各个方面。

3.1.1 配置文件

Angie 使用基于文本的配置文件。默认情况下，该文件名为 `angie.conf`，根据 --conf-path 构建参数的位置，通常位于 `/etc/angie` 目录下。

配置文件通常由以下上下文组成：

- *events* –一般连接处理
- *http* –HTTP 流量
- *mail* –邮件流量
- *stream* –TCP 和 UDP 流量
- *wasm_modules* –WASM 运行时

放置在这些上下文之外的指令被视为在 `main` 上下文中：

```
user angie; # 在 'main' 上下文中的指令

events {
```

```
    # 连接处理的配置
}

http {

    # 特定于 HTTP 的配置，影响所有虚拟服务器

    server {

        # HTTP 虚拟服务器 1 的配置
        location /one {

            # 处理以 '/one' 开头的 URI 的配置
        }
        location /two {

            # 处理以 '/two' 开头的 URI 的配置
        }
    }

    server {

        # HTTP 虚拟服务器 2 的配置
    }
}

stream {

    # 特定于 TCP/UDP 的配置，影响所有虚拟服务器
    server {

        # TCP 虚拟服务器 1 的配置
    }
}
```

为了简化配置管理，我们建议在主 `angie.conf` 文件中使用*include* 指令引用特定功能文件的内容：

```
include /etc/angie/http.d/*.conf;
include /etc/angie/stream.d/*.conf;
```

继承

一般来说，子上下文（包含在另一个上下文中的上下文，被视为其父上下文）继承在父级定义的指令设置。一些指令可以出现在多个上下文中；在这种情况下，您可以通过在子上下文中包含该指令来覆盖从父级继承的设置。

语法

测量单位

可以使用以下单位来指定大小：

无后缀	字节
k, K	千字节
m, M	兆字节
g, G	吉字节

例如：1024、8k、1m、16g。

时间间隔可以以毫秒、秒、分钟、小时、天等来指定，使用以下后缀：

ms	毫秒
s	秒
m	分钟
h	小时
d	天
w	周
M	月（假定等于 30 天）
y	年（假定等于 365 天）

多个单位可以通过按顺序从最重要到最不重要指定它们来组合成一个值，选用空格分隔。例如，"1h 30m" 指定的持续时间与 "90m" 或 "5400s" 相同。没有后缀的值被解释为秒。建议始终指定后缀。

某些时间间隔只能以秒级精度指定。

指令

每个指令由一个名称和一组参数组成。如果指令的任何部分需要包含空格，它应被引号括起来或转义空格：

```
add_header X-MyHeader "foo bar";
add_header X-MyHeader foo\ bar;
```

如果命名参数需要空格，并且您使用了引号，则其名称也必须用引号括起来：

```
server example.com "sid=server 1";
```

设置哈希

为了有效处理静态数据集，例如服务器名称、*map* 指令值、MIME 类型和请求头名称，Angie 使用哈希表。在启动和每次重新配置时，Angie 确定这些哈希表的最佳大小，以确保存储具有相同哈希值的键的桶大小不超过配置参数（*hash bucket size*）。表的大小以桶为单位测量，并调整直到超过 *hash*

max size 参数。大多数哈希表都有相应的指令来调整这些参数，例如*server_names_hash_max_size* 和*server_names_hash_bucket_size* 用于服务器名称。

hash bucket size 参数与处理器的缓存行大小对齐。这种对齐提高了现代处理器上键搜索的效率，减少了内存访问的次数。如果 *hash bucket size* 等于一个缓存行大小，则在键搜索期间的最大内存访问次数将为两个：一个用于计算桶地址，另一个用于在桶内搜索。因此，如果 Angie 指示应该增加 *hash max size* 或 *hash bucket size*，则首先增加 *hash max size*。

重新加载配置

要应用对配置文件的更改，必须重新加载它。您可以先通过配置语法检查重启 Angie 进程：

```
$ sudo angie -t && sudo service angie restart
```

或者，您可以重新加载服务，以在不中断当前请求处理的情况下应用新配置：

```
$ sudo angie -t && sudo service angie reload
```

3.1.2 运行时控制

要启动 Angie，请使用 **systemd** 运行以下命令：

```
$ sudo service angie start
```

建议事先检查配置语法。操作如下：

```
$ sudo angie -t && sudo service angie start
```

要重新加载配置：

```
$ sudo angie -t && sudo service angie reload
```

要停止 Angie：

```
$ sudo service angie stop
```

安装后，运行以下命令以确保 Angie 正在运行：

```
$ curl localhost:80
```

> **备注**
>
> 开源版本的 Angie 的运行方法可能因安装方法而异。

Angie 有一个主进程和多个工作进程。主进程负责读取和评估配置，并维护工作进程。工作进程负责实际请求处理。Angie 使用基于事件的模型和操作系统相关的机制来高效地在工作进程之间分配请求。工作进程的数量在配置文件中定义，可以在给定配置中是固定的，也可以根据可用 CPU 核心的数量自动调整（参见*worker_processes*）。

在启用配置后，Angie 还会将某些共享内存区域（目前是*proxy_cache_path* 中的 `keys_zone`）在退出前刷写到磁盘，以便新启动的主进程可以以更高的性能恢复它们。如果由于区域大小变化、二进制版本不兼容或其他原因导致恢复失败，Angie 将记录一条警报 (`failed to restore zone at address`)，并且不会使用区域恢复机制。

使用信号

也可以使用信号来控制 Angie。默认情况下，主进程的进程 ID 写入到文件 `/run/angie.pid`。可以在配置时或在 `angie.conf` 中使用*pid* 指令更改此文件名。主进程支持以下信号：

TERM, INT	快速关闭
QUIT	优雅的 关闭
HUP	重新加载配置，更新时区（仅适用于 FreeBSD 和 Linux），使用更新的配置启动新的工作进程，优雅地 关闭旧的工作进程
USR1	重新打开日志文件
USR2	升级可执行文件
WINCH	优雅地 关闭工作进程

您可以使用 `kill` 发送信号：

```
$ sudo kill -QUIT $(cat /run/angie.pid)
```

单个工作进程也可以使用信号控制，尽管这是可选的。支持的信号有：

TERM, INT	快速关闭
QUIT	优雅的 关闭
USR1	重新打开日志文件
WINCH	用于调试的异常终止（需要启用*debug_points*）

更改配置

为了让 Angie 重新读取配置文件，应向主进程发送 `HUP` 信号。主进程首先检查语法的有效性，然后尝试应用新配置，包括打开新的日志文件和监听套接字。如果应用新配置失败，主进程将回滚更改并继续使用旧配置运行。如果应用成功，主进程将启动新的工作进程，并向旧的工作进程发送消息，要求它们优雅地 关闭。旧的工作进程关闭它们的监听套接字并继续为现有客户端提供服务。在所有客户端服务完成后，旧的工作进程将关闭。

Angie 跟踪每个进程的配置更改。生成号在服务器首次启动时从 1 开始。这些数字随着每次配置重新加载而递增，并在进程标题中可见：

```
$ sudo angie
$ ps aux | grep angie

   angie: master process v1.10.0 #1 [angie]
   angie: worker process #1
```

成功重新加载配置后（无论是否有实际更改），Angie 将为接收到新配置的进程增加生成号：

```
$ sudo kill -HUP $(cat /run/angie.pid)
$ ps aux | grep angie

   angie: master process v1.10.0 #2 [angie]
   angie: worker process #2
```

如果任何来自先前生成的工作进程继续运行，它们将立即可见：

```
$ ps aux | grep angie

   angie: worker process #1
   angie: worker process #2
```

备注

不要将配置生成号与" 进程编号" 混淆；Angie 出于实际目的不使用连续的进程编号。

轮换日志文件

要轮换日志文件，首先重命名文件。然后，向主进程发送 USR1 信号。主进程将重新打开当前所有打开的日志文件，并将它们分配给工作进程运行的非特权用户。成功重新打开文件后，主进程关闭所有打开的文件，并通知工作进程重新打开它们的日志文件。工作进程也将打开新文件并立即关闭旧文件。结果，旧文件几乎立即可用于后续处理，例如压缩。

即时可执行文件升级

要升级服务器可执行文件，首先用新文件替换旧的可执行文件。然后，向主进程发送 USR2 信号。主进程将其当前文件重命名为带有进程 ID 的新文件，带有 .oldbin 后缀，例如 /usr/local/angie/logs/angie.pid.oldbin，然后启动新的可执行文件，新的可执行文件将启动新的工作进程。

请注意，旧主进程不会关闭其监听套接字，并且可以管理以重新启动其工作进程（如果需要）。如果新可执行文件未按预期执行，您可以采取以下操作之一：

- 向旧主进程发送 HUP 信号。这将启动新的工作进程，而无需重新读取配置。然后，您可以通过向新主进程发送 QUIT 信号来优雅地 关闭所有新进程。
- 向新主进程发送 TERM 信号。它将向其工作进程发送消息，要求它们立即退出。如果有任何进程未退出，请发送 KILL 信号以强制退出。当新主进程退出时，旧主进程将自动启动新的工作进程。

如果新主进程退出，旧主进程将从文件名中移除带有进程 ID 的 .oldbin 后缀。

如果升级成功，向旧主进程发送 QUIT 信号，只有新的进程将保留。

在启用配置后，Angie 还会在升级前将某些共享内存区域（目前是*proxy_cache_path* 中的 keys_zone）刷写到磁盘，以便新启动的主进程可以以更高的性能恢复它们。如果由于区域大小变化、二进制版本不兼容或其他原因导致恢复失败，Angie 将记录一条警报 (failed to restore zone at address)，并且不会使用区域恢复机制。

命令行选项

`-?`, `-h`	显示命令行参数帮助，然后退出。
`--build-env`	显示构建环境的辅助信息，然后退出。
`-c` *file*	使用 *file* 作为配置文件，而不是默认文件。
`-e` *file*	使用 *file* 作为错误日志文件，而不是默认文件。特殊值 `stderr` 指定标准错误输出。
`-g` *directives*	应用额外的全局配置指令，例如：`angie -g "pid /var/run/angie.pid; worker_processes `sysctl -n hw.ncpu`;"`。
`-m`, `-M`	显示内置 (`-m`) 或内置和加载 (`-M`) 模块的列表，然后退出。
`-p` *prefix*	使用指定的 *prefix* 路径作为 `angie` 的路径（服务器文件所在的目录；默认是 `/usr/local/angie/`）。
`-q`	如果设置了 `-t` 或 `-T`，则仅显示错误消息；否则，没有效果。
`-s` *signal*	向主进程发送信号：`stop`、`quit`、`reopen`、`reload` 等。
`-t`	测试配置文件，然后退出。Angie 检查配置语法，递归包含配置中提到的文件。
`-T`	与 `-t` 相同，但在递归包含配置中提到的所有文件后，还将摘要配置输出到标准输出。
`-v`	显示 Angie 版本，然后退出。
`-V`	显示 Angie 版本、编译器版本、构建时间以及使用的 构建参数，然后退出。

3.1.3 连接、会话、请求、日志

连接处理机制

Angie 支持多种连接处理方法。特定方法的可用性取决于所使用的平台。在支持多种方法的平台上，Angie 通常会自动选择最有效的方法。然而，如果需要，可以使用 *use* 指令明确选择连接处理方法。

以下是可用的连接处理方法：

方法	描述
`select`	一种标准方法。在没有更高效方法的平台上，支持模块会自动构建。可以使用 `--with-select_module` 和 `--without-select_module` 构建选项来强制启用或禁用此模块的构建。
`poll`	一种标准方法。在没有更高效方法的平台上，支持模块会自动构建。可以使用 `--with-poll_module` 和 `--without-poll_module` 构建选项来强制启用或禁用此模块的构建。
`kqueue`	一种在 FreeBSD 4.1+、OpenBSD 2.9+、NetBSD 2.0 和 macOS 上可用的高效方法。
`epoll`	一种在 Linux 2.6+ 上可用的高效方法。
`/dev/poll`	一种在 Solaris 7 11/99+、HP/UX 11.22+（eventport）、IRIX 6.5.15+ 和 Tru64 UNIX 5.1A+ 上可用的高效方法。
`eventport`	事件端口方法在 Solaris 10+ 上可用。（由于已知问题，建议使用 `/dev/poll` 方法。）

HTTP 请求处理

HTTP 请求经过一系列阶段，每个阶段执行特定类型的处理。

`Post-read`	初始阶段。在此阶段调用*RealIP* 模块。
`Server-rewrite`	在此阶段处理 `server` 块中（但在 `location` 块之外）定义的*Rewrite* 模块指令。
`Find-config`	一个特殊阶段，根据请求 URI 选择*location*。
`Rewrite`	类似于 `Server-rewrite` 阶段，但应用于在上一个阶段选择的 location 块中定义的*rewrite* 规则。
`Post-rewrite`	一个特殊阶段，如果在 `Rewrite` 阶段中修改了 URI，则请求被重定向到新位置，如在 `Find-config` 阶段。
`Preaccess`	在此阶段，标准 Angie 模块如*Limit Req* 注册它们的处理程序。
`Access`	验证客户端请求授权的阶段，通常通过调用标准 Angie 模块如*Auth Basic*。
`Post-access`	一个特殊阶段，处理*satisfy any* 指令。
`Precontent`	在此阶段，标准模块指令如*try_files* 和*mirror* 注册它们的处理程序。
`Content`	通常在此阶段生成响应。多个标准 Angie 模块在此阶段注册它们的处理程序，包括*Index*。*proxy_pass*、*fastcgi_pass*、*uwsgi_pass*、*scgi_pass* 和*grpc_pass* 指令也在此处处理。 处理程序按顺序调用，直到其中一个生成输出。
`Log`	最终阶段，执行请求日志记录。目前，只有*Log* 模块在此阶段注册其处理程序用于访问日志。

TCP/UDP 会话处理

来自客户端的 TCP/UDP 会话经过一系列阶段，每个阶段执行特定类型的处理：

`Post-accept`	接受客户端连接后的初始阶段。在此阶段调用*RealIP* 模块。
`Pre-access`	检查访问权限的初步阶段。在此阶段调用*Set* 模块。
`Access`	在实际数据处理之前限制客户端访问的阶段。在此阶段调用*Access* 模块。
`SSL`	执行 TLS/SSL 终止的阶段。在此阶段调用*SSL* 模块。
`Preread`	将初始数据字节读入*preread buffer* 的阶段，以允许模块如*SSL Preread* 在处理前分析数据。
`Content`	一个必需阶段，数据在此实际处理，通常涉及*Return* 模块以向客户端发送响应。在此也处理*proxy_pass* 指令。
`Log`	记录客户端会话处理结果的最终阶段。在此阶段调用*Log* 模块。

处理请求

虚拟服务器选择

最初，连接在默认服务器的上下文中创建。然后可以在请求处理的以下阶段确定服务器名称，每个阶段都参与服务器配置的选择：

- 在 SSL 握手时，根据 SNI 提前确定。
- 在处理请求行之后。

- 在处理 `Host` 头字段之后。

如果在处理请求行或 `Host` 头字段之后未确定服务器名称，Angie 将使用空名称作为服务器名称。

在这些阶段中的每一个阶段，可能会应用不同的服务器配置。因此，某些指令应谨慎指定：

- 在*ssl_protocols* 指令的情况下，协议列表由 OpenSSL 库在服务器配置根据 SNI 请求的名称应用之前设置。因此，协议应仅为默认服务器指定。
- *client_header_buffer_size* 和*merge_slashes* 指令在读取请求行之前应用。因此，这些指令要么使用默认服务器配置，要么使用 SNI 选择的服务器配置。
- 在处理请求头字段时，*ignore_invalid_headers*、*large_client_header_buffers* 和*underscores_in_headers* 指令的情况下，服务器配置还取决于它是否根据请求行或 `Host` 头字段进行了更新。
- 错误响应使用当前处理请求的服务器中的*error_page* 指令处理。

基于名称的虚拟服务器

Angie 首先确定哪个服务器应处理请求。考虑一个简单的配置，其中所有三个虚拟服务器都监听 80 端口：

```
server {

    listen 80;
    server_name example.org www.example.org;
    # ...
}

server {

    listen 80;
    server_name example.net www.example.net;
    #  ...
}

server {

    listen 80;
    server_name example.com www.example.com;
    #  ...
}
```

在此配置中，Angie 仅根据 `Host` 头字段确定哪个服务器应处理请求。如果此头字段的值与任何服务器名称不匹配或请求不包含此头字段，Angie 将请求路由到此端口的默认服务器。在上述配置中，默认服务器是第一个——这是 Angie 的标准默认行为。还可以使用*listen* 指令中的 `default_server` 参数显式指定哪个服务器应为默认：

```
server {

    listen 80 default_server;
```

```
    server_name example.net www.example.net;
    #  ...
}
```

> **备注**
>
> 请注意，默认服务器是监听套接字的属性，而不是服务器名称的属性。

国际化名称

国际化域名（IDNs）应在*server_name* 指令中使用 ASCII（Punycode）表示：

```
server {

    listen 80;
    server_name xn--e1afmkfd.xn--80akhbyknj4f; #     .
    #    ...
}
```

防止带有未定义服务器名称的请求

如果不应允许没有 `Host` 头字段的请求，可以定义一个只需丢弃此类请求的服务器：

```
server {

    listen 80;
    server_name "";
    return 444;
}
```

在此配置中，服务器名称设置为空字符串，这与没有 `Host` 头字段的请求匹配。然后返回一个特殊的非标准代码 444，它关闭连接。

结合基于名称和基于 IP 的虚拟服务器

让我们看看一个更复杂的配置，其中一些虚拟服务器监听不同的地址：

```
server {

    listen 192.168.1.1:80;
    server_name example.org www.example.org;
    #  ...
}

server {

    listen 192.168.1.1:80;
```

```
    server_name example.net www.example.net;
    #  ...
}

server {

    listen 192.168.1.2:80;
    server_name example.com www.example.com;
    #  ...
}
```

在此配置中，Angie 首先测试请求的 IP 地址和端口与*server* 块的*listen* 指令进行匹配。然后测试请求的 `Host` 头字段与匹配 IP 地址和端口的*server* 块的*server_name* 条目进行匹配。如果未找到服务器名称，请求将由默认服务器处理。例如，收到的请求 `www.example.com` 在端口 192.168.1.1:80 上将由该端口的默认服务器处理——即第一个服务器——因为 `www.example.com` 未在此端口定义。

如前所述，默认服务器是监听端口的属性，可以为不同的端口定义不同的默认服务器：

```
server {

    listen 192.168.1.1:80;
    server_name example.org www.example.org;
    #  ...
}

server {

    listen 192.168.1.1:80 default_server;
    server_name example.net www.example.net;
    #  ...
}

server {

    listen 192.168.1.2:80 default_server;
    server_name example.com www.example.com;
    #  ...
}
```

选择位置

考虑一个简单的 PHP 网站配置：

```
server {

    listen 80;
    server_name example.org www.example.org;
    root /data/www;
```

```
    location / {

        index index.html index.php;
    }

    location ~* \.(gif|jpg|png)$ {

        expires 30d;
    }

    location ~ \.php$ {

        fastcgi_pass localhost:9000;
        fastcgi_param SCRIPT_FILENAME
        $document_root$fastcgi_script_name;
        include fastcgi_params;
    }
}
```

Angie 首先搜索由字面字符串给出的最具体的前缀 `location`，而不管它们的列出顺序。在上述配置中，唯一的前缀位置是 `location /`，它匹配任何请求，并将作为最后的手段使用。然后，Angie 按配置文件中出现的顺序检查由正则表达式定义的位置。第一个匹配的表达式停止搜索，Angie 将使用该 `location`。如果没有正则表达式匹配请求，Angie 将使用之前找到的最具体的前缀 `location`。

备注

所有类型的位置仅测试请求行的 URI 部分，不包括参数。这是因为可以以各种方式指定查询字符串中的参数，例如：

- `/index.php?user=john&page=1`
- `/index.php?page=1&user=john`

此外，查询字符串可能包含任意数量的参数：

- `/index.php?page=1&something+else&user=john`

现在让我们看看在上述配置中如何处理请求：

- 请求 `/logo.gif` 首先匹配前缀 `location /`，然后匹配正则表达式 `.(gif|jpg|png)$`。因此，它由后者位置处理。使用指令 `root /data/www`，请求映射到文件 `/data/www/logo.gif`，并将文件发送给客户端。
- 请求 `/index.php` 也首先匹配前缀 `location /`，然后匹配正则表达式 `.(php)$`。因此，它由后者位置处理，请求被传递到监听 `localhost:9000` 的 FastCGI 服务器。*fastcgi_param* 指令将 FastCGI 参数 `SCRIPT_FILENAME` 设置为 `/data/www/index.php`，FastCGI 服务器执行文件。变量*$document_root* 设置为 `root` 指令的值，变量*$fastcgi_script_name* 设置为请求 URI，即 `/index.php`。

- 请求 `/about.html` 仅匹配前缀 `location /`，因此在此位置处理。使用指令 `root /data/www`，请求映射到文件 `/data/www/about.html`，并将文件发送给客户端。

处理请求 `/` `更为复杂。它仅匹配前缀 `location /`，因此在此位置处理。*index* 指令然后根据其参数和 `root /data/www` 指令测试索引文件的存在。如果文件 `/data/www/index.html` 不存在但文件 `/data/www/index.php` 存在，该指令执行内部重定向到 `/index.php`，Angie 再次搜索位置，仿佛请求是由客户端发送的。如前所述，重定向的请求最终将由 FastCGI 服务器处理。

代理和负载均衡

设置 Angie 作为代理服务器是一个常见用途。在此角色中，Angie 接收请求，将它们转发到代理服务器，检索来自这些服务器的响应，并将响应发送回客户端。

一个简单的代理服务器：

```
server {

    location / {

        proxy_pass http://backend:8080;
    }
```

proxy_pass 指令指示 Angie 将客户端请求传递给后台 `backend:8080` 代理服务器）。还有许多额外的*directives* 可用于进一步配置代理连接。

FastCGI 代理

Angie 可以用于将请求路由到运行各种框架和编程语言（如 PHP）的 FastCGI 服务器。

与 FastCGI 服务器一起工作的最基本的 Angie 配置涉及使用*fastcgi_pass* 指令而不是*proxy_pass* 指令，并使用*fastcgi_param* 指令设置传递给 FastCGI 服务器的参数。假设 FastCGI 服务器可通过 `localhost:9000` 访问。在 PHP 中，`SCRIPT_FILENAME` 参数用于确定脚本名称，`QUERY_STRING` 参数用于传递请求参数。结果配置如下：

```
server {

    location / {

        fastcgi_pass localhost:9000;
        fastcgi_param SCRIPT_FILENAME $document_root$fastcgi_script_name;
        fastcgi_param QUERY_STRING $query_string;
    }

    location ~ \.(gif|jpg|png)$ {

        root /data/images;
    }
}
```

此配置设置了一个服务器，通过 FastCGI 协议将除静态图像之外的所有请求路由到在 `localhost:9000` 上运行的代理服务器。

WebSocket 代理

要从 HTTP/1.1 升级连接到 WebSocket，使用 HTTP/1.1 中可用的 协议切换 机制。

不过，这里有一个细微之处：由于 `Upgrade` 头是一个 逐跳头，它不会从客户端传递到代理服务器。对于正向代理，客户端可以使用 CONNECT 方法来规避此问题。此方法不适用于反向代理，因为客户端不知道任何代理服务器，并且需要在代理服务器上进行特殊处理。

Angie 实现了一种特殊的操作模式，允许在客户端和代理服务器之间建立隧道，如果代理服务器返回代码 101（协议切换）的响应，并且客户端通过请求中的 `Upgrade` 头请求协议切换。

如前所述，逐跳头，包括 `Upgrade` 和 `Connection`，不会从客户端传递到代理服务器。因此，为了使代理服务器了解客户端想要切换到 WebSocket 协议，这些头必须显式传递：

```
location /chat/ {

    proxy_pass http://backend;
    proxy_http_version 1.1;
    proxy_set_header Upgrade $http_upgrade;
    proxy_set_header Connection "upgrade";
}
```

更复杂的示例演示了在向代理服务器的请求中 `Connection` 头字段的值如何依赖于客户端请求头中的 `Upgrade` 字段的存在：

```
http {

    map $http_upgrade $connection_upgrade {

        default upgrade;
        '' close;
    }

    server {

        ...

        location /chat/ {

            proxy_pass http://backend;
            proxy_http_version 1.1;
            proxy_set_header Upgrade $http_upgrade;
            proxy_set_header Connection $connection_upgrade;
        }
    }
}
```

默认情况下，如果代理服务器在 60 秒内没有传输任何数据，连接将关闭。可以使用*proxy_read_timeout* 指令增加此超时。或者，可以配置代理服务器定期发送 WebSocket ping 帧以重置超时并检查连接是否仍然活动。

负载均衡

跨多个应用实例进行负载均衡是一种广泛使用的技术，以优化资源利用，最大化吞吐量，减少延迟，并确保容错配置。

Angie 可以用作高效的 HTTP 负载均衡器，将流量分配到多个应用服务器，从而提高 Web 应用程序的性能、可扩展性和可靠性。

使用 Angie 进行负载均衡的最简单配置可能如下所示：

```
http {

    upstream myapp1 {

        server srv1.example.com;
        server srv2.example.com;
        server srv3.example.com;
    }

    server {

        listen 80;

        location / {

            proxy_pass http://myapp1;
        }
    }
}
```

在上述示例中，三个相同应用的实例在 `srv1` 到 `srv3` 上运行。当负载均衡方法未明确配置时，默认为轮询。其他支持的负载均衡机制包括：*weight*、*least_conn* 和*ip_hash*。Angie 中的反向代理实现还支持带内（或被动）服务器健康检查。这些通过在*upstream* 上下文中的*server* 块内使用*max_fails* 和*fail_timeout* 指令进行配置。

日志记录

> **备注**
>
> 除了这里列出的选项外，还可以启用*debugging log*。

Syslog

error_log 和*access_log* 指令支持记录到 `syslog`。以下参数用于配置记录到 `syslog`：

server=*address*	指定 syslog 服务器的地址。地址可以是域名或 IP 地址，带有可选端口，或者是在 "unix:" 前缀后指定的 UNIX 域套接字路径。如果未指定端口，则使用 UDP 端口 514。如果一个域名解析为多个 IP 地址，则使用第一个解析的地址。
facility=*string*	设置 syslog 消息的 facility，如 RFC 3164 中定义的。可能的 facilities 包括："kern"、"user"、"mail"、"daemon"、"auth"、"intern"、"lpr"、"news"、"uucp"、"clock"、"authpriv"、"ftp"、"ntp"、"audit"、"alert"、"cron"、"local0"..."local7"。默认值是 "local7"。
severity=*string*	定义*access_log* 的 syslog 消息的严重级别，如 RFC 3164 中指定的。可能的值与*error_log* 指令的第二个参数（级别）相同。默认值是 "info"。错误消息的严重性由 Angie 确定，因此在*error_log* 指令中忽略此参数。
tag=*string*	设置 syslog 消息的标签。默认标签是 "angie"。
nohostname	禁用在 syslog 消息头中添加 hostname 字段。

示例 syslog 配置：

```
error_log syslog:server=192.168.1.1 debug;

access_log syslog:server=unix:/var/log/angie.sock,nohostname;
access_log syslog:server=[2001:db8::1]:12345,facility=local7,tag=angie,severity=info combined;
```

备注

为防止泛滥，syslog 条目每秒报告不超过一次。

3.2 参考和索引

这些摘要部分提供了内置模块的参考信息，它们的配置示例，以及支持的指令和变量。

3.2.1 内置模块

本指南描述了 Angie 的内置模块，提供了配置示例，列出了它们的指令和参数，以及内置变量。

核心模块

该模块提供服务器基本运行所需的核心功能和配置指令，负责处理关键任务，如管理工作进程、配置事件驱动模型以及处理传入连接和请求。它包含了设置主进程、错误日志记录和控制服务器底层行为的关键指令。

配置示例

```
user www www;
worker_processes 2;

error_log /var/log/error.log info;
```

```
events {

    use kqueue; worker_connections 2048;
}
```

指令

accept_mutex

语法	accept_mutex on | off;
默认值	accept_mutex off;
上下文	events

当启用 accept_mutex 时，工作进程将轮流接受新连接。如果不启用此设置，所有工作进程都会收到新连接的通知，在新连接数量较低的情况下可能导致系统资源使用效率低下。

> **备注**
>
> 在支持 EPOLLEXCLUSIVE 标志的系统上或使用*reuseport* 指令时，无需启用 accept_mutex。

accept_mutex_delay

语法	accept_mutex_delay *time*;
默认值	accept_mutex_delay 500ms;
上下文	events

如果启用了*accept_mutex*，此指令指定工作进程在另一个工作进程已经处理新连接的情况下，等待继续接受新连接的最长时间。

daemon

语法	daemon on | off;
默认值	daemon on;
上下文	main

确定 Angie 是否应该作为守护进程运行。这主要在开发过程中使用。

debug_connection

语法	`debug_connection` *address* | *CIDR* | `unix:`;
默认值	—
上下文	events

为特定客户端连接启用调试日志。其他连接将使用*error_log* 指令设置的日志级别。可以通过 IPv4 或 IPv6 地址、网络或主机名指定连接。对于使用 UNIX 域套接字的连接，使用 `unix:` 参数启用调试日志。

```
events {

    debug_connection 127.0.0.1;
    debug_connection localhost;
    debug_connection 192.0.2.0/24;
    debug_connection ::1;
    debug_connection 2001:0db8::/32;
    debug_connection unix:;
    #  ...
}
```

重要

要使此指令生效，Angie 必须在构建时启用调试日志。

debug_points

语法	`debug_points abort | stop;`
默认值	—
上下文	main

此指令用于调试。

当发生内部错误时，例如工作进程重启期间的套接字泄漏，启用 `debug_points` 将创建核心文件 (`abort`) 或停止进程 (`stop`) 以便使用系统调试器进行进一步分析。

env

语法	`env` *variable*[=*value*];
默认值	`env TZ;`
上下文	main

默认情况下，Angie 会删除从其父进程继承的所有环境变量，除了 `TZ` 变量。此指令允许您保留某些继承的变量、修改它们的值或创建新的环境变量。

这些变量随后会：

- 在可执行文件的实时升级 期间被继承
- 被*Perl* 模块使用
- 对工作进程可用

请注意，以这种方式控制系统库可能并不总是有效，因为库通常只在初始化期间检查变量，而初始化发生在此指令生效之前。除非另有明确配置，否则 `TZ` 变量始终会被继承并可供*Perl* 模块访问。

示例：

```
env MALLOC_OPTIONS;
env PERL5LIB=/data/site/modules;
env OPENSSL_ALLOW_PROXY_CERTS=1;
```

备注

`ANGIE` 环境变量由 Angie 内部使用，用户不应直接设置。

error_log

语法	`error_log` *file* [*level*];
默认值	`error_log logs/error.log error;` (路径取决于 `--error-log-path` 构建选项)
上下文	main, http, mail, stream, server, location

配置日志记录，允许在同一配置级别指定多个日志。如果在 `main` 配置级别没有明确定义日志文件，将使用默认文件。

第一个参数指定存储日志的文件。特殊值 `stderr` 选择标准错误流。要配置日志记录到*syslog*，使用 `"syslog:"` 前缀。要记录到循环内存缓冲区，使用 `"memory:"` 前缀后跟缓冲区大小；这通常用于调试。

第二个参数设置日志级别，可以是以下之一：`debug`、`info`、`notice`、`warn`、`error`、`crit`、`alert` 或 `emerg`。这些级别按严重性递增的顺序列出。设置日志级别将捕获等于和高于该级别的消息。

设置	捕获的级别
debug	debug、info、notice、warn、error、crit、alert、emerg
info	info、notice、warn、error、crit、alert、emerg
notice	notice、warn、error、crit、alert、emerg
warn	warn、error、crit、alert、emerg
error	error、crit、alert、emerg
crit	crit、alert、emerg
alert	alert、emerg
emerg	emerg

如果省略此参数，则使用 `error` 作为默认日志级别。

> **重要**
>
> 为了使 `debug` 日志级别有效，Angie 必须以启用*debugging log* 的方式构建。

events

语法	`events { ... };`
默认	—
上下文	main

提供影响连接处理的指令的配置文件上下文。

include

语法	`include` 文件 | 掩码;
默认	—
上下文	任何

将另一个文件或与指定 掩码匹配的文件包含到配置中。包含的文件必须包含语法正确的指令和块。

示例：

```
include mime.types;
include vhosts/*.conf;
```

load_module

语法	`load_module` 文件;
默认	—
上下文	main

从指定文件加载动态模块。如果提供了相对路径，则根据 `--prefix` 构建选项进行解释。要验证路径：

```
$ sudo angie -V
```

示例：

```
load_module modules/ngx_mail_module.so;
```

lock_file

语法	`lock_file` 文件;
默认	`lock_file logs/angie.lock;` (路径取决于 `--lock-path` 构建选项)
上下文	main

Angie 使用锁定机制来实现*accept_mutex* 并序列化对共享内存的访问。在大多数系统中，锁是使用原子操作进行管理的，因此此指令并非必需。然而，在某些系统上，使用替代的 *lock file* 机制。此指令设置锁文件名称的前缀。

master_process

语法	`master_process on | off;`
默认	`master_process on;`
上下文	main

确定是否启动工作进程。此指令供 Angie 开发者使用。

multi_accept

语法	`multi_accept on | off;`
默认	`multi_accept off;`
上下文	events

`on`	工作进程将同时接受所有新连接。
`off`	工作进程将一次接受一个新连接。

备注

如果使用*kqueue* 连接处理方法，则此指令被忽略，因为它提供了准备接受的新连接的数量。

pcre_jit

语法	`pcre_jit on | off;`
默认	`pcre_jit off;`
上下文	main

启用或禁用对配置解析时已知的正则表达式进行”即时编译”（PCRE JIT）。

PCRE JIT 可以显著加速正则表达式处理。

重要

JIT 在 PCRE 库 8.20 版本及以上可用，前提是它们是使用 `--enable-jit` 配置选项构建的。当 Angie 使用 PCRE 库构建时（`--with-pcre=`），JIT 支持通过 `--with-pcre-jit` 选项启用。

pid

语法	`pid` 文件 | `off`;
默认	`pid logs/angie.pid`; (路径取决于 `--pid-path` 构建选项)
上下文	main

指定将存储 Angie 主进程 ID 的 文件。该文件是原子创建的，这确保其内容始终正确。`off` 设置禁用此文件的创建。

备注

如果在重新配置期间修改了 文件设置，但指向先前 PID 文件的符号链接，则不会重新创建该文件。

ssl_engine

语法	`ssl_engine` 设备;
默认	—
上下文	main

指定硬件 SSL 加速器的名称。

ssl_object_cache_inheritable

语法	`ssl_object_cache_inheritable on` | `off`;
默认	`ssl_object_cache_inheritable on`;
上下文	main

如果启用，SSL 对象（SSL 证书、密钥、受信任的 CA 证书、CRL 列表）将在配置重新加载时继承。

如果从上次配置加载以来它们的修改时间和文件索引没有变化，则从文件加载的 SSL 对象将被继承。指定为 `engine:name:id` 的密钥永远不会被继承，而指定为 `data:value` 的密钥始终会被继承。

从变量加载的 SSL 对象不能被继承。

示例：

```
ssl_object_cache_inheritable on;

http {
    server {
        ssl_certificate     example.com.crt;
        ssl_certificate_key example.com.key;
    }
}
```

thread_pool

语法	thread_pool 名称 threads= 数量 [max_queue= 数量];
默认	thread_pool default threads=32 max_queue=65536;
上下文	main

定义用于多线程文件读取和发送的线程池的 名称和参数，无阻塞 工作进程。

threads 参数指定池中的线程数量。

如果所有线程都忙，新任务将会在队列中等待。max_queue 参数限制可以在队列中等待的任务数量。默认情况下，队列最多可以容纳 65536 个任务。当队列溢出时，任务将以错误方式完成。

timer_resolution

语法	timer_resolution 间隔;
默认	—
上下文	main

在工作进程中减少计时器分辨率，从而减少 gettimeofday() 系统调用的次数。默认情况下，每当接收到内核事件时，都会调用 gettimeofday()。通过降低分辨率，它仅在指定的间隔内调用一次。

示例：

```
timer_resolution 100ms;
```

间隔的内部实现取决于所使用的方法：

- 如果使用*kqueue*，则使用 EVFILT_TIMER 过滤器；
- 如果使用*eventport*，则使用 timer_create() 函数；
- 否则使用 setitimer() 函数。

use

语法	use 方法;
默认	—
上下文	events

指定要使用的连接处理 方法。通常不需要明确指定，因为 Angie 默认使用最有效的方法。

user

语法	user 用户 [组];
默认	user <build parameter --user> <build parameter --group>;
上下文	main

定义工作进程使用的 用户和 组凭据（另请参见 构建参数）。如果省略 组，则使用与 用户同名的组。

worker_aio_requests

语法	worker_aio_requests 数量;
默认	worker_aio_requests 32;
上下文	events

在使用*aio* 与*epoll* 连接处理方法时，设置单个工作进程的最大异步 I/O 操作 数量。

worker_connections

语法	worker_connections 数量;
默认	worker_connections 512;
上下文	events

设置工作进程可以打开的最大并发连接数。

需要注意的是，这个数字包括所有连接（例如与代理服务器的连接等），而不仅仅是与客户端的连接。另外，实际的并发连接数不能超过当前对最大打开文件数的限制，该限制可以通过*worker_rlimit_nofile* 更改。

worker_cpu_affinity

语法	worker_cpu_affinity *cpumask* ...; worker_cpu_affinity auto [*cpumask*];
默认	—
上下文	main

将工作进程绑定到 CPU 集合。每个 CPU 集合由允许的 CPU 的位掩码表示。应为每个工作进程定义一个单独的集合。默认情况下，工作进程不绑定到任何特定的 CPU。

例如：

```
worker_processes    4;
worker_cpu_affinity 0001 0010 0100 1000;
```

此配置将每个工作进程绑定到一个单独的 CPU。

或者：

```
worker_processes    2;
worker_cpu_affinity 0101 1010;
```

这将第一个工作进程绑定到 CPU0 和 CPU2，第二个工作进程绑定到 CPU1 和 CPU3。这种设置适合超线程。

特殊值 `auto` 允许自动将工作进程绑定到可用的 CPU：

```
worker_processes auto;
worker_cpu_affinity auto;
```

可选的掩码参数可用于限制自动绑定可用的 CPU：

```
worker_cpu_affinity auto 01010101;
```

重要

此指令仅在 FreeBSD 和 Linux 上可用。

worker_priority

语法	`worker_priority` 数值;
默认	`worker_priority 0;`
上下文	main

定义工作进程的调度优先级，类似于 `nice` 命令：负 数值表示更高的优先级。允许的范围通常从-20 到 20。

示例：

```
worker_priority -10;
```

worker_processes

语法	worker_processes 数量 | auto;
默认	worker_processes 1;
上下文	main

定义工作进程的数量。

最佳值取决于多种因素，包括（但不限于）CPU 核心数量、存储数据的硬盘驱动器数量和负载模式。如果不确定，将其设置为可用 CPU 核心的数量是一个很好的起点（值"auto" 将尝试自动检测）。

worker_rlimit_core

语法	worker_rlimit_core 大小;
默认	—
上下文	main

更改工作进程的核心文件最大大小限制（RLIMIT_CORE）。用于在不重启主进程的情况下增加限制。

worker_rlimit_nofile

语法	worker_rlimit_nofile 数量;
默认	—
上下文	main

更改工作进程的最大打开文件数限制（RLIMIT_NOFILE）。用于在不重启主进程的情况下增加限制。

worker_shutdown_timeout

语法	worker_shutdown_timeout 时间;
默认	—
上下文	main

配置工作进程优雅关闭的超时。一旦 时间到期，Angie 将尝试关闭所有当前打开的连接以便完成关闭过程。

优雅关闭是通过向主进程发送*QUIT 信号* 来启动的，这会指示工作进程停止接受新连接，同时允许现有连接完成。工作进程会继续处理活动请求直到完成，然后优雅地关闭。如果连接在超过 worker_shutdown_timeout 的时间后仍然保持打开状态，Angie 将强制关闭这些连接以完成关闭。此外，客户端保持活动的连接仅在空闲至少*lingering_timeout* 指定的时间后才会关闭。

working_directory

语法	`working_directory` 目录;
默认	—
上下文	main

定义工作进程的当前工作目录。这主要用于写入核心文件，因此工作进程必须对指定目录具有写入权限。

HTTP 模块

访问

该模块根据客户端 IP 地址或网络控制对服务器资源的访问。它允许允许或阻止特定的 IP、IP 范围或 UNIX 域套接字，以通过限制对网站或应用程序敏感区域的访问来增强安全性。

访问还可以通过使用*Auth Basic* 模块的密码或基于*Auth Request* 模块的子请求结果进行限制。要同时应用地址和密码限制，请使用*satisfy* 指令。

配置示例

```
location / {

    deny 192.168.1.1;
    allow 192.168.1.0/24;
    allow 10.1.1.0/16;
    allow 2001:0db8::/32;
    deny all;
}
```

规则按顺序评估，直到找到匹配项。在此示例中，仅允许访问 IPv4 网络 `10.1.1.0/16` 和 `192.168.1.0/24`，排除特定地址 `192.168.1.1`，以及 IPv6 网络 `2001:0db8::/32`。当规则较多时，最好使用*Geo* 模块中的变量。

指令

allow

语法	`allow` *address* | *CIDR* | `unix:` | `all`;
默认	—
上下文	http, server, location, limit_except

允许对指定网络或地址的访问。特殊值 `all` 表示所有客户端 IP。

Added in version 1.5.1: 特殊值 `unix:` 允许对任何 UNIX 域套接字的访问。

deny

语法	`deny` *address* | *CIDR* | `unix:` | `all`;
默认	—
上下文	http, server, location, limit_except

拒绝对指定网络或地址的访问。特殊值 `all` 表示所有客户端 IP。

Added in version 1.5.1: 特殊值 `unix:` 拒绝对任何 UNIX 域套接字的访问。

ACME

提供使用 ACME 协议 自动获取证书的功能。

在 从源代码构建时，默认不会构建此模块；需要使用 构建选项 `--with-http_acme_module` 启用它。在来自 我们的软件仓库的包和镜像中，该模块已包含在构建中。

配置示例

有关配置示例和设置说明，请参阅*ACME 配置* 部分。

指令

acme

语法	`acme` 名称;
默认值	—
上下文	server

对于所有引用名为 名称的*ACME 客户端* 的*server* 块中的*server_name* 指令指定的所有域名，将获得单个证书；如果 `server_name` 配置更改，证书将更新以反映这些更改。

每次 Angie 启动时，都会为所有缺少有效证书的域名请求新证书。可能的原因包括证书已过期、文件丢失或不可读、证书配置发生更改等。

备注

目前，不支持使用正则表达式指定的域名，这些域名将被忽略。

通配符域名仅在 `acme_client` 中设置了 `challenge=dns` 时才受支持。

此指令可以多次指定以加载不同类型的证书，例如 RSA 和 ECDSA：

```
server {

    listen 443 ssl;
```

```
    server_name example.com www.example.com;

    ssl_certificate $acme_cert_rsa;
    ssl_certificate_key $acme_cert_key_rsa;

    ssl_certificate $acme_cert_ecdsa;
    ssl_certificate_key $acme_cert_key_ecdsa;

    acme rsa;
    acme ecdsa;
}
```

acme_client

语法	acme_client 名称 *uri* [enabled=on | off] [key_type= 类型] [key_bits= 数字] [email=*email*] [max_cert_size= 数字] [renew_before_expiry= 时间] [renew_on_load] [retry_after_error=off| 时间] [challenge=dns | http] [account_key= 文件];
默认值	—
上下文	http

定义一个全局唯一的名为 名称的 ACME 客户端。该名称必须对目录有效，是一个包含变量的字符串，并且将被视为不区分大小写。

小技巧

此处指定的客户端名称在 Angie 配置中标识该客户端，允许将 acme_client、*acme* 指令和使用此名称的模块变量 相互关联；不要将其与您的域名或服务器名称混淆。

第二个必需参数是 ACME 目录的 *uri*。例如，Let's Encrypt ACME 目录的 URI 列出 为 https://acme-v02.api.letsencrypt.org/directory。

备注

ACME 模块会在*client* 上下文中添加一个名为 location @acme 的位置，可用于配置对 ACME 目录的请求；默认情况下，此 location 包含一个带有目录 *uri* 的*proxy_pass* 指令，可以向其添加来自*Proxy* 模块的其他设置。

为了使此指令生效，必须在相同上下文中配置*resolver*。

备注

出于测试目的，证书颁发机构通常提供单独的测试环境。例如，Let's Encrypt 的测试环境 为 https://acme-staging-v02.api.letsencrypt.org/directory。

`enabled`	启用或禁用客户端证书的更新；例如，在不从配置中删除客户端的情况下临时暂停时，这非常有用。 默认值：`on`。
`key_type`	证书的私钥算法类型。有效值：`rsa`，`ecdsa`。 默认值：`ecdsa`。
`key_bits`	证书密钥的位数。默认值：`ecdsa` 为 256，`rsa` 为 2048。
`email`	用于反馈的可选电子邮件地址；在 CA 服务器上创建账户时使用。
`max_cert_size`	指定新证书文件的最大允许大小（以字节为单位），以在共享内存中为新证书保留空间；请求的域名越多，需要的空间越大。 如果启动时已存在证书但其大小超过 `max_cert_size` 值，则 `max_cert_size` 值会动态增加以匹配现有证书文件的大小。 如果更新的证书大小超过 `max_cert_size`，则更新过程将失败并报错。 默认值：`8192`。
`renew_before_exp`	证书到期前的时间，在此时间点应开始证书更新。 默认值：`30d`。
`renew_on_load`	指定在每次加载配置时强制更新证书。
`retry_after_erro`	在无法获取证书时重试的时间。如果设置为 `off`，客户端不会在失败后重试获取证书。 默认值：`2h`。
`challenge`	设置 ACME 客户端的验证类型。有效值：`dns`，`http`。 默认值：`http`。
`account_key`	指定包含 PEM 格式密钥的文件的完整路径。如果您想使用现有账户密钥而不是自动生成一个，或者需要为多个 ACME 客户端使用同一个密钥，这很有用。 支持的密钥类型： • RSA 密钥，长度为 8 的倍数，范围从 2048 到 8192 位。 • ECDSA 密钥，长度为 256、384 或 521 位。 指定 `account_key` 参数时，应确保密钥文件确实存在。如果文件不存在，Angie 将尝试在指定路径创建它。 请注意，ACME 客户端的密钥是按照它们在配置中通过*acme_client*、*acme* 或*acme_hook* 指令提及的顺序创建的。因此，如果一个客户端应该使用为另一个客户端创建的密钥，那么另一个客户端应该在配置中更早出现。 此外，密钥仅为设置了 `enabled=on` 的客户端创建。

acme_client_path

语法	`acme_client_path` 路径;
默认值	—
上下文	http

覆盖用于存储证书和密钥的目录的 路径，该路径在构建时由 构建选项 `--http-acme-client-path` 指定。

acme_dns_port

语法	`acme_dns_port` 端口 | *ip*[*:* 端口] | *[ip6]*[*:* 端口];
默认值	`acme_dns_port 53;`
上下文	http

指定模块通过 UDP 处理来自 ACME 服务器的 DNS 查询所使用的端口。端口号必须在 1 到 65535 之间。

也支持同时指定 IP 地址和可选端口。可以使用 `ip:端口`格式的 IPv4 地址和 `[ip6]:端口`格式的 IPv6 地址：

```
acme_dns_port 8053;
acme_dns_port 127.0.0.1;
acme_dns_port [::1];
```

要使用 1024 或更低的端口号，Angie 必须以超级用户 权限运行。

acme_hook

语法	`acme_hook` 名称 [*uri*];
默认值	—
上下文	location

该指令将服务器与指定的 ACME 客户端关联起来。通过定义该指令的 `location` 上下文，调用由外部服务实现的处理程序（hook）。

名称	指定关联的 ACME 客户端。
uri	包含变量的字符串；定义调用处理程序的请求字符串。 默认值为 `/`。

例如，以下配置将*hook 变量* 的值通过请求字符串传递给 FastCGI 应用程序：

```
acme_hook example uri=/acme_hook/$acme_hook_name?domain=$acme_hook_domain&key=$acme_hook_
↪keyauth;
fastcgi_param REQUEST_URI $request_uri;
fastcgi_pass ...;
```

内置变量

`$acme_cert_<` 名称>

由此 名称的客户端获得的最后一个证书文件（如果有）的内容。

`$acme_cert_key_< 名称>`

此 名称的客户端使用的证书密钥文件的内容。

重要

证书文件仅在 ACME 客户端获得至少一个证书后可用，而密钥文件在启动后立即可用。

`$acme_hook_challenge`

验证类型。可能的值：`dns`，`http`。

`$acme_hook_client`

发起请求的 ACME 客户端的名称。

`$acme_hook_domain`

被验证的域名。如果是通配符域名，将不带 `*.` 前缀传递。

`$acme_hook_keyauth`

授权字符串：

- 在 DNS 验证中，用作 TXT 记录的值，记录名称格式为 `_acme-challenge.` + `$acme_hook_domain` + `.`。
- 在 HTTP 验证中，此字符串应用作 ACME 服务器请求的响应内容。

`$acme_hook_name`

钩子名称。对于不同类型的验证，它可能有不同的值和含义：

值	DNS 验证中的含义	HTTP 验证中的含义
`add` (添加钩子)	需要在 DNS 配置中添加相应的 TXT 记录。	需要准备对相应 HTTP 请求的响应。
`remove` (移除钩子)	可以从 DNS 配置中删除 TXT 记录。	此 HTTP 请求不再相关；可以删除先前创建的包含授权字符串的文件。

`$acme_hook_token`

验证令牌。在 HTTP 验证中，用作请求的文件名：`/.well-known/acme-challenge/` + `$acme_hook_token`。

Addition

该模块是一个过滤器，用于在响应之前和之后添加文本。

当 从源代码构建时，该模块默认不被构建；它需要通过 `--with-http_addition_module` 构建选项来启用。

在来自 我们的仓库的包和镜像中，该模块已包含在构建中。

配置示例

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

指令

add_before_body

语法	`add_before_body` *uri*;
默认值	—
上下文	http, server, location

在响应体之前添加处理给定子请求返回的文本。参数为空字符串 (`""`) 则取消继承自先前配置级别的添加。

add_after_body

语法	`add_after_body` *uri*;
默认值	—
上下文	http, server, location

在响应体之后添加处理给定子请求返回的文本。参数为空字符串 (`""`) 则取消继承自先前配置级别的添加。

addition_types

语法	`addition_types` mime-*type* ...;
默认值	`addition_types text/html;`
上下文	http, server, location

允许在具有指定 MIME 类型的响应中添加文本，除了”text/html” 之外。特殊值”*” 匹配任何 MIME 类型。

API

该模块提供了 HTTP RESTful 接口，以 JSON 格式访问有关 Web 服务器实例的基本信息，以及客户端连接、共享内存区域、DNS 查询、HTTP 请求、HTTP 响应缓存、*stream* 模块的会话，和*limit_conn http* 、*limit_conn stream* 、*limit_req* 和*http upstream* 模块的区域的:ref: 统计信息 *<metrics>*。

接口接受 `GET` 和 `HEAD` HTTP 方法；使用其他方法的请求将导致错误：

```
{
    "error": "MethodNotAllowed",
    "description": "请求的API元素\"/\"不允许使用POST方法。"
}
```

在 Angie PRO 中，此接口包含一个:ref: 动态配置 *<api_config>* 部分，允许在不重新加载配置或重启的情况下更改设置；目前，可以配置:ref:*u_upstream* 中的单个服务器。

Directives

api

语法	`api` *path*;
默认	—
上下文	location

在 `location` 中启用 HTTP RESTful 接口。

path 参数是必需的。类似于*alias* 指令，它设置用于替换 `location` 中指定路径的路径，但作用于 API 树而非文件系统。

如果在前缀 `location` 中指定：

```
location /stats/ {
    api /status/http/server_zones/;
}
```

与前缀 */stats/* 匹配的请求 URI 部分将被 *path* 参数中指定的路径替换：*/status/http/server_zones/*。例如，对 */stats/foo/* 的请求将访问 API 元素 `/status/http/server_zones/foo/`。

允许使用变量：*api /status/$module/server_zones/$name/* 以及在正则表达式 location 中使用：

```
location ~^/api/([^/]+)/(.*)$ {
    api /status/http/$1_zones/$2;
}
```

这里 *path* 参数定义了 API 元素的完整路径；因此，从对 `/api/location/data/` 的请求中将提取以下变量：

```
$1 = "location"
$2 = "data/"
```

最终请求将是 /status/http/location_zones/data/。

备注

在 Angie PRO 中，您可以将动态配置 *API* 与反映当前状态的不可变状态 *API* 分开：

```
location /config/ {
    api /config/;
}

location /status/ {
    api /status/;
}
```

path 参数还允许控制 API 访问：

```
location /status/ {
    api /status/;

    allow 127.0.0.1;
    deny  all;
}
```

或者：

```
location /blog/requests/ {
    api /status/http/server_zones/blog/requests/;

    auth_basic           "blog";
    auth_basic_user_file conf/htpasswd;
}
```

api_config_files

语法	api_config_files on | off;
默认	off
上下文	location

启用或禁用向*/status/angie/* API 部分添加 config_files 对象，该对象列出服务器实例当前加载的所有 Angie 配置文件的内容。例如，使用此配置：

```
location /status/ {
    api /status/;
    api_config_files on;
}
```

对 /status/angie/ 的请求大约返回以下内容：

```
{
    "version":"1.10.0",
    "address":"192.168.16.5",
    "generation":1,
    "load_time":"2025-07-03T12:58:39.789Z",
    "config_files": {
        "/etc/angie/angie.conf": "...",
        "/etc/angie/mime.types": "..."
    }
}
```

默认情况下，输出是禁用的，因为配置文件可能包含特别敏感的机密信息。

指标

Angie 在 API 的 /status/ 部分公开使用统计信息；您可以通过设置相应的 location 来开放访问权限。完全访问：

```
location /status/ {
    api /status/;
}
```

部分访问示例，如上所示：

```
location /stats/ {
    api /status/http/server_zones/;
}
```

示例配置

包含 location /status/、resolver、http 中的 upstream、http server、location、cache、http 中的 limit_conn 和 limit_req 区域的配置：

```
http {

    resolver 127.0.0.53 status_zone=resolver_zone;
    proxy_cache_path /var/cache/angie/cache keys_zone=cache_zone:2m;
    limit_conn_zone $binary_remote_addr zone=limit_conn_zone:10m;
    limit_req_zone $binary_remote_addr zone=limit_req_zone:10m rate=1r/s;

    upstream upstream {
```

```
        zone upstream 256k;
        server backend.example.com service=_example._tcp resolve max_conns=5;
       keepalive 4;
    }

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

        status_zone http_server_zone;
        proxy_cache cache_zone;

        access_log /var/log/access.log main;

        location / {
            root /usr/share/angie/html;
            status_zone location_zone;
            limit_conn limit_conn_zone 1;
            limit_req zone=limit_req_zone burst=5;
        }
        location /status/ {
            api /status/;

            allow 127.0.0.1;
            deny all;
        }

    }
}
```

响应请求 `curl https://www.example.com/status/` 时，Angie 返回：

JSON 树

```
{
    "angie": {
        "version":"1.10.0",
        "address":"192.168.16.5",
        "generation":1,
        "load_time":"2025-07-03T12:58:39.789Z"
    },

    "connections": {
        "accepted":2257,
        "dropped":0,
        "active":3,
        "idle":1
    },
```

```
"slabs": {
    "cache_zone": {
        "pages": {
            "used":2,
            "free":506
        },

        "slots": {
            "64": {
                "used":1,
                "free":63,
                "reqs":1,
                "fails":0
            },

            "512": {
                "used":1,
                "free":7,
                "reqs":1,
                "fails":0
            }
        }
    },

    "limit_conn_zone": {
        "pages": {
            "used":2,
            "free":2542
        },

        "slots": {
            "64": {
                "used":1,
                "free":63,
                "reqs":74,
                "fails":0
            },

            "128": {
                "used":1,
                "free":31,
                "reqs":1,
                "fails":0
            }
        }
    },
```

```
        "limit_req_zone": {
            "pages": {
                "used":2,
                "free":2542
            },

            "slots": {
                "64": {
                    "used":1,
                    "free":63,
                    "reqs":1,
                    "fails":0
                },

                "128": {
                    "used":2,
                    "free":30,
                    "reqs":3,
                    "fails":0
                }
            }
        }
    },

    "http": {
        "server_zones": {
            "http_server_zone": {
                "ssl": {
                    "handshaked":4174,
                    "reuses":0,
                    "timedout":0,
                    "failed":0
                },

                "requests": {
                    "total":4327,
                    "processing":0,
                    "discarded":8
                },

                "responses": {
                    "200":4305,
                    "302":12,
                    "404":4
                },

                "data": {
                    "received":733955,
```

```
                "sent":59207757
            }
        }
    },

    "location_zones": {
        "location_zone": {
            "requests": {
                "total":4158,
                "discarded":0
            },

            "responses": {
                "200":4157,
                "304":1
            },

            "data": {
                "received":538200,
                "sent":177606236
            }
        }
    },
    "caches": {
        "cache_zone": {
            "size":0,
            "cold":false,
            "hit": {
                "responses":0,
                "bytes":0
            },

            "stale": {
                "responses":0,
                "bytes":0
            },

            "updating": {
                "responses":0,
                "bytes":0
            },

            "revalidated": {
                "responses":0,
                "bytes":0
            },

            "miss": {
```

```
                "responses":0,
                "bytes":0,
                "responses_written":0,
                "bytes_written":0
            },

            "expired": {
                "responses":0,
                "bytes":0,
                "responses_written":0,
                "bytes_written":0
            },

            "bypass": {
                "responses":0,
                "bytes":0,
                "responses_written":0,
                "bytes_written":0
            }
        }
    },

    "limit_conns": {
        "limit_conn_zone": {
            "passed":73,
            "skipped":0,
            "rejected":0,
            "exhausted":0
        }
    },

    "limit_reqs": {
        "limit_req_zone": {
            "passed":54816,
            "skipped":0,
            "delayed":65,
            "rejected":26,
            "exhausted":0
        }
    },

    "upstreams": {
        "upstream": {
            "peers": {
                "192.168.16.4:80": {
                    "server":"backend.example.com",
                    "service":"_example._tcp",
                    "backup":false,
```

```
                    "weight":5,
                    "state":"up",
                    "selected": {
                        "current":2,
                        "total":232
                    },

                    "max_conns":5,
                    "responses": {
                        "200":222,
                        "302":12
                    },

                    "data": {
                        "sent":543866,
                        "received":27349934
                    },

                    "health": {
                        "fails":0,
                        "unavailable":0,
                        "downtime":0
                    },

                    "sid":"<server_id>"
                }
            },

            "keepalive":2
        }
    },

    "resolvers": {
        "resolver_zone": {
            "queries": {
                "name":442,
                "srv":2,
                "addr":0
            },

            "responses": {
                "success":440,
                "timedout":1,
                "format_error":0,
                "server_failure":1,
                "not_found":1,
                "unimplemented":0,
```

```
                "refused":1,
                "other":0
            }
        }
    }
}
```

可以通过构建适当的请求来获取单个 JSON 分支的指标集。例如：

```
$ curl https://www.example.com/status/angie
$ curl https://www.example.com/status/connections
$ curl https://www.example.com/status/slabs
$ curl https://www.example.com/status/slabs/<zone>/slots
$ curl https://www.example.com/status/slabs/<zone>/slots/64
$ curl https://www.example.com/status/http/
$ curl https://www.example.com/status/http/server_zones
$ curl https://www.example.com/status/http/server_zones/<http_server_zone>
$ curl https://www.example.com/status/http/server_zones/<http_server_zone>/ssl
```

备注

默认情况下，模块使用 ISO 8601 格式字符串表示日期；要使用整数 UNIX 纪元格式，请在查询字符串中添加 `date=epoch` 参数：

```
$ curl https://www.example.com/status/angie/load_time

  "2024-04-01T00:59:59+01:00"

$ curl https://www.example.com/status/angie/load_time?date=epoch

  1711929599
```

服务器状态

`/status/angie`

```
{
    "version": "1.10.0",
    "build_time": "2025-07-03T16:05:43.805Z",
    "address": "192.168.16.5",
    "generation": 1,
    "load_time": "2025-07-03T16:15:43.805Z"
    "config_files": {
        "/etc/angie/angie.conf": "...",
        "/etc/angie/mime.types": "..."
    }
}
```

version	字符串；正在运行的 Angie Web 服务器的版本
build	字符串；编译期间指定的特定构建名称
build_time	字符串；Angie 可执行文件的构建时间，采用*date* 格式
address	字符串；接受 API 请求的服务器地址
generation	数字；自上次启动以来配置重新加载的总次数
load_time	字符串；最后一次配置重新加载的时间，采用*date* 格式；字符串值具有毫秒级精度
config_files	对象；其成员是服务器实例当前加载的所有 Angie 配置文件的绝对路径，其值是这些文件内容的字符串表示，例如： `{ "/etc/angie/angie.conf": "server {\n  listen 80;\n  # ...\n\n}\n" }` **小心** 只有在启用了*api_config_files* 指令的情况下，config_files 对象才在 /status/angie/ 中可用。

连接

/status/connections

```
{
  "accepted": 2257,
  "dropped": 0,
  "active": 3,
  "idle": 1
}
```

accepted	数字；已接受的客户端连接总数
dropped	数字；已丢弃的客户端连接总数
active	数字；当前活动的客户端连接数
idle	数字；当前空闲的客户端连接数

使用 slab 分配的共享内存区域

/status/slabs/<zone>

使用 slab 分配 的共享内存区域的使用统计信息，如*limit_conn*、*limit_req* 和*HTTP 缓存*：

```
limit_conn_zone $binary_remote_addr zone=limit_conn_zone:10m;
limit_req_zone $binary_remote_addr zone=limit_req_zone:10m rate=1r/s;
proxy_cache cache_zone;
```

指定的共享内存区域将收集以下统计信息：

pages	对象；内存页统计信息
used	数字；当前使用的内存页数量
free	数字；当前空闲的内存页数量
slots	对象；每个槽大小的内存槽统计信息。slots 对象包含内存槽大小的数据（8、16、32 等，最大为页面大小的一半，以字节为单位）
used	数字；当前使用的指定大小内存槽的数量
free	数字；当前空闲的指定大小内存槽的数量
reqs	数字；尝试分配指定大小内存的总次数
fails	数字；尝试分配指定大小内存失败的次数

示例：

```
{
  "pages": {
    "used": 2,
    "free": 506
  },

  "slots": {
    "64": {
      "used": 1,
      "free": 63,
      "reqs": 1,
      "fails": 0
  }
}
```

解析器的 DNS 查询

`/status/resolvers/<zone>`

要收集解析器统计信息，*resolver* 指令必须设置 status_zone 参数（*HTTP* 或*Stream*）：

```
resolver 127.0.0.53 status_zone=resolver_zone;
```

指定的共享内存区域将收集以下统计信息：

queries	对象；查询统计信息
name	数字；将名称解析为地址的查询数量（A 和 AAAA 查询）
srv	数字；将服务解析为地址的查询数量（SRV 查询）
addr	数字；将地址解析为名称的查询数量（PTR 查询）
responses	对象；响应统计信息
success	数字；成功响应的数量
timedout	数字；超时查询的数量
format_error	数字；代码为 1（格式错误）的响应数量
server_failure	数字；代码为 2（服务器故障）的响应数量
not_found	数字；代码为 3（名称错误）的响应数量
unimplemented	数字；代码为 4（未实现）的响应数量
refused	数字；代码为 5（拒绝）的响应数量
other	数字；以其他非零代码完成的查询数量
sent	对象；已发送 DNS 查询的统计信息
a	数字；A 类型查询的数量
aaaa	数字；AAAA 类型查询的数量
ptr	数字；PTR 类型查询的数量
srv	数字；SRV 类型查询的数量

响应代码在 RFC 1035 的 4.1.1 部分中有描述。

各种 DNS 记录类型在 RFC 1035、RFC 2782 和 RFC 3596 中有详细说明。

示例：

```
{
  "queries": {
    "name": 442,
    "srv": 2,
    "addr": 0
  },

  "responses": {
    "success": 440,
    "timedout": 1,
    "format_error": 0,
    "server_failure": 1,
    "not_found": 1,
    "unimplemented": 0,
    "refused": 1,
    "other": 0
  },

  "sent": {
    "a": 185,
    "aaaa": 245,
    "srv": 2,
```

```
    "ptr": 12
  }
}
```

HTTP 服务器和位置

`/status/http/server_zones/<zone>`

要收集 server 指标，在*server* 上下文中设置*status_zone* 指令：

```
server {
    ...
    status_zone server_zone;
}
```

要按自定义值对指标进行分组，请使用替代语法。在这里，指标按*$host* 聚合，每个组作为独立区域报告：

```
status_zone $host zone=server_zone:5;
```

指定的共享内存区域将收集以下统计信息：

ssl	对象；SSL 统计信息。如果 server 设置了 listen ssl;，则存在
handshaked	数字；成功的 SSL 握手总数
reuses	数字；SSL 握手期间会话重用的总数
timedout	数字；超时的 SSL 握手总数
failed	数字；失败的 SSL 握手总数
requests	对象；请求统计信息
total	数字；客户端请求总数
processing	数字；当前正在处理的客户端请求数
discarded	数字；未发送响应而完成的客户端请求总数
responses	对象；响应统计信息
<code>	数字；状态为 <code>（100-599）的非零响应数
xxx	数字；其他状态代码的非零响应数
data	对象；数据统计信息
received	数字；从客户端接收的字节总数
sent	数字；发送给客户端的字节总数

示例：

```
{
    "ssl":{
        "handshaked":4174,
        "reuses":0,
        "timedout":0,
        "failed":0
    },
```

```
    "requests":{
        "total":4327,
        "processing":0,
        "discarded":0
    },

    "responses":{
        "200":4305,
        "302":6,
        "304":12,
        "404":4
    },

    "data":{
        "received":733955,
        "sent":59207757
    }
}
```

`/status/http/location_zones/<zone>`

要收集 `location` 指标，请在*location* 或*if in location* 上下文中设置*status_zone* 指令：

```
location / {
    root /usr/share/angie/html;
    status_zone location_zone;

    if ($request_uri ~* "^/condition") {
        # ...
        status_zone if_location_zone;
    }
}
```

要按自定义值对指标进行分组，请使用替代语法。在这里，指标按*$host* 聚合，每个组作为独立区域报告：

```
status_zone $host zone=server_zone:5;
```

指定的共享内存区域将收集以下统计信息：

requests	对象；请求统计信息
total	数字；客户端请求总数
discarded	数字；未发送响应而完成的客户端请求总数
responses	对象；响应统计信息
<code>	数字；状态码为 <code> 的非零响应数量（100-599）
xxx	数字；其他状态码的非零响应数量
data	对象；数据统计信息
received	数字；从客户端接收的字节总数
sent	数字；发送给客户端的字节总数

示例：

```
{
  "requests": {
    "total": 4158,
    "discarded": 0
  },

  "responses": {
    "200": 4157,
    "304": 1
  },

  "data": {
    "received": 538200,
    "sent": 177606236
  }
}
```

流服务器

/status/stream/server_zones/<zone>

要收集 server 指标，请在*server* 上下文中设置*status_zone* 指令：

```
server {
    ...
    status_zone server_zone;
}
```

要按自定义值对指标进行分组，请使用替代语法。在这里，指标按*$host* 聚合，每个组作为独立区域报告：

```
status_zone $host zone=server_zone:5;
```

指定的共享内存区域将收集以下统计信息：

ssl	对象；SSL 统计信息。当 server 设置 listen ssl; 时存在
handshaked	数字；成功的 SSL 握手总数
reuses	数字；SSL 握手期间会话重用的总数
timedout	数字；超时的 SSL 握手总数
failed	数字；失败的 SSL 握手总数
connections	对象；连接统计信息
total	数字；客户端连接总数
processing	数字；当前正在处理的客户端连接数
discarded	数字；未创建会话而完成的客户端连接总数
passed	数字；通过 pass 指令转发到另一个监听端口的客户端连接总数
sessions	对象；会话统计信息
success	数字；以代码 200 完成的会话数，表示成功完成
invalid	数字；以代码 400 完成的会话数，当客户端数据无法解析时发生，例如 PROXY 协议头
forbidden	数字；以代码 403 完成的会话数，当访问被禁止时，例如当某些客户端地址的访问受到限制时
internal_error	数字；以代码 500 完成的会话数，内部服务器错误
bad_gateway	数字；以代码 502 完成的会话数，网关错误，例如当无法选择或到达上游服务器时
service_unavailable	数字；以代码 503 完成的会话数，服务不可用，例如当连接数受到限制时
data	对象；数据统计信息
received	数字；从客户端接收的字节总数
sent	数字；发送给客户端的字节总数

示例：


```
{
  "ssl": {
    "handshaked": 24,
    "reuses": 0,
    "timedout": 0,
    "failed": 0
  },

  "connections": {
    "total": 24,
    "processing": 1,
    "discarded": 0,
    "passed": 2
  },

  "sessions": {
    "success": 24,
    "invalid": 0,
    "forbidden": 0,
```

```
    "internal_error": 0,
    "bad_gateway": 0,
    "service_unavailable": 0
  },

  "data": {
    "received": 2762947,
    "sent": 53495723
  }
}
```

HTTP 缓存

```
proxy_cache cache_zone;
```

第二个文件内容：.. code-block:: console

$ curl https://www.example.com/status/angie $ curl https://www.example.com/status/connections $ curl https://www.example.com/status/slabs $ curl https://www.example.com/status/slabs/<zone>/slots $ curl https://www.example.com/status/slabs/<zone>/slots/64 $ curl https://www.example.com/status/http/ $ curl https://www.example.com/status/http/server_zones $ curl https://www.example.com/status/http/server_zones/<http_server_zone> $ curl https://www.example.com/status/http/server_zones/<http_server_zone>/ssl

/status/http/caches/<cache>

对于每个使用*proxy_cache*配置的区域，存储以下数据：

```
{
  "name_zone": {
    "size": 0,
    "cold": false,
    "hit": {
      "responses": 0,
      "bytes": 0
    },

    "stale": {
      "responses": 0,
      "bytes": 0
    },

    "updating": {
      "responses": 0,
      "bytes": 0
    },
```

```
    "revalidated": {
      "responses": 0,
      "bytes": 0
    },

    "miss": {
      "responses": 0,
      "bytes": 0,
      "responses_written": 0,
      "bytes_written": 0
    },

    "expired": {
      "responses": 0,
      "bytes": 0,
      "responses_written": 0,
      "bytes_written": 0
    },

    "bypass": {
      "responses": 0,
      "bytes": 0,
      "responses_written": 0,
      "bytes_written": 0
    }
  }
}
```

size	数字；缓存的当前大小
max_size	数字；缓存最大大小的配置限制
cold	布尔值；当 ‘缓存加载器 ‘从磁盘加载数据时为 true
hit	对象；有效缓存响应的统计信息 (*proxy_cache_valid*)
responses	数字；从缓存读取的响应总数
bytes	数字；从缓存读取的字节总数
stale	对象；从缓存中获取的过期响应的统计信息 (*proxy_cache_use_stale*)
responses	数字；从缓存读取的响应总数
bytes	数字；从缓存读取的字节总数
updating	对象；在更新响应时从缓存中获取的过期响应的统计信息 (*proxy_cache_use_stale* updating)
responses	数字；从缓存读取的响应总数
bytes	数字；从缓存读取的字节总数
revalidated	对象；从缓存中获取的过期和重新验证的响应的统计信息 (*proxy_cache_revalidate*)
responses	数字；从缓存读取的响应总数
bytes	数字；从缓存读取的字节总数
miss	对象；在缓存中未找到的响应的统计信息
responses	数字；相应响应的总数
bytes	数字；从代理服务器读取的字节总数
responses_written	数字；写入缓存的响应总数
bytes_written	数字；写入缓存的字节总数
expired	对象；未从缓存中获取的过期响应的统计信息
responses	数字；相应响应的总数
bytes	数字；从代理服务器读取的字节总数
responses_written	数字；写入缓存的响应总数
bytes_written	数字；写入缓存的字节总数
bypass	对象；未在缓存中查找的响应的统计信息 (*proxy_cache_bypass*)
responses	数字；相应响应的总数
bytes	数字；从代理服务器读取的字节总数
responses_written	数字；写入缓存的响应总数
bytes_written	数字；写入缓存的字节总数

Added in version 1.2.0: PRO

在 Angie PRO 中，如果通过*proxy_cache_path* 指令启用了 ‘缓存分片 ‘，各个分片将作为 shards 对象的成员公开：

shards	对象；将各个分片列为成员
<shard>	对象；表示一个单独的分片，其缓存路径作为名称
size	数字；分片的当前大小
max_size	数字；最大分片大小，如果已配置
cold	布尔值；当 ‘缓存加载器 ‘从磁盘加载数据时为 true

```
{
  "name_zone": {
    "shards": {
        "/path/to/shard1": {
            "size": 0,
            "cold": false
        },

        "/path/to/shard2": {
            "size": 0,
            "cold": false
        }
    }
}
```

limit_conn

```
limit_conn_zone $binary_remote_addr zone=limit_conn_zone:10m;
```

/status/http/limit_conns/<zone>, /status/stream/limit_conns/<zone>

为每个配置的*limit_conn in http* 或*limit_conn in stream* 上下文提供具有以下字段的对象：

```
{
  "passed": 73,
  "skipped": 0,
  "rejected": 0,
  "exhausted": 0
}
```

passed	数字；通过的连接总数
skipped	数字；通过的具有零长度键或超过 255 字节的键的连接总数
rejected	数字；超过配置限制的连接总数
exhausted	数字；由于区域存储耗尽而被拒绝的连接总数

limit_req

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

/status/http/limit_reqs/<zone>

为每个配置的*limit_req* 提供具有以下字段的对象：

```
{
  "passed": 54816,
```

```
  "skipped": 0,
  "delayed": 65,
  "rejected": 26,
  "exhausted": 0
}
```

passed	数字；通过的请求总数
skipped	数字；通过的具有零长度键或超过 255 字节的键的请求总数
delayed	数字；延迟的请求总数
rejected	数字；拒绝的请求总数
exhausted	数字；由于区域存储耗尽而被拒绝的请求总数

HTTP 上游

Added in version 1.1.0.

要启用以下指标的收集，请在*upstream* 上下文中设置*zone* 指令，例如：

```
upstream upstream {
    zone upstream 256k;
    server backend.example.com service=_example._tcp resolve max_conns=5;
    keepalive 4;
}
```

/status/http/upstreams/<upstream>

其中 *<upstream>* 是使用*zone* 指令指定的任何*upstream* 的名称

```
{
    "peers": {
        "192.168.16.4:80": {
            "server": "backend.example.com",
            "service": "_example._tcp",
            "backup": false,
            "weight": 5,
            "state": "up",
            "selected": {
                "current": 2,
                "total": 232
            },

            "max_conns": 5,
            "responses": {
                "200": 222,
                "302": 12
            },
```

```
            "data": {
                "sent": 543866,
                "received": 27349934
            },

            "health": {
                "fails": 0,
                "unavailable": 0,
                "downtime": 0
            },

            "sid": "<server_id>"
        }
    },

    "keepalive": 2
}
```

peers	对象；包含上游对等节点的指标作为子对象，其名称是对等节点地址的规范表示。每个子对象的成员：
server	字符串；*server* 指令的参数
service	字符串；如果配置了，则为*server* 指令中指定的服务名称
slow_start (PRO 1.4.0+)	数字；为服务器指定的*slow_start* 值，以秒为单位表示。 通过动态配置 API 的相应子部分 设置值时，可以指定数字或具有毫秒精度的时间 值。
backup	布尔值；备份服务器为 `true`
weight	数字；配置的*weight*
state	字符串；对等节点的当前状态以及向其发送的请求： • `busy`：表示对服务器的请求数量已达到*max_conns* 设置的限制，不会发送新请求 • `down`：手动禁用，不发送请求 • `recovering`：根据*slow_start* 从故障中恢复，随着时间的推移会发送越来越多的请求 • `unavailable`：达到*max_fails* 限制，仅以*fail_timeout* 定义的间隔发送试探性客户端请求； • `up`：正常运行，照常发送请求 Angie PRO 中的其他状态： • `checking`：配置为 `essential` 并正在检查，仅发送探测请求 • `draining`：类似于 `down`，但仍然发送来自先前绑定会话的请求（通过*sticky*） • `unhealthy`：非运行状态，仅发送探测请求
selected	对象；对等节点选择统计信息
current	数字；当前与对等节点的连接数
total	数字；转发到对等节点的请求总数
last	字符串或数字；最后一次选择对等节点的时间，格式化为日期
max_conns	数字；如果指定，则为配置的最大 同时连接数
responses	对象；响应统计信息
<code>	数字；状态为 <code>（100-599）的非零响应数量
xxx	数字；其他状态码的非零响应数量
data	对象；数据统计信息
received	数字；从对等节点接收的总字节数
sent	数字；发送到对等节点的总字节数
health	对象；健康统计信息
fails	数字；与对等节点通信的不成功尝试总数
unavailable	数字；由于达到*max_fails* 限制，对等节点变为 `unavailable` 的次数
downtime	数字；对等节点 `unavailable` 无法选择的总时间（以毫秒为单位）
downstart	字符串或数字；对等节点变为 `unavailable` 的时间，格式化为日期
header_time (PRO 1.3.0+)	数字；从对等节点接收响应头的平均时间（以毫秒为单位）；参见*response_time_factor (PRO)*
response_time (PRO 1.3.0+)	数字；接收整个对等节点响应的平均时间（以毫秒为单位）；参见*response_time_factor (PRO)*
sid	字符串；上游组中服务器的配置 *id*
keepalive	数字；当前缓存的连接数
backup_switch	对象；包含主动备份逻辑的当前状态，如果为上游配置了*backup_switch (PRO)*，则存在
active	数字；活动组标识符（如果有）

health/probes (PRO)

在 1.2.0 版本发生变更: PRO

如果上游配置了*upstream_probe (PRO)* 探测，health 对象还有一个 probes 子对象，用于存储对等节点的健康探测计数器，而对等节点的 state 除了上表中列出的值外，还可以是 checking 和 unhealthy：

```
{
    "192.168.16.4:80": {
        "state": "unhealthy",
        "...": "...",
        "health": {
            "...": "...",
            "probes": {
                "count": 10,
                "fails": 10,
                "last": "2025-07-03T09:56:07Z"
            }
        }
    }
}
```

state 的 checking 值不计入 downtime，表示配置为 essential 的对等节点尚未被检查；unhealthy 值表示对等节点运行不正常。这两种状态也意味着对等节点不包括在负载均衡中。有关健康探测的详细信息，请参见*upstream_probe*。

probes 中的计数器：

count	数字；此对等节点的总探测次数
fails	数字；总失败探测次数
last	字符串或数字；最后探测时间，格式化为日期

queue (PRO)

在 1.4.0 版本发生变更: PRO

如果为上游配置了请求队列，上游对象还包含一个嵌套的 queue 对象，其中包含请求队列计数器：

```
{
    "queue": {
        "queued": 20112,
        "waiting": 1011,
        "dropped": 6031,
        "timedout": 560,
        "overflows": 13
    }
}
```

计数器值在所有工作进程中累加：

queued	数字；进入队列的请求总数
waiting	数字；队列中当前的请求数
dropped	数字；因客户端提前关闭连接而从队列中移除的请求总数
timedout	数字；因超时而从队列中移除的请求总数
overflows	数字；队列溢出发生的总次数

流上游

要启用以下指标的收集，在*upstream* 上下文中设置*zone* 指令，例如：

```
upstream upstream {
    zone upstream 256k;
    server backend.example.com service=_example._tcp resolve max_conns=5;
    keepalive 4;
}
```

/status/stream/upstreams/<upstream>

这里，*<upstream>* 是通过*zone* 指令配置的*upstream* 的名称。

```
{
    "peers": {
        "192.168.16.4:1935": {
            "server": "backend.example.com",
            "service": "_example._tcp",
            "backup": false,
            "weight": 5,
            "state": "up",
            "selected": {
                "current": 2,
                "total": 232
            },

            "max_conns": 5,
            "data": {
                "sent": 543866,
                "received": 27349934
            },

            "health": {
                "fails": 0,
                "unavailable": 0,
                "downtime": 0
            }
        }
    }
}
```

peers	对象；包含上游对等体的指标作为子对象，其名称是对等体地址的规范表示。每个子对象的成员：
server	字符串；由*server* 指令设置的地址
service	字符串；服务名称，如在*server* 指令中设置
slow_start (PRO 1.4.0+)	数字；为服务器指定的*slow_start* 值，以秒为单位表示。当通过动态配置 API 的相应子部分 设置值时，可以指定数字或具有毫秒精度的时间 值。
backup	布尔值；对于备份服务器为 true
weight	数字；对等体的*weight*
state	字符串；对等体的当前状态以及向其发送的请求类型： • busy：表示发送到服务器的请求数量已达到*max_conns* 限制，不再发送新的请求 • down：手动禁用，不发送任何请求 • recovering：故障恢复中，按照*slow_start*，发送的请求数量逐渐增加 • unavailable：达到*max_fails* 限制，仅以*fail_timeout* 间隔发送试探性客户端请求； • up：运行正常，正常发送请求 Angie PRO 中的附加状态： • checking：配置为 essential 并处于检查中，仅发送探测请求 • draining：类似于 down，但仍会发送先前通过*sticky* 绑定的会话请求 • unhealthy：不可用，仅发送探测请求
selected	对象；对等体选择统计信息
current	数字；当前连接到对等体的连接数
total	数字；转发到对等体的连接总数
last	字符串或数字；上次选择对等体的时间，格式化为*date*
max_conns	数字；最大 同时连接到对等体的数量（如果已设置）
data	对象；数据传输统计信息
received	数字；从对等体接收到的字节总数
sent	数字；发送到对等体的字节总数
health	对象；对等体健康统计信息
fails	数字；尝试连接对等体失败的总次数
unavailable	数字；由于达到*max_fails* 限制而变为 unavailable 的次数
downtime	数字；对等体在选择时处于 unavailable 状态的总时间（以毫秒为单位）
downstart	字符串或数字；对等体最后一次变为 unavailable 的时间，格式化为*date*
connect_time (PRO 1.4.0+)	数字；与对等体建立连接所需的平均时间（以毫秒为单位）；参见*response_time_factor (PRO)* 指令。
first_byte_time (PRO 1.4.0+)	数字；从对等体接收响应的第一个字节的平均时间（以毫秒为单位）；参见*response_time_factor (PRO)* 指令。
last_byte_time (PRO 1.4.0+)	数字；从对等体接收完整响应的平均时间（以毫秒为单位）；参见*response_time_factor (PRO)* 指令。
backup_switch (PRO 1.10.0+)	对象；包含当前的主动备用逻辑状态，当上游配置了*backup_switch (PRO)* 时存在
active	数字；当前用于请求负载均衡的活动组级别。如果活动组是主组，则值为 0
timeout	数字；剩余等待时间（毫秒），到期后负载均衡器将重新检查较低级别组中的健康节点，从主组开始，而不检查较高级别的组；主组（级别 0）不显示此字段

在 1.4.0 版本发生变更: PRO

在 Angie PRO 中，如果上游配置了*upstream_probe (PRO)* 探针，则 `health` 对象还具有一个 `probes` 子对象，存储对等体的健康探针计数器，同时对等体的 `state` 也可以是 `checking` 和 `unhealthy`，除了上表中列出的值：

```
{
    "192.168.16.4:80": {
        "state": "unhealthy",
        "...": "...",
        "health": {
            "...": "...",
            "probes": {
                "count": 2,
                "fails": 2,
                "last": "2025-07-03T11:03:54Z"
            }
        }
    }
}
```

`state` 的 `checking` 值表示配置为 `essential` 的对等体尚未被检查；`unhealthy` 值表示对等体出现故障。这两种状态也意味着对等体不包含在负载均衡中。有关健康探针的详细信息，请参见*upstream_probe*。

`probes` 中的计数器：

`count`	数字；此对等体的探针总数
`fails`	数字；探针失败的总数
`last`	字符串或数字；最后一次探测时间，格式化为*date*

动态配置 API（仅 PRO 版）

Added in version 1.2.0: PRO

API 包含一个 `/config` 部分，允许通过 `PUT`、`PATCH` 和 `DELETE` HTTP 请求以 JSON 格式动态更新 Angie 的配置。所有更新都是原子性的；新设置要么全部应用，要么一个都不应用。出错时，Angie 会报告原因。

`/config` 的子部分

目前，`/config` 部分中可以动态配置上游内的单个服务器，适用于*HTTP* 和*stream* 模块；可动态配置的设置数量正在稳步增加。

`/config/http/upstreams/<upstream>/servers/<name>`

允许配置单个上游对等体，包括删除现有对等体或添加新对等体。

URI 路径参数：

<upstream>	上游的名称；要通过 /config 进行配置，必须配置一个*zone* 指令，以定义一个共享内存区域。
<name>	对等体在上游中的名称，定义为 <service>@<host>，其中： • <service>@ 是可选的服务名称，用于 SRV 记录解析。 • <host> 是服务的域名（如果存在 resolve）或其 IP；此处可以定义一个可选端口。

例如，以下配置：

```
upstream backend {
    server backend.example.com service=_http._tcp resolve;
    server 127.0.0.1;
    zone backend 1m;
}
```

允许以下对等体名称：

```
$ curl http://127.0.0.1/config/http/upstreams/backend/servers/_http._tcp@backend.example.com/
$ curl http://127.0.0.1/config/http/upstreams/backend/servers/127.0.0.1:80/
```

该 API 子部分允许设置 weight、max_conns、max_fails、fail_timeout、backup、down 和 sid 参数，如*server* 中所述。

备注

此处没有单独的 drain (PRO) 参数；要启用 drain，将 down 设置为字符串值 drain：

```
$ curl -X PUT -d \"drain\" \
  http://127.0.0.1/config/http/upstreams/backend/servers/backend.example.com/down
```

示例：

```
$ curl http://127.0.0.1/config/http/upstreams/backend/servers/backend.example.com?defaults=on
```

```
{
    "weight": 1,
    "max_conns": 0,
    "max_fails": 1,
    "fail_timeout": 10,
    "backup": true,
    "down": false,
    "sid": ""
}
```

实际可用的参数仅限于当前*upstream* 负载均衡方法支持的参数。因此，如果上游配置为 random 方法：

```
upstream backend {
    zone backend 256k;
    server backend.example.com resolve max_conns=5;
    random;
}
```

您将无法添加定义 backup 的新对等体：

```
$ curl -X PUT -d '{ "backup": true }' \
    http://127.0.0.1/config/http/upstreams/backend/servers/backend1.example.com
```

```
{
    "error": "FormatError",
    "description": "The \"backup\" field is unknown."
}
```

备注

即使使用兼容的负载均衡方法，backup 参数只能在添加新对等体时设置。

`/config/stream/upstreams/<upstream>/servers/<name>`

允许配置上游中的单个服务器，包括添加新服务器和删除已配置的服务器。

URI 路径参数：

<upstream>	upstream 块的名称；要通过 /config 进行配置，它必须包含定义共享内存区域的*zone* 指令。
<name>	指定 <upstream> 中特定服务器的名称；以 <service>@<host> 格式指定，其中： • <service>@ —可选部分，指定用于解析 SRV 记录的服务名称。 • <host> —服务的域名（当存在 *resolve* 时）或 IP 地址；也可以指定端口。

例如，对于以下配置：

```
upstream backend {
    server backend.example.com:8080 service=_example._tcp resolve;
    server 127.0.0.1:12345;
    zone backend 1m;
}
```

以下服务器名称有效：

```
$ curl http://127.0.0.1/config/stream/upstreams/backend/servers/_example._tcp@backend.example.
↪com:8080/
$ curl http://127.0.0.1/config/stream/upstreams/backend/servers/127.0.0.1:12345/
```

此 API 子部分允许设置*server* 部分中描述的 `weight`、`max_conns`、`max_fails`、`fail_timeout`、`backup` 和 `down` 参数。

备注

没有单独的 `drain` 参数 (PRO)；要启用 `drain` 模式，将 `down` 参数设置为字符串值 `drain`：

```
$ curl -X PUT -d \"drain\" \
  http://127.0.0.1/config/stream/upstreams/backend/servers/backend.example.com/down
```

示例：

```
curl http://127.0.0.1/config/stream/upstreams/backend/servers/backend.example.com?defaults=on
```

```
{
    "weight": 1,
    "max_conns": 0,
    "max_fails": 1,
    "fail_timeout": 10,
    "backup": true,
    "down": false,
}
```

只有当前*upstream* 负载均衡方法支持的参数才会实际可用。例如，如果上游配置了 `random` 负载均衡方法：

```
upstream backend {
    zone backend 256k;
    server backend.example.com resolve max_conns=5;
    random;
}
```

那么无法添加带有 `backup` 参数的新服务器：

```
$ curl -X PUT -d '{ "backup": true }' \
    http://127.0.0.1/config/stream/upstreams/backend/servers/backend1.example.com
```

```
{
    "error": "FormatError",
    "description": "The \"backup\" field is unknown."
}
```

> **备注**
>
> 即使使用兼容的负载均衡方法，`backup` 参数也只能在添加新服务器时设置。

删除服务器时，可以设置 `connection_drop=<value>` 参数 (PRO) 来覆盖*proxy_connection_drop* 设置：

```
$ curl -X DELETE \
    http://127.0.0.1/config/stream/upstreams/backend/servers/backend1.example.com?connection_
↪drop=off

$ curl -X DELETE \
    http://127.0.0.1/config/stream/upstreams/backend/servers/backend2.example.com?connection_
↪drop=on

$ curl -X DELETE \
    http://127.0.0.1/config/stream/upstreams/backend/servers/backend3.example.com?connection_
↪drop=1000
```

HTTP 方法

让我们考虑适用于此部分的所有 HTTP 方法的语义，给定以下上游配置：

```
http {
    # ...

    upstream backend {
        zone upstream 256k;
        server backend.example.com resolve max_conns=5;
        # ...
    }

    server {
        # ...

        location /config/ {
            api /config/;

            allow 127.0.0.1;
            deny all;
        }
    }
}
```

GET

GET HTTP 方法在 /config 中的任何现有路径查询实体，就像它对其他 API 部分所做的那样。

例如，/config/http/upstreams/backend/servers/ 上游服务器分支允许这些查询：

```
$ curl http://127.0.0.1/config/http/upstreams/backend/servers/backend.example.com/max_conns
$ curl http://127.0.0.1/config/http/upstreams/backend/servers/backend.example.com
$ curl http://127.0.0.1/config/http/upstreams/backend/servers
$ # ...
$ curl http://127.0.0.1/config
```

您可以使用 defaults=on 获取默认参数值：

```
$ curl http://127.0.0.1/config/http/upstreams/backend/servers?defaults=on
```

```
{
    "backend.example.com": {
        "weight": 1,
        "max_conns": 5,
        "max_fails": 1,
        "fail_timeout": 10,
        "backup": false,
        "down": false,
        "sid": ""
    }
}
```

PUT

PUT HTTP 方法在指定路径创建一个新的 JSON 实体或 完全替换一个现有实体。

例如，要设置 backend.example.com 服务器在 backend 上游中早先未指定的 max_fails 参数：

```
$ curl -X PUT -d '2' \
    http://127.0.0.1/config/http/upstreams/backend/servers/backend.example.com/max_fails
```

```
{
    "success": "Updated",
    "description": "Existing configuration API entity \"/config/http/upstreams/backend/servers/
↪backend.example.com/max_fails\" was updated with replacing."
}
```

验证更改：

```
$ curl http://127.0.0.1/config/http/upstreams/backend/servers/backend.example.com
```

```
{
    "max_conns": 5,
    "max_fails": 2
}
```

DELETE

DELETE HTTP 方法删除指定路径下的 先前定义设置；在执行此操作时，如果有任何默认值，则会返回到默认值。

例如，要删除 backend 上游中 backend.example.com 服务器的先前设置的 max_fails 参数：

```
$ curl -X DELETE \
    http://127.0.0.1/config/http/upstreams/backend/servers/backend.example.com/max_fails
```

```
{
    "success": "Reset",
    "description": "Configuration API entity \"/config/http/upstreams/backend/servers/backend.
↪example.com/max_fails\" was reset to default."
}
```

使用 defaults=on 验证更改：

```
$ curl http://127.0.0.1/config/http/upstreams/backend/servers/backend.example.com?defaults=on
```

```
{
    "weight": 1,
    "max_conns": 5,
    "max_fails": 1,
    "fail_timeout": 10,
    "backup": false,
    "down": false,
    "sid": ""
}
```

max_fails 设置已恢复为默认值。

在删除服务器时，您可以设置 connection_drop=<value> 参数（PRO）以覆盖*proxy_connection_drop*、*grpc_connection_drop*、*fastcgi_connection_drop*、*scgi_connection_drop* 和*uwsgi_connection_drop* 设置：

```
$ curl -X DELETE \
    http://127.0.0.1/config/http/upstreams/backend/servers/backend1.example.com?connection_
↪drop=off

$ curl -X DELETE \
    http://127.0.0.1/config/http/upstreams/backend/servers/backend2.example.com?connection_
↪drop=on
```

```
$ curl -X DELETE \
    http://127.0.0.1/config/http/upstreams/backend/servers/backend3.example.com?connection_
↪drop=1000
```

PATCH

PATCH HTTP 方法在指定路径下创建一个新实体或部分替换或补充现有的实体 (RFC 7386) 通过在其有效负载中提供 JSON 定义。

该方法的操作如下：如果新定义中的实体在配置中存在，则它们会被覆盖；否则，它们将被添加。

例如，要更改 backend 上游中 backend.example.com 服务器的 down 设置，同时保持其他设置不变：

```
$ curl -X PATCH -d '{ "down": true }' \
    http://127.0.0.1/config/http/upstreams/backend/servers/backend.example.com
```

```
{
    "success": "Updated",
    "description": "Existing configuration API entity \"/config/http/upstreams/backend/servers/
↪backend.example.com\" was updated with merging."
}
```

验证更改：

```
$ curl http://127.0.0.1/config/http/upstreams/backend/servers/backend.example.com
```

```
{
    "max_conns": 5,
    "down": true
}
```

与 PUT 不同，随 PATCH 请求提供的 JSON 对象 已与现有对象合并，而不是覆盖它。

null 值是一个特殊情况；它们用于 删除在这种合并过程中的特定配置项目。

> **备注**
>
> 此删除与 DELETE 相同；特别是，它会恢复默认值。

例如，要删除之前添加的 down 设置并同时更新 max_conns：

```
$ curl -X PATCH -d '{ "down": null, "max_conns": 6 }' \
    http://127.0.0.1/config/http/upstreams/backend/servers/backend.example.com
```

```
{
    "success": "Updated",
```

```
    "description": "Existing configuration API entity \"/config/http/upstreams/backend/servers/
↪backend.example.com\" was updated with merging."
}
```

验证更改：

```
$ curl http://127.0.0.1/config/http/upstreams/backend/servers/backend.example.com
```

```
{
    "max_conns": 6
}
```

提供了 `null` 的 `down` 参数被删除；`max_conns` 被更新。

Auth Basic

允许通过使用"HTTP 基本认证" 协议验证用户名和密码来限制对资源的访问。

访问也可以通过地址 或通过子请求结果 来限制。通过地址和密码同时限制访问由*satisfy* 指令控制。

配置示例

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

指令

auth_basic

语法	`auth_basic` *string* | `off`;
默认	`auth_basic off;`
上下文	http, server, location, limit_except

通过"HTTP 基本认证" 协议启用用户名和密码验证。指定的参数用作 领域。参数值可以包含变量。

`off`	取消从先前配置级别继承的 *auth_basic* 指令的效果

auth_basic_user_file

语法	`auth_basic_user_file` *file*;
默认	—
上下文	http, server, location, limit_except

指定一个 文件来保存用户名和密码，格式如下：

```
# 注释
name1:password1
name2:password2:comment
name3:password3
```

文件名可以包含变量。

支持以下密码类型：

- 使用 *crypt()* 函数加密；可以使用 Apache HTTP Server 发行版中的 `htpasswd` 工具或"*openssl passwd*" 命令生成；
- 使用基于 MD5 的密码算法的 Apache 变体（apr1）进行哈希；可以使用相同的工具生成；
- 按照 RFC 2307 中描述的"*{scheme}data*" 语法指定；目前实现的方案包括 *PLAIN* (一个示例，不应使用)、*SHA* (纯 SHA-1 哈希，不应使用）和 *SSHA* (加盐 SHA-1 哈希，一些软件包使用，尤其是 OpenLDAP 和 Dovecot)。

小心

SHA 方案的支持仅为从其他 Web 服务器迁移提供帮助。它不应用于新密码，因为采用的未加盐 SHA-1 哈希易受 彩虹表 攻击。

Auth Request

根据子请求的结果实现客户端授权。如果子请求返回 2xx 响应代码，则允许访问。如果返回 401 或 403，则拒绝访问，并返回相应的错误代码。子请求返回的任何其他响应代码都被视为错误。

对于 401 错误，客户端还会收到来自子请求响应的"WWW-Authenticate" 头。

当 从源代码构建时，该模块默认未构建；应通过 `--with-http_auth_request_module` 构建选项启用。

在来自 我们的仓库的包和镜像中，模块已包含在构建中。

该模块可以与其他访问模块组合，例如*Access* 和*Auth Basic*，通过*satisfy* 指令。

配置示例

```
location /private/ {
    auth_request /auth;
#    ...
}

location = /auth {
    proxy_pass ...;
    proxy_pass_request_body off;
    proxy_set_header Content-Length "";
```

```
    proxy_set_header X-Original-URI $request_uri;
}
```

指令

auth_request

语法	`auth_request uri | off;`
默认值	`auth_request off;`
上下文	http, server, location

根据子请求的结果启用授权，并设置子请求将被发送的 URI。

auth_request_set

语法	`auth_request_set` *$variable value*;
默认值	—
上下文	http, server, location

在授权请求完成后，将请求变量设置为给定值。值可以包含来自授权请求的变量，例如 `$upstream_http_*`。

AutoIndex

该模块处理以斜杠字符 (/) 结尾的请求，并生成目录列表。通常，当*Index* 模块找不到索引文件时，请求会传递给 `AutoIndex` 模块。

配置示例

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

指令

autoindex

语法	`autoindex on | off;`
默认值	`autoindex off;`
上下文	http, server, location

启用或禁用目录列表输出。

autoindex_exact_size

语法	autoindex_exact_size on | off;
默认值	autoindex_exact_size on;
上下文	http, server, location

对于 HTML 格式，指定目录列表中是否输出精确的文件大小，或者将其四舍五入到千字节、兆字节和千兆字节。

autoindex_format

语法	autoindex_format html | xml | json | jsonp;
默认值	autoindex_format html;
上下文	http, server, location

设置目录列表的格式。

当使用 JSONP 格式时，回调函数的名称通过 `callback` 请求参数设置。如果参数缺失或为空值，则使用 JSON 格式。

XML 输出可以通过*XSLT* 模块进行转换。

输出格式

响应中的对象字段包含以下数据：

字段	描述
name	文件或目录名称
type	对象类型：file 或 directory
size	根据*autoindex_exact_size* 的对象大小；对于目录—0
mtime	Unix 时间格式的最后修改时间

HTML

```
<html>
<head>
    <title>Index of /files/</title>
</head>
<body>
    <h1>Index of /files/</h1>
    <hr>
    <pre>
            <a href="../">../</a>
            <a href="example.txt">example.txt</a>                12-Jun-2025 14:21    1234
            <a href="image.png">image.png</a>                    12-Jun-2025 14:21    4321
```

```
            </pre>
    <hr>
</body>
</html>
```

XML

```
<?xml version="1.0" encoding="UTF-8"?>
<listing>
<file>
    <name>example.txt</name>
    <type>file</type>
    <size>1234</size>
    <mtime>2025-06-12T14:21:00Z</mtime>
</file>
<file>
    <name>image.png</name>
    <type>file</type>
    <size>4321</size>
    <mtime>2025-06-12T14:21:00Z</mtime>
</file>
</listing>
```

JSON

```
[
{
    "name": "example.txt",
    "type": "file",
    "size": 1234,
    "mtime": "2025-06-12T14:21:00Z"
},
{
    "name": "image.png",
    "type": "file",
    "size": 4321,
    "mtime": "2025-06-12T14:21:00Z"
}
]
```

JSONP

```
callback([
{
    "name": "example.txt",
    "type": "file",
    "size": 1234,
    "mtime": "2025-06-12T14:21:00Z"
```

```
},
{
    "name": "image.png",
    "type": "file",
    "size": 4321,
    "mtime": "2025-06-12T14:21:00Z"
}
]);
```

autoindex_localtime

语法	autoindex_localtime on | off;
默认值	autoindex_localtime off;
上下文	http, server, location

对于 HTML 格式，指定目录列表中的时间是以本地时区还是以 UTC 输出。

Browser

该模块创建的变量值依赖于”User-Agent” 请求头字段的值。

变量

`$modern_browser`

如果识别为现代浏览器，则等于由*modern_browser_value* 指令设置的值；

`$ancient_browser`

如果识别为古老浏览器，则等于由*ancient_browser_value* 指令设置的值；

`$msie`

如果识别为任何版本的 MSIE 浏览器，则等于”1”。

配置示例

选择索引文件：

```
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;
```

为旧浏览器重定向：

```
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;
}
```

指令

ancient_browser

语法	`ancient_browser` *string* ...;
默认	—
上下文	http, server, location

如果在”User-Agent” 请求头字段中找到任何指定的子字符串，则该浏览器将被视为古老。特殊字符串” netscape4” 对应于正则表达式”^Mozilla/[1-4]”。

ancient_browser_value

语法	`ancient_browser_value` *string*;
默认	`ancient_browser_value 1;`
上下文	http, server, location

为*$ancient_browser* 变量设置值。

modern_browser

语法	`modern_browser` *browser version*; `modern_browser unlisted`;
默认	—
上下文	http, server, location

指定从哪个版本开始浏览器被视为现代。浏览器可以是以下任意一个：`msie`、`gecko`（基于 `Mozilla` 的浏览器）、`:samp:`opera`、`safari` 或 `konqueror`。

版本可以采用以下格式指定：X、X.X、X.X.X 或 X.X.X.X。每种格式的最大值分别为 4000、4000.99、4000.99.99 和 4000.99.99.99。

特殊值 `unlisted` 指定如果浏览器未在 *modern_browser* 和*ancient_browser* 指令中列出，则视为现代浏览器。否则，该浏览器将被视为古老。如果请求未在头中提供"User-Agent" 字段，则该浏览器被视为未列出。

modern_browser_value

语法	`modern_browser_value` *string*;
默认	`modern_browser_value 1;`
上下文	http, server, location

为*$modern_browser* 变量设置值。

Charset

该模块将指定的字符集添加到"Content-Type" 响应头字段。此外，该模块可以在某些限制条件下将数据从一种字符集转换为另一种字符集：

- 转换是单向进行的——从服务器到客户端，
- 只能转换单字节字符集，
- 或单字节字符集与 UTF-8 之间的转换。

配置示例

```
include        conf/koi-win;

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

指令

charset

语法	`charset` *charset* | `off`;
默认	`charset off;`
上下文	http, server, location, if in location

将指定的字符集添加到"Content-Type" 响应头字段。如果该字符集与*source_charset* 指令中指定的字符集不同，则会执行转换。

参数 `off` 取消将字符集添加到"Content-Type" 响应头字段。

字符集可以通过变量定义：

```
charset $charset;
```

在这种情况下，变量的所有可能值必须至少在配置中以*charset_map*、*charset* 或*source_charset* 指令的形式出现一次。对于 `utf-8`、`windows-1251` 和 `koi8-r` 字符集，只需在配置中包含 `conf/koi-win`、`conf/koi-utf` 和 `conf/win-utf` 文件。对于其他字符集，只需创建一个虚构的转换表即可，例如：

```
charset_map iso-8859-5 _ { }
```

此外，可以在" X-Accel-Charset" 响应头字段中设置字符集。可以使用*proxy_ignore_headers*、*fastcgi_ignore_headers*、*uwsgi_ignore_headers*、*scgi_ignore_headers* 和*grpc_ignore_headers* 指令禁用此功能。

charset_map

语法	`charset_map` *charset1 charset2* { ... }
默认	—
上下文	http

描述从一种字符集到另一种字符集的转换表。使用相同的数据构建反向转换表。字符编码以十六进制给出。在范围 80-FF 中缺失的字符用"*?*" 替代。从 UTF-8 转换时，在单字节字符集中缺失的字符用"*&#XXXX;*" 替代。

示例：

```
charset_map koi8-r windows-1251 {
    C0 FE ; # 小
    C1 E0 ; # 小
    C2 E1 ; # 小
    C3 F6 ; # 小
}
```

在描述到 UTF-8 的转换表时，UTF-8 字符集的代码应在第二列给出，例如：

```
charset_map koi8-r utf-8 {
    C0 D18E ; # 小
    C1 D0B0 ; # 小
    C2 D0B1 ; # 小
    C3 D186 ; # 小
}
```

从 koi8-r 到 windows-1251，以及从 koi8-r 和 windows-1251 到 utf-8 的完整转换表在分发文件 conf/koi-win、conf/koi-utf 和 conf/win-utf 中提供。

charset_types

语法	charset_types mime-*type* ...;
默认	charset_types text/html text/xml text/plain text/vnd.wap.wml application/javascript application/rss+xml;
上下文	http, server, location

使模块在处理指定 MIME 类型的响应时生效，除了 text/html 之外。特殊值 * 匹配任何 MIME 类型。

override_charset

语法	override_charset on | off;
默认	override_charset off;
上下文	http, server, location, if in location

确定是否应对从代理或 FastCGI/uwsgi/SCGI/gRPC 服务器接收到的响应进行转换，当这些响应已在”Content-Type” 响应头字段中携带字符集时。如果启用转换，则使用接收到的响应中指定的字符集作为源字符集。

备注

如果在子请求中收到响应，则始终会将响应字符集转换为主请求字符集，而不管 *override_charset* 指令的设置。

source_charset

语法	source_charset *charset*;
默认	—
上下文	http, server, location, if in location

定义响应的源字符集。如果该字符集与*charset* 指令中指定的字符集不同，则会执行转换。

DAV

该模块旨在通过 WebDAV 协议实现文件管理自动化。模块处理 HTTP 和 WebDAV 方法：PUT、DELETE、MKCOL、COPY 和 MOVE。

当 从源代码构建时，默认不包含该模块；需要通过 `--with-http_dav_module` 构建选项来启用。

在来自 我们的仓库的包和镜像中，该模块包含在构建中。

重要

需要其他 WebDAV 方法才能操作的 WebDAV 客户端将无法与此模块一起使用。

配置示例

```
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;
    }
}
```

指令

create_full_put_path

语法	`create_full_put_path on | off;`
默认值	`create_full_put_path off;`
上下文	http, server, location

WebDAV 规范仅允许在已存在的目录中创建文件。此指令允许创建所有需要的中间目录。

dav_access

语法	`dav_access` *users:permissions ...*;
默认值	`dav_access user:rw;`
上下文	http, server, location

为新创建的文件和目录设置访问权限，例如：

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

如果指定了任一组或所有访问权限，则可以省略用户权限：

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

dav_methods

语法	`dav_methods off` | *method* ...;
默认值	`dav_methods off;`
上下文	http, server, location

允许指定的 HTTP 和 WebDAV 方法。参数 off 拒绝所有此模块处理的方法。支持以下方法：PUT、DELETE、MKCOL、COPY 和 MOVE。

使用 PUT 方法上传的文件首先被写入临时文件，然后文件被重命名。从版本 0.8.9 开始，临时文件和持久存储可以放在不同的文件系统上。但请注意，在这种情况下，文件是在两个文件系统之间复制，而不是便宜的重命名操作。因此建议在任何给定的 `location` 中，保存的文件和由*client_body_temp_path* 指令设置的临时文件目录放在同一个文件系统中。

使用 PUT 方法创建文件时，可以通过在 `Date` 头字段中传递修改日期来指定。

min_delete_depth

语法	`min_delete_depth` *number*;
默认值	`min_delete_depth 0;`
上下文	http, server, location

允许 DELETE 方法删除文件，前提是请求路径中的元素数量不小于指定的数量。例如，指令

```
min_delete_depth 4;
```

允许删除以下请求的文件

```
/users/00/00/name
/users/00/00/name/pic.jpg
/users/00/00/page.html
```

并拒绝删除

```
/users/00/00
```

Docker

该模块提供基于 Docker 容器标签在*HTTP* 和*stream* 上下文中动态配置代理服务器组的功能。为了使该功能正常工作，必须在组中配置共享内存区域（参见*http* 和*stream* 的 `zone` 描述）。

> **备注**
>
> 该模块支持与 Docker 及其替代品（如 Podman）一起工作，只要它们实现了兼容的 API。

该模块通过 API 连接到 Docker 守护进程，与之交互的方法由*docker_endpoint* 指令指定。获取运行中容器列表后，Angie 分析它们是否存在合适的标签。如果容器描述包含带有端口的标签，则该容器的地址和端口，以及来自该容器其他标签的参数，将自动添加到 Angie 配置中相应的 `upstream` 块中。

> **备注**
>
> 同一个容器可以添加到多个 `upstream` 组；只需指定多组带有不同组名和端口的标签即可。
>
> 如果容器在不同端口上运行多个不同的服务，这特别有用；每个服务可以与自己的组相关联。

然后，该模块订阅容器生命周期事件，并开始在不重新加载 Angie 的情况下更新代理服务器配置：

- 当启动带有合适标签的容器时，其内部 IP 地址会被添加到指定的组中；
- 当停止或删除容器时，它会自动从组中移除；
- 当使用 `docker pause` 命令暂停容器时，服务器会被标记为 `down`，而使用 `docker unpause` 时则标记为 `up`。

配置示例

模块的指令始终位于 `http` 上下文中，但代理服务器组可以定义在 `http` 上下文和 `stream` 上下文中。

`http` 的配置示例：

```
http {

    # 连接选项示例：
    # docker_endpoint http://127.0.0.1:2375;
    # docker_endpoint https://127.0.0.1:2376;
    docker_endpoint unix:/var/run/docker.sock;

    # Docker 响应缓冲区最大大小（可选）
    # docker_max_object_size 128k;

    upstream u {

        zone z 1m; # 需要共享内存区域
    }
```

```
    server {

        listen 80;
        server_name example.com;

        location / {

            proxy_pass http://u;
        }
    }
}
```

stream 上下文中的类似配置：

```
http {

    # 连接选项示例：
    # docker_endpoint http://127.0.0.1:2375;
    # docker_endpoint https://127.0.0.1:2376;
    docker_endpoint unix:/var/run/docker.sock;

    # Docker 响应缓冲区最大大小（可选）
    # docker_max_object_size 128k;
}

stream {

    upstream u {

        zone z 1m;
    }

    server {

        listen 12345;
        proxy_pass u;
    }
}
```

当收到容器事件时，Angie 会查找形如 `angie.http.upstreams.<name>.port=<port>`（HTTP 上下文）或:samp:`angie.stream.upstreams.<name>.port=<port>`（stream 上下文）的标签。当存在标签时，容器在指定 Docker 网络中的地址（如果未指定:samp:`angie.network` 标签，则使用第一个可用网络）将被添加到相应的代理服务器组中。

如果容器停止或被移除，服务器将从组中移除；如果容器被暂停，服务器将被标记为 `down`。

带有 Angie 识别标签的 `docker-compose.yml` 文件片段：

```
services:
  myapp:
    image: myapp:latest
    labels:
      - "angie.http.upstreams.u.port=8080"
      - "angie.network=my_bridge"
      - "angie.http.upstreams.u.weight=2"
      - "angie.http.upstreams.u.max_conns=50"
      - "angie.http.upstreams.u.max_fails=3"
      - "angie.http.upstreams.u.fail_timeout=10s"
      - "angie.http.upstreams.u.backup=true"
```

标签

标签指定代理服务器组中的服务器参数，类似于 `server` 指令的参数：

标签	用途
`angie.(http|stream).upstreams.<name>.port=<port>` （必需）	Angie 将连接到的容器端口；容器本身被添加到名为 `<name>` 的组中。
`angie.network=<docker-network>`	从中获取容器 IP 地址的 Docker 网络名称。
`angie.(http|stream).upstreams.<name>.weight=<n>`	`weight` 参数的值。
`angie.(http|stream).upstreams.<name>.max_conns=<n>`	最大同时连接数 (`max_conns`)。
`angie.(http|stream).upstreams.<name>.max_fails=<n>`	失败尝试阈值 (`max_fails`)。
`angie.(http|stream).upstreams.<name>.fail_timeout=<t>`	计算失败尝试的间隔 (`fail_timeout`)。
`angie.(http|stream).upstreams.<name>.backup=true|false`	将服务器标记为 `backup`。
`angie.(http|stream).upstreams.<name>.sid=<string>`	为代理服务器设置自定义服务器标识符 (`sid`)。
`angie.(http|stream).upstreams.<name>.slow_start=<time>`	启用 `slow_start` 模式，并配置时间周期。

指令

docker_endpoint

语法	`docker_endpoint URL;`
默认值	—
上下文	http

指定连接到 Docker 守护进程的方法并启用容器事件跟踪。支持以下选项：

`unix:/var/run/docker.sock`	通过 Unix 套接字连接（例如，`/var/run/docker.sock`）。
`http://host:port`, `https://host:port`	通过 HTTP 或 HTTPS 连接到远程 Docker API。

连接可以通过*client* 上下文进一步配置，其中模块添加了两个命名 `location` 块：

- `@docker_events` 用于接收容器事件；
- `@docker_containers` 用于获取容器信息。

默认情况下,它们包含带有连接地址的*proxy_pass* 指令和其他几个最佳默认设置,可以向其添加来自*Proxy* 模块的其他设置。

如果指定了该指令，Angie 将使用指定方法打开与 Docker 的连接，请求运行中容器列表，分析它们的标签并处理所有后续容器事件，根据标签在代理服务器组中添加或删除服务器。

> **小技巧**
>
> 要通过 Unix 套接字访问 Docker 守护进程（`/var/run/docker.sock` 或其他），运行 Angie 的用户 必须对该套接字具有读写权限。

docker_max_object_size

语法	`docker_max_object_size <size>;`
默认值	`64k`
上下文	http

设置用于 Docker 请求的 JSON 响应和容器事件流的最大缓冲区大小。

- 对于常规请求（API 版本、容器列表、容器信息）：整个响应必须适合缓冲区，否则会发生错误。
- 对于容器事件，使用流式处理并重用缓冲区，这允许处理无限的事件流。

典型值 64k 足以处理大约 25 个容器。

当发生 Docker API 连接错误或响应处理错误时，模块会在特定时间间隔后自动重试。获取特定容器信息的最大重试尝试次数限制为两次 * 额外 * 尝试；之后，模块将停止对该容器的尝试。

Empty GIF

该模块发出单像素透明 GIF。

配置示例

```
location = /_.gif {
    empty_gif;
}
```

指令

empty_gif

语法	empty_gif;
默认	—
上下文	location

在包含的 location 中启用发出单像素透明 GIF。

FastCGI

该模块允许将请求传递给 FastCGI 服务器。

配置示例

```
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;
}
```

指令

fastcgi_bind

语法	`fastcgi_bind` 地址 [`transparent`] | `off`;
默认	—
上下文	http, server, location

使发往 FastCGI 服务器的外部连接来自指定的本地 IP 地址，可以选择性地指定端口。参数值可以包含变量。特殊值 `off` 取消从上一级配置层继承的 fastcgi_bind 指令的效果，这允许系统自动分配本地 IP 地址和端口。

`transparent` 参数允许发往 FastCGI 服务器的外部连接来自非本地 IP 地址，例如，客户端的真实 IP 地址：

```
fastcgi_bind $remote_addr transparent;
```

为了使此参数生效，Angie 工作进程通常需要以超级用户 权限运行。在 Linux 上，这不是必需的：如果指定了 `transparent` 参数，工作进程从主进程继承 *CAP_NET_RAW* 能力。

> **重要**
>
> 内核路由表也应配置为拦截来自 FastCGI 服务器的网络流量。

fastcgi_buffer_size

语法	`fastcgi_buffer_size` 大小;
默认	`fastcgi_buffer_size 4k|8k;`
上下文	http, server, location

设置用于读取来自 FastCGI 服务器的响应第一部分的缓冲区大小。该部分通常包含一个小的响应头。默认情况下，缓冲区大小等于一个内存页。根据平台，这可以是 4K 或 8K。然而，它可以设置得更小。

fastcgi_buffering

语法	`fastcgi_buffering on` | `off`;
默认	`fastcgi_buffering on;`
上下文	http, server, location

启用或禁用来自 FastCGI 服务器的响应缓冲。

`off`	Angie 尽快从 FastCGI 服务器接收响应，并将其保存到由*fastcgi_buffer_size* 和*fastcgi_buffers* 指令设置的缓冲区中。如果整个响应无法完全放入内存，则可以将其部分保存到磁盘上的临时文件 中。写入临时文件受*fastcgi_max_temp_file_size* 和*fastcgi_temp_file_write_size* 指令控制。
`on`	响应将同步传递给客户端，立即在接收到时传递。Angie 不会尝试从 FastCGI 服务器读取整个响应。Angie 一次可以从服务器接收的数据的最大大小由*fastcgi_buffer_size* 指令设置。

缓冲还可以通过在"X-Accel-Buffering" 响应头字段中传递"yes" 或"no" 来启用或禁用。此功能可以使用*fastcgi_ignore_headers* 指令禁用。

fastcgi_buffers

语法	`fastcgi_buffers` 数量 大小;
默认	`fastcgi_buffers 8 4k|8k;`
上下文	http, server, location

设置用于读取来自 FastCGI 服务器的响应的缓冲区数量和大小，针对单个连接。

默认情况下，缓冲区大小等于一个内存页。根据平台，这可以是 4K 或 8K。

fastcgi_busy_buffers_size

语法	`fastcgi_busy_buffers_size` 大小;
默认	`fastcgi_busy_buffers_size 8k | 16k;`
上下文	http, server, location

当启用缓冲 来自 FastCGI 服务器的响应时，限制在响应尚未完全读取时可以忙于向客户端发送响应的缓冲区的总大小。与此同时，其余的缓冲区可以用于读取响应，并在需要时，将部分响应缓冲到临时文件中。默认情况下，大小受*fastcgi_buffer_size* 和*fastcgi_buffers* 指令设置的两个缓冲区的大小限制。

fastcgi_cache

语法	`fastcgi_cache` *zone* | `off;`
默认	`fastcgi_cache off;`
上下文	http, server, location

定义用于缓存的共享内存区域。可以在多个地方使用相同的区域。参数值可以包含变量。`off` 参数禁用从上一级配置层继承的缓存。

fastcgi_cache_background_update

语法	fastcgi_cache_background_update on | off;
默认	fastcgi_cache_background_update off;
上下文	http, server, location

允许在返回过期缓存项的同时启动后台子请求以更新过期的缓存项。请注意，在更新缓存时，有必要允许 使用过期的缓存响应。

fastcgi_cache_bypass

语法	fastcgi_cache_bypass 字符串...;
默认	—
上下文	http, server, location

定义在何种条件下响应不会从缓存中提取。如果字符串参数的至少一个值不为空且不等于"0"，则响应将不会从缓存中提取：

```
fastcgi_cache_bypass $cookie_nocache $arg_nocache$arg_comment;
fastcgi_cache_bypass $http_pragma    $http_authorization;
```

可以与*fastcgi_no_cache* 指令一起使用。

fastcgi_cache_key

语法	fastcgi_cache_key 字符串;
默认	—
上下文	http, server, location

定义缓存的键，例如

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

fastcgi_cache_lock

语法	fastcgi_cache_lock on | off;
默认	fastcgi_cache_lock off;
上下文	http, server, location

启用时，只有一个请求可以通过将请求传递给 FastCGI 服务器来填充由*fastcgi_cache_key* 指令标识的新缓存元素。其他同一缓存元素的请求将等待响应在缓存中出现或该元素的缓存锁被释放，等待时间由*fastcgi_cache_lock_timeout* 指令设置。

fastcgi_cache_lock_age

语法	`fastcgi_cache_lock_age` 时间;
默认	`fastcgi_cache_lock_age 5s;`
上下文	http, server, location

如果发送到 FastCGI 服务器以填充新缓存条目的最后一个请求在指定时间内未完成，则可以向 FastCGI 服务器发送另一个请求。

fastcgi_cache_lock_timeout

语法	`fastcgi_cache_lock_timeout` 时间;
默认	`fastcgi_cache_lock_timeout 5s;`
上下文	http, server, location

为*fastcgi_cache_lock* 设置超时。当时间到期时，请求将被传递给 FastCGI 服务器，但响应不会被缓存。

fastcgi_cache_max_range_offset

语法	`fastcgi_cache_max_range_offset` 数字;
默认	—
上下文	http, server, location

为字节范围请求设置字节偏移量。如果范围超出偏移量，范围请求将被传递给 FastCGI 服务器，并且响应不会被缓存。

fastcgi_cache_methods

语法	`fastcgi_cache_methods GET | HEAD | POST ...;`
默认	`fastcgi_cache_methods GET HEAD;`
上下文	http, server, location

如果客户端请求方法在该指令中列出，则响应将被缓存。“GET”和“HEAD”方法始终被添加到列表中，尽管建议明确指定它们。另见*fastcgi_no_cache* 指令。

fastcgi_cache_min_uses

语法	`fastcgi_cache_min_uses` 数字;
默认	`fastcgi_cache_min_uses 1;`
上下文	http, server, location

设置响应被缓存后请求的数量。

fastcgi_cache_path

语法	`fastcgi_cache_path` *path* [`levels=`*levels*] [`use_temp_path=on` | `off`] `keys_zone=`*name:size* [`inactive=`*time*] [`max_size=`*size*] [`min_free=`*size*] [`manager_files=`*number*] [`manager_sleep=`*time*] [`manager_threshold=`*time*] [`loader_files=`*number*] [`loader_sleep=`*time*] [`loader_threshold=`*time*];
默认	—
上下文	http, server, location

设置缓存的路径和其他参数。缓存数据存储在文件中。缓存中的键和文件名是对代理 URL 应用 MD5 函数的结果。

`levels` 参数定义了缓存的层级级别：从 1 到 3，每个级别接受值 1 或 2。例如，在以下配置中

```
fastcgi_cache_path /data/angie/cache levels=1:2 keys_zone=one:10m;
```

缓存中的文件名将如下所示：

```
/data/angie/cache/c/29/b7f54b2df7773722d382f4809d65029c
```

缓存的响应首先写入临时文件，然后重命名。临时文件和缓存可以放在不同的文件系统上。然而，请注意，在这种情况下，文件是在两个文件系统之间复制，而不是便宜的重命名操作。因此，建议在任何给定位置的缓存和临时文件夹都放在同一文件系统上。

临时文件的目录是基于 `use_temp_path` 参数设置的。

`on`	如果省略此参数或将其设置为“on”，则将使用为给定 *location* 设置的*fastcgi_temp_path* 指令指定的目录。
`off`	临时文件将直接放在缓存目录中。

此外，所有活动的键和关于数据的信息都存储在共享内存区域中，其名称和大小由 `keys_zone` 参数配置。一个兆字节的区域可以存储大约 8000 个键。

在 `inactive` 参数指定的时间内未被访问的缓存数据将从缓存中移除，无论它们的新鲜程度如何。

默认情况下，inactive 设置为 10 分钟。

一个特殊的 **缓存管理器**进程监控最大缓存大小，以及缓存文件系统上的最小空闲空间。当大小超过或空闲空间不足时，它将删除最近最少使用的数据。数据是分批删除的。

max_size	最大缓存大小
min_free	缓存文件系统上的最小空闲空间
manager_files	限制在一个迭代过程中要删除的项目数量 默认情况下，100。
manager_threshol	限制一个迭代的持续时间 默认情况下，200 毫秒
manager_sleep	配置交互之间的暂停 默认情况下，50 毫秒

在 Angie 启动后一分钟，特殊的 **缓存加载器**进程被激活。它将存储在文件系统上的先前缓存数据的信息加载到缓存区域中。加载也是分批完成的。

loader_files	限制在一个迭代过程中要加载的项目数量 默认情况下，100
loader_threshold	限制一个迭代的持续时间 默认情况下，200 毫秒
loader_sleep	配置交互之间的暂停 默认情况下，50 毫秒

fastcgi_cache_revalidate

语法	fastcgi_cache_revalidate on | off;
默认	fastcgi_cache_revalidate off;
上下文	http, server, location

启用使用"If-Modified-Since" 和"If-None-Match" 头字段的条件请求来重新验证过期的缓存项。

fastcgi_cache_use_stale

语法	fastcgi_cache_use_stale error | timeout | invalid_header | updating | http_500 | http_503 | http_403 | http_429 | off ...;
默认	fastcgi_cache_use_stale off;
上下文	http, server, location

确定在与 FastCGI 服务器通信时发生错误的情况下，可以在何种情况下使用过期的缓存响应。指令的参数与*fastcgi_next_upstream* 指令的参数相匹配。

error	如果无法选择处理请求的 FastCGI 服务器，则允许使用过期的缓存响应。
updating	附加参数，允许使用过期的缓存响应，如果它正在被更新。这可以在更新缓存数据时最小化对 FastCGI 服务器的访问次数。

还可以通过在响应头中直接启用使用过期的缓存响应，在响应变为过期后的指定秒数内。

- stale-while-revalidate "Cache-Control" 头字段的扩展允许在当前更新时使用过期的缓存响应。
- stale-if-error "Cache-Control" 头字段的扩展允许在发生错误时使用过期的缓存响应。

备注

这优先级低于使用指令参数。

为了在填充新的缓存元素时最小化对 FastCGI 服务器的访问次数，可以使用*fastcgi_cache_lock* 指令。

fastcgi_cache_valid

语法	`fastcgi_cache_valid` [*code* ...] *time*;
默认	—
上下文	http, server, location

为不同的响应代码设置缓存时间。例如，以下指令

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

为响应代码 200 和 302 设置 10 分钟的缓存，为响应代码 404 设置 1 分钟的缓存。

如果只指定缓存时间

```
fastcgi_cache_valid 5m;
```

则仅缓存 200、301 和 302 响应。

此外，可以指定 `any` 参数以缓存所有响应：

```
fastcgi_cache_valid 200 302 10m;
fastcgi_cache_valid 301      1h;
fastcgi_cache_valid any      1m;
```

备注

缓存的参数也可以直接在响应头中设置。这优先于使用指令设置缓存时间。

- "X-Accel-Expires" 头字段以秒为单位设置响应的缓存时间。零值禁用响应的缓存。如果值以 @ 前缀开头，它设置自 Epoch 起的绝对时间（以秒为单位），在此之前响应可以被缓存。
- 如果头中不包括"X-Accel-Expires" 字段，则可以在"Expires" 或"Cache-Control" 头字段中设置缓存参数。

- 如果头中包括"Set-Cookie" 字段，则该响应将不会被缓存。
- 如果头中包括"Vary" 字段且值为"*"，则该响应将不会被缓存。如果头中包含"Vary" 字段且值为其他值，则该响应将被缓存，并考虑相应的请求头字段。

可以使用*fastcgi_ignore_headers* 指令禁用对这些响应头字段之一或多个的处理。

fastcgi_catch_stderr

语法	`fastcgi_catch_stderr` *string*;
默认	—
上下文	http, server, location

设置一个字符串，用于在从 FastCGI 服务器接收的响应的错误流中进行搜索。如果找到该字符串，则认为 FastCGI 服务器返回了无效响应。这允许在 Angie 中处理应用程序错误，例如：

```
location /php/ {
    fastcgi_pass backend:9000;
    ...
    fastcgi_catch_stderr "PHP Fatal error";
    fastcgi_next_upstream error timeout invalid_header;
}
```

fastcgi_connect_timeout

语法	`fastcgi_connect_timeout` *time*;
默认	`fastcgi_connect_timeout 60s;`
上下文	http, server, location

定义与 FastCGI 服务器建立连接的超时时间。需要注意的是，这个超时时间通常不能超过 75 秒。

fastcgi_connection_drop

语法	`fastcgi_connection_drop` *time* | `on` | `off`;
默认	`fastcgi_connection_drop off;`
上下文	http, server, location

启用在通过*reresolve* 进程或*API* 命令 `DELETE` 将代理服务器移除或标记为永久不可用后，终止所有与该代理服务器的连接。

当为客户端或代理服务器处理下一个读取或写入事件时，连接将被终止。

设置 *time* 启用连接终止超时；设置 `on` 时，连接立即被丢弃。

fastcgi_force_ranges

语法	`fastcgi_force_ranges on | off;`
默认	`fastcgi_force_ranges off;`
上下文	http, server, location

启用对 FastCGI 服务器的缓存和未缓存响应的字节范围支持，而不管这些响应中的"Accept-Ranges" 字段。

fastcgi_hide_header

语法	`fastcgi_hide_header` *field*;
默认	—
上下文	http, server, location

默认情况下，Angie 不会将 FastCGI 服务器响应中的 `Status` 和 `X-Accel-...` 头字段传递给客户端。*fastcgi_hide_header* 指令设置了将不被传递的其他字段。如果相反，需要允许字段的传递，可以使用*fastcgi_pass_header* 指令。

fastcgi_ignore_client_abort

语法	`fastcgi_ignore_client_abort on | off;`
默认值	`fastcgi_ignore_client_abort off;`
上下文	http, server, location

确定当客户端在未等待响应的情况下关闭连接时，是否应关闭与 FastCGI 服务器的连接。

fastcgi_ignore_headers

语法	`fastcgi_ignore_headers` *field*;
默认值	—
上下文	http, server, location

禁用对来自 FastCGI 服务器的某些响应头字段的处理。可以忽略以下字段：“X-Accel-Redirect”、“X-Accel-Expires”、“X-Accel-Limit-Rate”、“X-Accel-Buffering”、“X-Accel-Charset”、“Expires”、“Cache-Control”、“Set-Cookie” 和 “Vary”。

如果未禁用，对这些头字段的处理会产生以下影响：

- “X-Accel-Expires”、“Expires”、“Cache-Control”、“Set-Cookie” 和 “Vary” 设置响应缓存的参数；
- “X-Accel-Redirect” 执行对指定 URI 的内部 重定向；
- “X-Accel-Limit-Rate” 设置对客户端响应的速率限制；

- “X-Accel-Buffering”启用或禁用响应的缓冲；
- “X-Accel-Charset”设置响应所需的字符集。

fastcgi_index

语法	`fastcgi_index` *name*;
默认值	—
上下文	http, server, location

设置一个文件名，该文件名将在以斜杠结尾的 URI 后附加，位于*$fastcgi_script_name* 变量的值中。例如，使用以下设置

```
fastcgi_index index.php;
fastcgi_param SCRIPT_FILENAME /home/www/scripts/php$fastcgi_script_name;
```

对于 `/page.php` 请求，SCRIPT_FILENAME 参数将等于 `/home/www/scripts/php/page.php`，

而对于 `/` 请求，它将等于 - `/home/www/scripts/php/index.php`。

fastcgi_intercept_errors

语法	`fastcgi_intercept_errors on | off`;
默认值	`fastcgi_intercept_errors off`;
上下文	http, server, location

确定 FastCGI 服务器响应的状态码大于或等于 300 时，是否应将其传递给客户端，或被拦截并重定向到 Angie 进行处理，使用*error_page* 指令。

fastcgi_keep_conn

语法	`fastcgi_keep_conn on | off`;
默认值	`fastcgi_keep_conn off`;
上下文	http, server, location

默认情况下，FastCGI 服务器在发送响应后会立即关闭连接。然而，当该指令设置为 `on` 时，Angie 将指示 FastCGI 服务器保持连接打开。这在保持与 FastCGI 服务器的*keepalive* 连接正常工作时尤其重要。

fastcgi_limit_rate

语法	`fastcgi_limit_rate` *rate*;
默认值	`fastcgi_limit_rate 0`;
上下文	http, server, location

限制从 FastCGI 服务器读取响应的速度。*rate* 以每秒字节数指定，并且可以包含变量。

0	禁用速率限制

备注

限制是针对每个请求设置的，因此如果 Angie 同时打开两个与 FastCGI 服务器的连接，则整体速率将是指定限制的两倍。限制仅在启用了对来自 FastCGI 服务器的响应的缓冲 时有效。

fastcgi_max_temp_file_size

语法	`fastcgi_max_temp_file_size` *size*;
默认值	`fastcgi_max_temp_file_size 1024m;`
上下文	http, server, location

当启用对来自 FastCGI 服务器的响应的缓冲 时，如果整个响应不能放入由*fastcgi_buffer_size* 和*fastcgi_buffers* 指令设置的缓冲区中，则可以将响应的一部分保存到临时文件中。该指令设置临时文件的最大大小。写入临时文件的数据大小由*fastcgi_temp_file_write_size* 指令设置。

0	禁用对临时文件的响应缓冲

备注

此限制不适用于将要缓存 或存储在磁盘 上的响应。

fastcgi_next_upstream

语法	`fastcgi_next_upstream error` | `timeout` | `invalid_header` | `http_500` | `http_503` | `http_403` | `http_404` | `http_429` | `non_idempotent` | `off` ...;
默认值	`fastcgi_next_upstream error timeout;`
上下文	http, server, location

指定在何种情况下请求应传递给下一个服务器：

error	在与服务器建立连接、将请求传递给服务器或读取响应头时发生错误；
timeout	在与服务器建立连接、将请求传递给服务器或读取响应头时发生超时；
invalid_header	服务器返回了空的或无效的响应；
http_500	服务器返回了状态码 500 的响应；
http_503	服务器返回了状态码 503 的响应；
http_403	服务器返回了状态码 403 的响应；
http_404	服务器返回了状态码 404 的响应；
http_429	服务器返回了状态码 429 的响应；
non_idempotent	通常，请求使用 non-idempotent 方法 (POST, LOCK, PATCH) 时，如果请求已发送给上游服务器，则不会传递给下一个服务器；启用此选项明确允许重试此类请求；
off	禁用将请求传递给下一个服务器。

备注

应当记住，只有在尚未向客户端发送任何内容的情况下，才能将请求传递给下一个服务器。也就是说，如果在响应传输过程中发生错误或超时，则无法修复此问题。

该指令还定义了与服务器通信被认为是不成功尝试。

error timeout invalid_header	始终被视为不成功的尝试，即使在指令中未指定
http_500 http_503 http_429	仅在指令中指定时才被视为不成功的尝试
http_403 http_404	永远不会被视为不成功的尝试

将请求传递给下一个服务器的次数可以通过尝试次数 和时间 限制。

fastcgi_next_upstream_timeout

语法	fastcgi_next_upstream_timeout *time*;
默认值	fastcgi_next_upstream_timeout 0;
上下文	http, server, location

限制请求可以传递给下一个服务器 的时间。

0	关闭此限制

fastcgi_next_upstream_tries

语法	`fastcgi_next_upstream_tries` *number*;
默认值	`fastcgi_next_upstream_tries 0;`
上下文	http, server, location

限制将请求传递给下一个服务器 的可能尝试次数。

`0`	关闭此限制

fastcgi_no_cache

语法	`fastcgi_no_cache` *string* ...;
默认值	—
上下文	http, server, location

定义在什么条件下响应将不会被保存到缓存。如果字符串参数中的至少一个值不为空且不等于“0”，则响应将不会被保存：

```
fastcgi_no_cache $cookie_nocache $arg_nocache$arg_comment;
fastcgi_no_cache $http_pragma    $http_authorization;
```

可以与*fastcgi_cache_bypass* 指令一起使用。

fastcgi_param

语法	`fastcgi_param` *parameter value* [`if_not_empty`];
默认值	—
上下文	http, server, location

设置应传递给 FastCGI 服务器的参数。值可以包含文本、变量及其组合。如果当前级别未定义*fastcgi_param* 指令，则这些指令将从上一个配置级别继承。

以下示例显示了 PHP 的最小所需设置：

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

SCRIPT_FILENAME 参数用于 PHP 确定脚本名称，QUERY_STRING 参数用于传递请求参数。

对于处理 POST 请求的脚本，还需要以下三个参数：

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

如果 PHP 是使用 *--enable-force-cgi-redirect* 配置参数构建的，则还应传递 REDIRECT_STATUS 参数，值为“200”：

```
fastcgi_param REDIRECT_STATUS 200;
```

如果指令与 `if_not_empty` 一起指定，则仅在其值不为空时，参数才会传递给服务器：

```
fastcgi_param HTTPS           $https if_not_empty;
```

fastcgi_pass

语法	`fastcgi_pass` *address*;
默认值	—
上下文	location, 如果在 location 中

设置 FastCGI 服务器的地址。地址可以指定为域名或 IP 地址，以及一个端口：

```
fastcgi_pass localhost:9000;
```

或作为 UNIX 域套接字路径：

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

如果一个域名解析为多个地址，则所有地址将以轮询的方式使用。此外，地址可以指定为服务器组。如果使用组，则不能与其一起指定端口；相反，单独为组内每个服务器指定端口。

参数值可以包含变量。在这种情况下，如果地址指定为域名，则会在描述的服务器组 中搜索该名称，如果未找到，则使用*resolver* 确定。

fastcgi_pass_header

语法	`fastcgi_pass_header` *field*;
默认值	—
上下文	http, server, location

允许从 FastCGI 服务器将否则禁用 的头字段传递给客户端。

fastcgi_pass_request_body

语法	`fastcgi_pass_request_body on | off;`
默认值	—
上下文	http, server, location

指示是否将原始请求体传递给 FastCGI 服务器。另请参阅*fastcgi_pass_request_headers* 指令。

fastcgi_pass_request_headers

语法	`fastcgi_pass_request_headers on | off;`
默认值	—
上下文	http, server, location

指示是否将原始请求的头字段传递给 FastCGI 服务器。另请参阅*fastcgi_pass_request_body* 指令。

fastcgi_read_timeout

语法	`fastcgi_read_timeout` *time*;
默认值	`fastcgi_read_timeout 60s;`
上下文	http, server, location

定义从 FastCGI 服务器读取响应的超时时间。超时仅在两次连续读取操作之间设置，而不是针对整个响应的传输。如果 FastCGI 服务器在此时间内没有传输任何内容，则连接会关闭。

fastcgi_request_buffering

语法	`fastcgi_request_buffering on | off;`
默认值	`fastcgi_request_buffering on;`
上下文	http, server, location

启用或禁用客户端请求体的缓冲。

`on`	在将请求发送到 FastCGI 服务器之前，整个请求体会从客户端读取。
`off`	请求体会立即发送到 FastCGI 服务器，正如它被接收的一样。在这种情况下，如果 Angie 已经开始发送请求体，则请求无法传递给下一个服务器。

fastcgi_send_lowat

语法	`fastcgi_send_lowat` *size*;
默认值	`fastcgi_send_lowat 0;`
上下文	http, server, location

如果指令设置为非零值，Angie 将尝试通过使用*kqueue* 方法的 *NOTE_LOWAT* 标志或指定大小的 *SO_SNDLOWAT* 套接字选项，来最小化向 FastCGI 服务器的出站连接上的发送操作数量。

备注

此指令在 Linux、Solaris 和 Windows 上被忽略。

fastcgi_send_timeout

语法	`fastcgi_send_timeout` *time*;
默认值	`fastcgi_send_timeout 60s;`
上下文	http, server, location

设置向 FastCGI 服务器传输请求的超时时间。超时仅在两次连续写操作之间设置，而不是针对整个请求的传输。如果 FastCGI 服务器在此时间内没有接收到任何内容，则连接会关闭。

fastcgi_socket_keepalive

语法	`fastcgi_socket_keepalive on | off;`
默认值	`fastcgi_socket_keepalive off;`
上下文	http, server, location

配置与 FastCGI 服务器的出站连接的"TCP keepalive" 行为。

`""`	默认情况下，套接字使用操作系统的设置。
`on`	为套接字打开 *SO_KEEPALIVE* 套接字选项。

fastcgi_split_path_info

语法	`fastcgi_split_path_info` *regex*;
默认值	—
上下文	location

定义一个正则表达式，该表达式捕获*$fastcgi_path_info* 变量的值。正则表达式应该有两个捕获：第一个成为*$fastcgi_script_name* 变量的值，第二个成为*$fastcgi_path_info* 变量的值。例如，使用以下设置

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

并且请求 `/show.php/article/0001`，则 SCRIPT_FILENAME 参数将等于 `/path/to/php/show.php`，而 PATH_INFO 参数将等于 `/article/0001`。

fastcgi_store

语法	`fastcgi_store on | off | string;`
默认值	`fastcgi_store off;`
上下文	http, server, location

启用将文件保存到磁盘。

`on`	以与*alias* 或*root* 指令相对应的路径保存文件。
`off`	禁用文件保存。

此外，文件名可以使用包含变量的字符串显式设置：

```
fastcgi_store /data/www$original_uri;
```

文件的修改时间根据收到的"Last-Modified" 响应头字段设置。响应首先写入临时文件，然后重命名。临时文件和持久存储可以放在不同的文件系统上。然而，请注意，在这种情况下，文件在两个文件系统之间复制，而不是进行便宜的重命名操作。因此，建议对于任何给定的 *location*，保存的文件和由*fastcgi_temp_path* 指令设置的临时文件目录放在同一文件系统上。

此指令可用于创建静态不可更改文件的本地副本，例如：

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

location /fetch/ {
    internal;

    fastcgi_pass        backend:9000;
    ...

    fastcgi_store       on;
```

```
    fastcgi_store_access user:rw group:rw all:r;
    fastcgi_temp_path    /data/temp;

    alias                /data/www/;
}
```

fastcgi_store_access

语法	`fastcgi_store_access` users:permissions ...;
默认值	`fastcgi_store_access user:rw;`
上下文	http, server, location

设置新创建文件和目录的访问权限，例如：

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

如果指定了任何组或所有访问权限，则可以省略用户权限：

```
fastcgi_store_access group:rw all:r;
```

fastcgi_temp_file_write_size

语法	`fastcgi_temp_file_write_size` *size*;
默认值	`fastcgi_temp_file_write_size 8k|16k;`
上下文	http, server, location

限制在启用从 FastCGI 服务器缓冲响应到临时文件时一次写入临时文件的数据大小。默认情况下，大小受*fastcgi_buffer_size* 和*fastcgi_buffers* 指令设置的两个缓冲区限制。临时文件的最大大小由*fastcgi_max_temp_file_size* 指令设置。

fastcgi_temp_path

语法	`fastcgi_temp_path` *path* [*level1* [*level2* [*level3*]]]‘;
默认值	`fastcgi_temp_path fastcgi_temp;` (路径依赖于 `--http-fastcgi-temp-path` 构建选项)
上下文	http, server, location

定义一个目录，用于存储从 FastCGI 服务器接收到的临时文件。可以在指定目录下使用最多三级子目录结构。例如，在以下配置中

```
fastcgi_temp_path /spool/angie/fastcgi_temp 1 2;
```

临时文件可能看起来像这样：

```
/spool/angie/fastcgi_temp/7/45/00000123457
```

另请参见*fastcgi_cache_path* 指令的 *use_temp_path* 参数。

传递给 FastCGI 服务器的参数

HTTP 请求头字段作为参数传递给 FastCGI 服务器。在作为 FastCGI 服务器运行的应用程序和脚本中，这些参数通常作为环境变量提供。例如，“User-Agent”头字段作为 HTTP_USER_AGENT 参数传递。除了 HTTP 请求头字段，还可以使用*fastcgi_param* 指令传递任意参数。

内置变量

http_fastcgi 模块支持可以通过*fastcgi_param* 指令设置参数的内置变量：

$fastcgi_script_name

请求 URI，或者如果 URI 以斜杠结尾，则请求 URI 加上由*fastcgi_index* 指令配置的索引文件名。此变量可用于设置确定 PHP 中脚本名称的 SCRIPT_FILENAME 和 PATH_TRANSLATED 参数。例如，对于以下指令的 `/info/` 请求

```
fastcgi_index index.php;
fastcgi_param SCRIPT_FILENAME /home/www/scripts/php$fastcgi_script_name;
```

SCRIPT_FILENAME 参数将等于 `/home/www/scripts/php/info/index.php`。

使用*fastcgi_split_path_info* 指令时，变量 *$fastcgi_script_name* 等于指令所捕获的第一个值。

$fastcgi_path_info

由*fastcgi_split_path_info* 指令捕获的第二个值。此变量可用于设置 PATH_INFO 参数。

FLV

该模块为 Flash 视频（FLV）文件提供伪流式传输的服务器端支持。

它专门处理请求 URI 的查询字符串中的 start 参数，通过从请求的字节偏移量开始发送文件内容，并附加 FLV 头。

当 从源代码构建时，该模块默认未构建；需要使用 `--with-http_flv_module` 构建选项来启用。

在来自 我们的仓库的软件包和镜像中，该模块已包含在构建中。

配置示例

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

指令

flv

语法	flv;
默认值	—
上下文	location

在周围的 location 中开启模块处理。

Geo

该模块根据客户端 IP 地址创建具有不同值的变量。

配置示例

```
geo $geo {
    default         0;

    127.0.0.1       2;
    192.168.1.0/24  1;
    10.1.0.0/16     1;

    ::1             2;
    2001:0db8::/32  1;
}
```

指令

geo

Syntax	geo [*$address*] *$variable* { ... }
Default	—
Context	http

描述了指定变量的值如何依赖于客户端 IP 地址。默认情况下，地址从*$remote_addr* 变量中获取，但也可以从其他变量中获取，例如：

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

备注

由于变量仅在使用时才会被计算，即使声明了大量的 geo 变量，也不会对请求处理造成额外开销。

如果变量的值不是有效的 IP 地址，则会使用”255.255.255.255” 地址。

地址可以用 CIDR 表示法指定为前缀（包括单个地址）或作为范围。

还支持以下特殊参数：

`delete`	删除指定的网络
`default`	如果客户端地址不匹配任何指定的地址，则为变量设置的值。当地址以 CIDR 表示法指定时，可以使用 `0.0.0.0/0` 和 `::/0` 代替 `default`。如果未指定 `default`，则默认值将为空字符串
`include`	包含一个带有地址和值的文件。可以有多个包含。
`proxy`	定义受信任的地址。当请求来自受信任的地址时，将使用 `X-Forwarded-For` 请求头字段中的地址。与常规地址不同，受信任的地址是按顺序检查的。
`proxy_recursive`	启用递归地址搜索。如果禁用递归搜索，则将使用 `X-Forwarded-For` 中的最后一个地址，而不是与受信任地址匹配的原始客户端地址。如果启用递归搜索，则将使用 `X-Forwarded-For` 中的最后一个非受信任地址，而不是与受信任地址匹配的原始客户端地址。
`ranges`	表示地址是作为范围指定的。此参数应为第一个。为加快地理数据库的加载，地址应按升序排列。

示例：

```
geo $country {
    default         ZZ;
    include         conf/geo.conf;
    delete          127.0.0.0/16;
    proxy           192.168.100.0/24;
    proxy           2001:0db8::/32;

    127.0.0.0/24    US;
    127.0.0.1/32    RU;
    10.1.0.0/16     RU;
    192.168.1.0/24 UK;
}
```

`conf/geo.conf` 文件可能包含以下行：

```
10.2.0.0/16     RU;
192.168.2.0/24 RU;
```

使用最具体的匹配值。例如，对于 `127.0.0.1` 地址，将选择值 `RU`，而不是 `US`。

示例范围描述：

```
geo $country {
    ranges;
    default                   ZZ;
    127.0.0.0-127.0.0.0       US;
```

```
    127.0.0.1-127.0.0.1       RU;
    127.0.0.2-127.0.0.255     US;
    10.1.0.0-10.1.255.255     RU;
    192.168.1.0-192.168.1.255 UK;
}
```

GeoIP

根据客户端 IP 地址创建变量并赋值，使用预编译的 MaxMind 数据库或其对应版本。

使用支持 IPv6 的数据库时，IPv4 地址将作为映射到 IPv6 的地址进行查找。

当 从源代码构建时，默认不构建此模块；应通过 `--with-http_geoip_module` 构建选项进行启用。

重要

此模块需要 MaxMind GeoIP 数据库或类似的 MaxMind GeoLite2。

配置示例

```
http {
    geoip_country         GeoIP.dat;
    geoip_city            GeoLiteCity.dat;
    geoip_proxy           192.168.100.0/24;
    geoip_proxy           2001:0db8::/32;
    geoip_proxy_recursive on;
    ...
```

指令

geoip_country

语法	`geoip_country` *file*;
默认	—
上下文	http

指定用于根据客户端 IP 地址确定国家的数据库。使用此数据库时，可用以下变量：

`$geoip_country_c`	两位的国家代码，例如”RU”，”US”。
`$geoip_country_c`	三位的国家代码，例如”RUS”，”USA”。
`$geoip_country_n`	国家名称，例如”Russian Federation”，”United States”。

geoip_city

语法	`geoip_city` *file*;
默认	—
上下文	http

指定用于根据客户端 IP 地址确定国家、地区和城市的数据库。使用此数据库时，可用以下变量：

`$geoip_city_cont`	两位的大陆代码，例如”EU”，”NA”。
`$geoip_city_coun`	两位的国家代码，例如”RU”，”US”。
`$geoip_city_coun`	三位的国家代码，例如”RUS”，”USA”。
`$geoip_city_coun`	国家名称，例如”Russian Federation”，”United States”。
`$geoip_dma_code`	美国的 DMA 区域代码（也称为”metro code”），根据 Google AdWords API 中的 地理定位。
`$geoip_latitude`	纬度。
`$geoip_longitude`	经度。
`$geoip_region`	两位的国家地区代码（地区、领地、州、省、联邦土地等），例如”48”，”DC”。
`$geoip_region_na`	国家地区名称（地区、领地、州、省、联邦土地等），例如”Moscow City”，”District of Columbia”。
`$geoip_city`	城市名称，例如”Moscow”，”Washington”。
`$geoip_postal_co`	邮政编码。

geoip_org

语法	`geoip_org` *file*;
默认	—
上下文	http

指定用于根据客户端 IP 地址确定组织的数据库。使用此数据库时，可用以下变量：

`$geoip_org`	组织名称，例如”The University of Melbourne”。

geoip_proxy

语法	`geoip_proxy` *address* | *CIDR* | *unix:*;
默认	—
上下文	http

定义可信任地址。当请求来自可信任地址时，将使用 `X-Forwarded-For` 请求头字段中的地址。

geoip_proxy_recursive

语法	geoip_proxy_recursive on | off;
默认	geoip_proxy_recursive off;
上下文	http

如果禁用递归搜索，则将使用 X-Forwarded-For 中发送的最后一个地址，而不是与可信任地址匹配的原始客户端地址。如果启用递归搜索，则将使用 X-Forwarded-For 中发送的最后一个非可信任地址，而不是与可信任地址匹配的原始客户端地址。

gRPC

允许将请求传递给 gRPC 服务器。该模块需要*HTTP/2*。

配置示例

```
server {
    listen 9000;

    http2 on;

    location / {
        grpc_pass 127.0.0.1:9000;
    }
}
```

指令

grpc_bind

语法	grpc_bind *address* [transparent] | off;
默认值	—
上下文	http, server, location

使发往 gRPC 服务器的外部连接从指定的本地 IP 地址发起，且可选端口。参数值可以包含变量。特殊值 off 取消从上一个配置级别继承的 *grpc_bind* 指令的效果，从而允许系统自动分配本地 IP 地址和端口。

transparent 参数允许从非本地 IP 地址发起对 gRPC 服务器的外部连接，例如，从客户端的真实 IP 地址：

```
grpc_bind $remote_addr transparent;
```

为了使此参数生效，通常需要以*superuser* 权限运行 Angie 工作进程。在 Linux 上，如果指定了 transparent 参数，则工作进程会从主进程继承 *CAP_NET_RAW* 能力，因此不需要特权。

重要

必须配置内核路由表以拦截来自 gRPC 服务器的网络流量。

grpc_buffer_size

语法	`grpc_buffer_size` *size*;
默认值	`grpc_buffer_size 4k|8k;`
上下文	http, server, location

设置用于读取从 gRPC 服务器接收到的响应第一部分的缓冲区大小。响应会在同步接收后立即传递给客户端。

grpc_connect_timeout

语法	`grpc_connect_timeout` *time*;
默认值	`grpc_connect_timeout 60s;`
上下文	http, server, location

定义与 gRPC 服务器建立连接的超时时间。需要注意的是，此超时时间通常不能超过 75 秒。

grpc_connection_drop

语法	`grpc_connection_drop` *time* | `on` | `off`;
默认值	`grpc_connection_drop off;`
上下文	http, server, location

在代理服务器被从组中移除或通过*reresolve* 进程或*API* 命令 `DELETE` 标记为永久不可用后，允许终止与代理服务器的所有连接。

当处理客户端或代理服务器的下一个读或写事件时，连接被终止。

设置 *time* 启用连接终止超时；当设置为 `on` 时，连接会立即被丢弃。

grpc_hide_header

语法	`grpc_hide_header` *field*;
默认值	—
上下文	http, server, location

默认情况下，Angie 不会将 gRPC 服务器响应中的 `Date`、`Server` 和 `X-Accel-...` 头字段传递给客户端。*grpc_hide_header* 指令设置额外的字段不会被传递。相反，如果需要允许传递字段，可以使

用*grpc_pass_header* 指令。

grpc_ignore_headers

语法	`grpc_ignore_headers` *field* ...;
默认值	—
上下文	http, server, location

禁用处理来自 gRPC 服务器的某些响应头字段。可以忽略的字段包括："X-Accel-Redirect" 和"X-Accel-Charset"。

如果未禁用，这些头字段的处理会产生以下效果：

- "X-Accel-Redirect" 执行对指定 URI 的内部 重定向；
- "X-Accel-Charset" 设置响应的期望*charset*。

grpc_intercept_errors

语法	`grpc_intercept_errors on | off;`
默认值	`grpc_intercept_errors off;`
上下文	http, server, location

确定 gRPC 响应的状态码大于或等于 300 时，是否应将其传递给客户端，或者被拦截并重定向到 Angie 进行处理，与*error_page* 指令一起使用。

grpc_next_upstream

语法	`grpc_next_upstream error | timeout | invalid_header | http_500 | http_502 | http_503 | http_504 | http_403 | http_404 | http_429 | non_idempotent | off` ...;
默认值	`grpc_next_upstream error timeout;`
上下文	http, server, location

指定在什么情况下请求应传递到上游池 中的下一个服务器：

error	在建立与服务器的连接、向其传递请求或读取响应头时发生错误；
timeout	在建立与服务器的连接、向其传递请求或读取响应头时发生超时；
invalid_header	服务器返回空或无效响应；
http_500	服务器返回状态码 500 的响应；
http_502	服务器返回状态码 502 的响应；
http_503	服务器返回状态码 503 的响应；
http_504	服务器返回状态码 504 的响应；
http_403	服务器返回状态码 403 的响应；
http_404	服务器返回状态码 404 的响应；
http_429	服务器返回状态码 429 的响应；
non_idempotent	通常，带有 non-idempotent 方法的请求 (POST、LOCK、PATCH) 在向上游服务器发送请求后不会被传递到下一个服务器；启用此选项显式允许重试此类请求；
off	禁用将请求传递到下一个服务器。

备注

应注意，只有在尚未向客户端发送任何内容的情况下，才能将请求传递到下一个服务器。也就是说，如果在响应传输过程中发生错误或超时，无法修复此问题。

该指令还定义了与服务器通信的不成功尝试 的标准。

error timeout invalid_header	始终被视为不成功的尝试，即使未在指令中指定
http_500	仅在指令中指定时才被视为不成功的尝试
http_502	
http_503	
http_504	
http_429	
http_403	从不被视为不成功的尝试
http_404	

将请求传递到下一个服务器可以受到尝试次数 和时间 的限制。

grpc_next_upstream_timeout

语法	grpc_next_upstream_timeout *time*;
默认值	grpc_next_upstream_timeout 0;
上下文	http, server, location

限制可以传递给下一个 服务器的请求的时间。

0	关闭此限制

grpc_next_upstream_tries

语法	`grpc_next_upstream_tries` *number*;
默认值	`grpc_next_upstream_tries 0;`
上下文	http, server, location

限制可以传递请求到下一个 服务器的尝试次数。

0	关闭此限制

grpc_pass

语法	`grpc_pass` *address*;
默认值	—
上下文	location, if in location

设置 gRPC 服务器地址。地址可以指定为域名或 IP 地址，以及端口：

```
grpc_pass localhost:9000;
```

或作为 UNIX 域套接字路径：

```
grpc_pass unix:/tmp/grpc.socket;
```

或者，可以使用 `grpc://` 方案：

```
grpc_pass grpc://127.0.0.1:9000;
```

要使用 SSL 上的 gRPC，应使用 `grpcs://` 方案：

```
grpc_pass grpcs://127.0.0.1:443;
```

如果域名解析为多个地址，则所有地址将以轮询方式使用。此外，可以将地址指定为服务器组。如果使用了组，则不能与其一起指定端口；相反，必须为组内每个服务器单独指定端口。

参数值可以包含变量。在这种情况下，如果地址被指定为域名，则会在描述的服务器组中进行查找，如果未找到，则使用*resolver* 进行确定。

grpc_pass_header

语法	`grpc_pass_header` *field* ...;
默认值	—
上下文	http, server, location

允许将否则被禁用 的头字段从 gRPC 服务器传递给客户端。

grpc_read_timeout

语法	`grpc_read_timeout` *time*;
默认值	`grpc_read_timeout 60s;`
上下文	http, server, location

定义从 gRPC 服务器读取响应的超时。超时仅在两个连续的读取操作之间设置，而不是针对整个响应的传输。如果 gRPC 服务器在此时间内未传输任何内容，则连接将关闭。

grpc_send_timeout

语法	`grpc_send_timeout` *time*;
默认值	`grpc_send_timeout 60s;`
上下文	http, server, location

设置向 gRPC 服务器传输请求的超时。超时仅在两个连续的写操作之间设置，而不是针对整个请求的传输。如果 gRPC 服务器在此时间内未接收到任何内容，则连接将关闭。

grpc_set_header

语法	`grpc_set_header` *field value*;
默认值	`grpc_set_header Content-Length $content_length;`
上下文	http, server, location

允许重新定义或追加字段到请求头传递给 gRPC 服务器。值可以包含文本、变量及其组合。如果当前级别没有定义 grpc_set_header 指令，则这些指令将从上一个配置级别继承。

如果头字段的值为空字符串，则该字段将不会传递给 gRPC 服务器：

```
grpc_set_header Accept-Encoding "";
```

grpc_socket_keepalive

语法	`grpc_socket_keepalive on | off;`
默认值	`grpc_socket_keepalive off;`
上下文	http, server, location

配置与 gRPC 服务器的外发连接的“TCP keepalive”行为。

`""`	默认情况下，操作系统的设置对套接字生效。
`on`	套接字的 *SO_KEEPALIVE* 选项被打开。

grpc_ssl_certificate

语法	`grpc_ssl_certificate` *file*;
默认值	—
上下文	http, server, location

指定用于认证 gRPC SSL 服务器的 PEM 格式证书文件。文件名中可以使用变量。

grpc_ssl_certificate_cache

语法	`grpc_ssl_certificate_cache off;` `grpc_ssl_certificate_cache max=`*N* [`inactive=`*time*] [`valid=`*time*];
默认值	`grpc_ssl_certificate_cache off;`
上下文	http, server, location

定义一个用于存储通过变量指定的*SSL* 证书 和私钥 的缓存。

该指令支持以下参数：

- `max` —设置缓存中元素的最大数量。缓存溢出时，最少使用（LRU）的元素将被移除。
- `inactive` —指定在多长时间未访问后移除元素。默认值为 10 秒。
- `valid` —指定缓存元素被认为有效并可重复使用的时间。默认值为 60 秒。超过此时间后，证书将重新加载或重新验证。
- `off` —禁用缓存。

示例：

```
grpc_ssl_certificate       $grpc_ssl_server_name.crt;
grpc_ssl_certificate_key   $grpc_ssl_server_name.key;
grpc_ssl_certificate_cache max=1000 inactive=20s valid=1m;
```

grpc_ssl_certificate_key

语法	`grpc_ssl_certificate_key` *file*;
默认值	—
上下文	http, server, location

指定用于认证 gRPC SSL 服务器的 PEM 格式秘密密钥文件。

可以指定值"engine:*name*:id" 作为文件，这将从指定的 OpenSSL 引擎名称中加载带有指定 ID 的秘密密钥。文件名中可以使用变量。

grpc_ssl_ciphers

语法	`grpc_ssl_ciphers` *ciphers*;
默认值	`grpc_ssl_ciphers DEFAULT;`
上下文	http, server, location

指定向 gRPC SSL 服务器请求的启用密码。密码以 OpenSSL 库理解的格式指定。

密码列表取决于已安装的 OpenSSL 版本。可以使用 `openssl ciphers` 命令查看完整列表。

注意

当使用 OpenSSL 时，`grpc_ssl_ciphers` 指令 * 不会 * 配置 TLS 1.3 的加密套件。如需配置 TLS 1.3 的加密套件，请使用*grpc_ssl_conf_command* 指令，它是为支持高级 SSL 配置而添加的。

- 在 LibreSSL 中，可以使用 `grpc_ssl_ciphers` 配置 TLS 1.3 加密套件。
- 在 BoringSSL 中，无法配置 TLS 1.3 加密套件。

grpc_ssl_conf_command

语法	`grpc_ssl_conf_command` *name value*;
默认值	—
上下文	http, server, location

在与 gRPC SSL 服务器建立连接时设置任意的 OpenSSL 配置 命令。

重要

当使用 OpenSSL 1.0.2 或更高版本时支持该指令。要在 OpenSSL 中配置 TLS 1.3 密码套件，请使用 `ciphersuites` 命令。

可以在同一层级指定多个 *grpc_ssl_conf_command* 指令。如果当前级别没有定义 *grpc_ssl_conf_command* 指令，则这些指令将从上一个配置级别继承。

小心

请注意，直接配置 OpenSSL 可能会导致意外行为。

grpc_ssl_crl

语法	`grpc_ssl_crl` *file*;
默认值	—
上下文	http, server, location

指定带有撤销证书（CRL）的 PEM 格式文件，用于验证 gRPC SSL 服务器的证书。

grpc_ssl_name

语法	`grpc_ssl_name` *name*;
默认值	`grpc_ssl_name ``host` 来自 grpc_pass;‘
上下文	http, server, location

允许覆盖用于验证 gRPC SSL 服务器证书的服务器名称，并在与 gRPC SSL 服务器建立连接时通过 SNI 传递。

默认情况下，使用*grpc_pass* URL 的主机部分。

grpc_ssl_password_file

语法	`grpc_ssl_password_file` *file*;
默认值	—
上下文	http, server, location

指定一个文件，其中包含秘密密钥 的密码，每个密码单独一行。当加载密钥时，将依次尝试密码。

grpc_ssl_protocols

语法	`grpc_ssl_protocols` [SSLv2] [SSLv3] [TLSv1] [TLSv1.1] [TLSv1.2] [TLSv1.3];
默认值	`grpc_ssl_protocols TLSv1.2 TLSv1.3;`
上下文	http, server, location

在 1.2.0 版本发生变更: `TLSv1.3` 参数已添加到默认集。

启用指定的协议以请求 gRPC HTTPS 服务器。

grpc_ssl_server_name

语法	`grpc_ssl_server_name on | off;`
默认值	`grpc_ssl_server_name off;`
上下文	http, server, location

启用或禁用通过*grpc_ssl_name* 指令设置的服务器名称在与 gRPC SSL 服务器建立连接时通过 TLS 扩展 服务器名称指示 (SNI, RFC 6066) 进行传递。

grpc_ssl_session_reuse

语法	`grpc_ssl_session_reuse on | off;`
默认值	`grpc_ssl_session_reuse on;`
上下文	http, server, location

确定在与 gRPC 服务器交互时 SSL 会话是否可以重用。如果日志中出现错误"*SSL3_GET_FINISHED:digest check failed*"，请尝试禁用会话重用。

grpc_ssl_trusted_certificate

语法	`grpc_ssl_trusted_certificate` *file*;
默认值	—
上下文	http, server, location

指定一个带有受信任 CA 证书的 PEM 格式文件，用于验证 gRPC SSL 服务器的证书。

grpc_ssl_verify

语法	`grpc_ssl_verify on | off;`
默认值	`grpc_ssl_verify off;`
上下文	http, server, location

启用或禁用对 gRPC SSL 服务器证书的验证。

grpc_ssl_verify_depth

语法	`grpc_ssl_verify_depth` *number*;
默认值	`grpc_ssl_verify_depth 1;`
上下文	http, server, location

设置 gRPC SSL 服务器证书链中的验证深度。

GunZIP

该模块是一个过滤器，用于为不支持"gzip" 编码方法的客户端解压缩带有"Content-Encoding: gzip" 的响应。当需要存储压缩数据以节省空间和减少 I/O 成本时，该模块将非常有用。

当 从源代码构建时，默认情况下不会构建此模块；应通过 `--with-http_gunzip_module` 构建选项启用。

在来自 我们的仓库的软件包和镜像中，构建中已包含该模块。

配置示例

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

指令

gunzip

语法	`gunzip on | off;`
默认值	`gunzip off;`
上下文	http, server, location

启用或禁用对缺乏 gzip 支持的客户端的压缩响应解压缩。如果启用，还会考虑以下指令以确定客户端是否支持 gzip：*gzip_http_version*、*gzip_proxied* 和*gzip_disable*。另请参阅*gzip_vary* 指令。

gunzip_buffers

语法	`gunzip_buffers` 数量 大小;
默认值	`gunzip_buffers 32 4k | 16 8k;`
上下文	http, server, location

设置用于解压缩响应的缓冲区的数量和大小。默认情况下，缓冲区大小等于一个内存页大小。这取决于平台，可能是 4K 或 8K。

GZip

该模块是一个使用"gzip" 方法压缩响应的过滤器。这通常有助于将传输数据的大小减少一半甚至更多。

> **小心**
>
> 使用 SSL/TLS 协议时，压缩响应可能会受到 BREACH 攻击。

配置示例

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

可以使用*$gzip_ratio* 变量来记录达到的压缩比率。

指令

gzip

Syntax	`gzip on	off;`
默认	`gzip off;`
Context	http, server, location, if in location

启用或禁用响应的 gzip 压缩。

gzip_buffers

Syntax	`gzip_buffers` *number size*;
默认	`gzip_buffers 32 4k	16 8k;`
Context	http, server, location

设置用于压缩响应的缓冲区数量和大小。默认情况下，缓冲区大小等于一个内存页大小，这取决于平台，通常是 4K 或 8K。

gzip_comp_level

Syntax	`gzip_comp_level` *level*;
默认	`gzip_comp_level 1;`
Context	http, server, location

设置响应的 gzip 压缩级别。可接受的值范围从 1 到 9。

gzip_disable

Syntax	`gzip_disable` *regex ...*;
默认	—
Context	http, server, location

禁止对”User-Agent” 请求头字段与指定正则表达式匹配的请求进行 gzip 压缩。

特殊的掩码 `msie6` 对应于正则表达式"*MSIE [4-6].*"，但工作速度更快。"*MSIE 6.0; ... SV1*" 被排除在该掩码之外。

gzip_http_version

Syntax	`gzip_http_version 1.0	1.1;`
默认	`gzip_http_version 1.1;`
Context	http, server, location

设置请求所需的最低 HTTP 版本以压缩响应。

gzip_min_length

Syntax	`gzip_min_length` *length*;
默认	`gzip_min_length 20;`
Context	http, server, location

设置将被 gzip 压缩的响应的最小长度。长度仅从"Content-Length" 响应头字段中确定。

gzip_proxied

Syntax	`gzip_proxied off	expired	no-cache	no-store	private	no_last_modified	no_etag	auth	any` ...;
默认	`gzip_proxied off;`
Context	http, server, location

根据请求和响应的情况启用或禁用对代理请求的响应进行 gzip 压缩。请求是代理请求的事实由请求头字段"Via" 的存在来确定。该指令接受多个参数：

`off`	禁用对所有代理请求的压缩，忽略其他参数；
`expired`	如果响应头包含"Expires" 字段且值禁止缓存，则启用压缩；
`no-cache`	如果响应头包含"Cache-Control" 字段且具有"no-cache" 参数，则启用压缩；
`no-store`	如果响应头包含"Cache-Control" 字段且具有"no-store" 参数，则启用压缩；
`private`	如果响应头包含"Cache-Control" 字段且具有"private" 参数，则启用压缩；
`no_last_modified`	如果响应头不包含"Last-Modified" 字段，则启用压缩；
`no_etag`	如果响应头不包含"ETag" 字段，则启用压缩；
`auth`	如果请求头包含"Authorization" 字段，则启用压缩；
`any`	对所有代理请求启用压缩。

gzip_types

Syntax	gzip_types mime-*type* ...;
默认	gzip_types text/html;
Context	http, server, location

启用对指定 MIME 类型的响应进行 gzip 压缩，除了 text/html 之外。特殊值"*" 匹配任何 MIME 类型。text/html 类型的响应始终被压缩。

gzip_vary

Syntax	gzip_vary on | off;
默认	gzip_vary off;
Context	http, server, location

启用或禁用在响应头中插入"Vary: Accept-Encoding" 字段，如果指令*gzip*、*gzip_static* 或*gunzip* 激活。

内置变量

$gzip_ratio

计算出的压缩比率，作为原始响应大小与压缩响应大小的比率。

GZip Static

允许发送带有".gz" 文件扩展名的预压缩文件，而不是常规文件。

当从源码 构建时，该模块默认未构建；它应通过 --with-http_gzip_static_module 构建选项启用。

在从 我们的仓库中的包和镜像中，该模块已包含在构建中。

配置示例

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

指令

gzip_static

语法	gzip_static on | off | always;
默认值	gzip_static off;
上下文	http, server, location

启用 (on) 或禁用 (off) 检查预压缩文件的存在。以下指令也会被考虑：*gzip_http_version*、*gzip_proxied*、*gzip_disable* 和*gzip_vary*。

使用 always 时，压缩文件在所有情况下都会被使用，而无需检查客户端是否支持。这在磁盘上没有未压缩文件或者使用了*GunZIP* 模块时很有用。

文件可以通过 gzip 命令或任何其他兼容命令进行压缩。建议原始文件和压缩文件的修改日期和时间相同。

Headers

允许在响应头中添加"Expires" 和"Cache-Control" 头字段，以及任意字段。

配置示例

```
expires    24h;
expires    modified +24h;
expires    @24h;
expires    0;
expires    -1;
expires    epoch;
expires    $expires;
add_header Cache-Control private;
```

指令

add_header

语法	add_header *name value* [always];
默认值	—
上下文	http, server, location, if in location

在响应码为 200、201 (1.3.10)、204、206、301、302、303、304、307 或 308 时，添加指定字段到响应头。参数值可以包含变量。

可以有多个 *add_header* 指令。只有在当前级别未定义任何 *add_header* 指令时，这些指令才会从上一级配置继承。

如果指定了 always 参数，则无论响应码如何，都会添加该头字段。

add_trailer

语法	add_trailer *name value* [always];
默认值	—
上下文	http, server, location, if in location

在响应码为 200、201、206、301、302、303、307 或 308 时，添加指定字段至响应的末尾。参数值可以包含变量。

可以有多个 *add_trailer* 指令。只有在当前级别未定义任何 *add_trailer* 指令时，这些指令才会从上一级配置继承。

如果指定了 `always` 参数，则无论响应码如何，都会添加该字段。

expires

语法	`expires` [`modified`] *time*; `expires epoch` | `max` | `off`;
默认值	`expires off;`
上下文	http, server, location, if in location

启用或禁用在响应码为 200、201、204、206、301、302、303、304、307 或 308 时添加或修改"Expires" 和"Cache-Control" 响应头字段。参数可以是正或负的时间。

"Expires" 字段中的时间是当前时间与指令中指定时间的总和。如果使用了 `modified` 参数，则时间是文件修改时间与指令中指定时间的总和。

此外，可以使用"@" 前缀指定一天中的某个时间：

```
expires @15h30m;
```

"Cache-Control" 字段的内容取决于指定时间的符号：

- 时间为负—"Cache-Control: no-cache"。
- 时间为正或零—"Cache-Control: max-age=‘t‘"，其中 *t* 是指令中指定的时间，以秒为单位。

`epoch`	将"Expires" 设置为"Thu, 01 Jan 1970 00:00:01 GMT"，"Cache-Control" 设置为"no-cache"。
`max`	将"Expires" 设置为"Thu, 31 Dec 2037 23:55:55 GMT"，"Cache-Control" 设置为 10 年。
`off`	禁止添加或修改"Expires" 和"Cache-Control" 响应头字段。

最后的参数值可以包含变量：

```
map $sent_http_content_type $expires {
    default         off;
    application/pdf 42d;
    ~image/         max;
}

expires $expires;
```

Image Filter

该模块是一个过滤器，用于转换 JPEG、GIF、PNG 和 WebP 格式的图像。

当 从源代码构建时，默认不会构建此模块；需要使用 `--with-http_image_filter_module` 构建选项启用。

在我们的代码库中，该模块是 动态构建的，并作为名为 `angie-module-image-filter` 或 `angie-pro-module-image-filter` 的单独软件包提供。

重要

此模块利用 libgd 库。建议使用库的最新可用版本。

要转换 WebP 格式的图像，`libgd` 库必须编译时启用 WebP 支持。

配置示例

```
location /img/ {
    proxy_pass   http://backend;
    image_filter resize 150 100;
    image_filter rotate 90;
    error_page   415 = /empty;
}

location = /empty {
    empty_gif;
}
```

指令

image_filter

语法	• `image_filter off;` • `image_filter test;` • `image_filter size;` • `image_filter` rotate 90 | 180 | 270; • `image_filter` resize *width height*; • `image_filter` crop *width height*;
默认	`image_filter off;`
上下文	location

设置要对图像执行的转换类型：

`off`	关闭周围位置的模块处理。
`test`	确保响应为 JPEG、GIF、PNG 或 WebP 格式的图像。否则，将返回 415（不支持的媒体类型）错误。
`size`	以 JSON 格式输出图像的信息，例如： *`{ "img" : { "width": 100, "height": 100, "type": "gif" } }`* 如果发生错误，输出如下：`{}`
`rotate 90| 180| 270`	逆时针旋转图像指定的度数。参数值可以包含变量。此模式可以单独使用或与 `resize` 和 `crop` 转换一起使用。
`resize` *width height*	按照指定的大小按比例缩小图像。要仅在一个维度上缩小，可以将另一个维度指定为"-"。如果发生错误，服务器将返回 415（不支持的媒体类型）代码。参数值可以包含变量。当与 `rotate` 参数一起使用时，旋转在缩小 **之后**发生。
`crop` *width height*	按照更大边的大小按比例缩小图像，并裁剪另一边的多余边缘。要仅在一个维度上缩小，可以将另一个维度指定为"-"。如果发生错误，服务器将返回 415（不支持的媒体类型）代码。参数值可以包含变量。当与 `rotate` 参数一起使用时，旋转在缩小 **之前**发生。

image_filter_buffer

语法	`image_filter_buffer` *size*;
默认	`image_filter_buffer 1M;`
上下文	http, server, location

设置用于读取图像的缓冲区的最大大小。当超出该大小时，服务器返回 415（不支持的媒体类型）错误。

image_filter_interlace

语法	`image_filter_interlace on | off;`
默认	`image_filter_interlace off;`
上下文	http, server, location

如果启用，最终图像将会交错。对于 JPEG，最终图像将为" 渐进式 JPEG" 格式。

image_filter_jpeg_quality

语法	`image_filter_jpeg_quality` *quality*;
默认	`image_filter_jpeg_quality 75;`
上下文	http, server, location

设置转换后的 JPEG 图像的期望质量。可接受的值范围为 1 到 100。较小的值通常意味着图像质量较低和传输的数据较少。建议的最大值为 95。参数值可以包含变量。

image_filter_sharpen

语法	`image_filter_sharpen` *percent*;
默认	`image_filter_sharpen 0;`
上下文	http, server, location

增加最终图像的锐度。锐度百分比可以超过 100。`0` 值禁用锐化。参数值可以包含变量。

image_filter_transparency

语法	`image_filter_transparency on | off;`
默认	`image_filter_transparency on;`
上下文	http, server, location

定义在转换 GIF 图像或具有调色板指定颜色的 PNG 图像时是否应保留透明度。透明度的丧失会导致图像质量更好。PNG 中的 alpha 通道透明度始终保留。

image_filter_webp_quality

语法	`image_filter_webp_quality` *quality*;
默认	`image_filter_webp_quality 80;`
上下文	http, server, location

设置转换后的 WebP 图像的期望质量。可接受的值范围为 1 到 100。较小的值通常意味着图像质量较低和传输的数据较少。参数值可以包含变量。

Index

该模块处理以斜杠字符 (`/`) 结尾的请求。这类请求也可以由*http_autoindex* 和*http_random_index* 模块处理。

配置示例

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

指令

index

语法	index *file* ...;
默认值	index index.html;
上下文	http, server, location

定义将用作索引的文件。文件名可以包含变量。文件按照指定的顺序进行检查。列表的最后一个元素可以是一个具有绝对路径的文件。示例：

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

需要注意的是，使用索引文件会导致内部重定向，并且请求可以在不同的位置进行处理。例如，使用以下配置：

```
location = / {
    index index.html;
}

location / {
    #    ...
}
```

一个 "/" 请求实际上将在第二个位置作为 "/index.html" 进行处理。

JS

该模块用于在 njs 中实现处理程序——这是 JavaScript 语言的一个子集。

在我们的代码库中，该模块是 动态构建并作为名为 angie-module-njs 或 angie-pro-module-njs 的单独包提供。

> **备注**
>
> 此软件包还提供一个轻量级版本，名为 ...-njs-light；但它无法与常规版本同时使用。

配置示例

```
http {
    js_import http.js;

    js_set $foo     http.foo;
    js_set $summary http.summary;
    js_set $hash    http.hash;

    resolver 127.0.0.53;
```

```
    server {
        listen 8000;

        location / {
            add_header X-Foo $foo;
            js_content http.baz;
        }

        location = /summary {
            return 200 $summary;
        }

        location = /hello {
            js_content http.hello;
        }

        location = /fetch {
            js_content                   http.fetch;
            js_fetch_trusted_certificate /path/to/ISRG_Root_X1.pem;
        }

        location = /crypto {
            add_header Hash $hash;
            return     200;
        }
    }
}
```

http.js 文件:

```
function foo(r) {
    r.log("hello from foo() handler");
    return "foo";
}

function summary(r) {
    var a, s, h;

    s = "JS summary\n\n";

    s += "Method: " + r.method + "\n";
    s += "HTTP version: " + r.httpVersion + "\n";
    s += "Host: " + r.headersIn.host + "\n";
    s += "Remote Address: " + r.remoteAddress + "\n";
    s += "URI: " + r.uri + "\n";

    s += "Headers:\n";
    for (h in r.headersIn) {
```

```
        s += "  header '" + h + "' is '" + r.headersIn[h] + "'\n";
    }

    s += "Args:\n";
    for (a in r.args) {
        s += "  arg '" + a + "' is '" + r.args[a] + "'\n";
    }

    return s;
}

function baz(r) {
    r.status = 200;
    r.headersOut.foo = 1234;
    r.headersOut['Content-Type'] = "text/plain; charset=utf-8";
    r.headersOut['Content-Length'] = 15;
    r.sendHeader();
    r.send("nginx");
    r.send("java");
    r.send("script");

    r.finish();
}

function hello(r) {
    r.return(200, "Hello world!");
}

async function fetch(r) {
    let results = await Promise.all([ngx.fetch('https://google.com/'),
                                     ngx.fetch('https://google.ru/')]);

    r.return(200, JSON.stringify(results, undefined, 4));
}

async function hash(r) {
    let hash = await crypto.subtle.digest('SHA-512', r.headersIn.host);
    r.setReturnValue(Buffer.from(hash).toString('hex'));
}

export default {foo, summary, baz, hello, fetch, hash};
```

指令

js_body_filter

语法	`js_body_filter` *function* | *module.function* [`buffer_type=`*string* | *buffer*];
默认	—
上下文	location, if in location, limit_except

将 njs 函数设置为响应体过滤器。过滤函数在响应体的每个数据块上被调用，具有以下参数：

`r`	HTTP 请求 对象
`data`	输入的数据块，可以是字符串或 Buffer，具体取决于 *buffer_type* 值，默认是字符串。
`flags`	具有以下属性的对象：`last` —布尔值 `true` —如果数据是最后一个缓冲区。

过滤函数可以通过调用 r.sendBuffer() 将其修改后的输入数据块传递给下一个体过滤器。例如，要将响应体中的所有小写字母转换为小写：

```
function filter(r, data, flags) {
    r.sendBuffer(data.toLowerCase(), flags);
}
```

要停止过滤（后续数据块将直接传递给客户端，而不调用 *js_body_filter*），可以使用 r.done()。

如果过滤函数更改了响应体的长度，则需要在*js_header_filter* 中清除”Content-Length” 响应头（如果有的话），以强制使用分块传输编码。

备注

由于 *js_body_filter* 处理程序立即返回其结果，因此仅支持同步操作。因此，不支持异步操作，例如 r.subrequest() 或 setTimeout()。

js_content

语法	`js_content` *function* | *module.function*;
默认	—
上下文	location, if in location, limit_except

将 njs 函数设置为位置内容处理程序。可以引用模块函数。

js_fetch_buffer_size

语法	`js_fetch_buffer_size` *size*;
默认	`js_fetch_buffer_size 16k;`
上下文	http, server, location

设置用于读取和写入 Fetch API 的缓冲区大小。

js_fetch_ciphers

语法	`js_fetch_ciphers` *ciphers*;
默认	`js_fetch_ciphers HIGH:!aNULL:!MD5;`
上下文	http, server, location

指定与 Fetch API 的 HTTPS 连接启用的密码。密码的格式与 OpenSSL 库理解的格式相同。

密码列表取决于已安装的 OpenSSL 版本。可以使用 `openssl ciphers` 命令查看完整列表。

js_fetch_max_response_buffer_size

语法	`js_fetch_max_response_buffer_size` *size*;
默认	`js_fetch_max_response_buffer_size 1m;`
上下文	http, server, location

设置通过 Fetch API 接收到的响应的最大大小。

js_fetch_protocols

语法	`js_fetch_protocols` [TLSv1] [TLSv1.1] [TLSv1.2] [TLSv1.3];
默认	`js_fetch_protocols TLSv1 TLSv1.1 TLSv1.2;`
上下文	http, server, location

启用与 Fetch API 的 HTTPS 连接的指定协议。

js_fetch_timeout

语法	`js_fetch_timeout` *time*;
默认	`js_fetch_timeout 60s;`
上下文	http, server, location

定义与 Fetch API 的读取和写入超时。超时仅在两个连续的读/写操作之间设置，而不是针对整个响应。如果在此时间内没有传输数据，连接将被关闭。

js_fetch_trusted_certificate

语法	`js_fetch_trusted_certificate` *file*;
默认	—
上下文	http, server, location

指定一个包含 PEM 格式的受信任 CA 证书的文件，用于通过 Fetch API 验证 HTTPS 证书。

js_fetch_verify

语法	`js_fetch_verify on` | `off`;
默认	`js_fetch_verify on;`
上下文	http, server, location

启用或禁用使用 Fetch API 验证 HTTPS 服务器证书。

js_fetch_verify_depth

语法	`js_fetch_verify_depth` *number*;
默认	`js_fetch_verify_depth 100;`
上下文	http, server, location

设置与 Fetch API 的 HTTPS 服务器证书链中的验证深度。

js_header_filter

语法	`js_header_filter` *function* | *module.function*;
默认	—
上下文	location, if in location, limit_except

将 njs 函数设置为响应头过滤器。该指令允许更改响应头的任意字段。

备注

由于 *js_header_filter* 处理程序立即返回其结果，因此仅支持同步操作。因此，不支持异步操作，例如 r.subrequest() 或 setTimeout()。

js_import

语法	`js_import` *module.js* | *export_name from module.js*;
默认	—
上下文	http, server, location

导入实现 njs 中位置和变量处理程序的模块。*export_name* 用作访问模块函数的命名空间。如果未指定 *export_name*，则将使用模块名称作为命名空间。

```
js_import http.js;
```

在这里，模块名称 `http` 被用作访问导出的命名空间。如果导入的模块导出 `foo()`，则使用 `http.foo` 来引用它。

可以指定多个 `js_import` 指令。

js_path

语法	`js_path` *path*;
默认	—
上下文	http, server, location

为 njs 模块设置额外的路径。

js_preload_object

语法	`js_preload_object` *name.json* | *name from file.json*;
默认	—
上下文	http, server, location

在配置时预加载一个不可变对象。*name* 用作全局变量的名称，通过该变量可以在 njs 代码中访问对象。如果未指定 *name*，则将使用文件名。

```
js_preload_object map.json;
```

在这里，*map* 被用作访问预加载对象时的名称。

可以指定多个 *js_preload_object* 指令。

js_set

语法	`js_set` *$variable function* | *module.function*;
默认	—
上下文	http, server, location

为指定变量设置一个 njs 函数。可以引用模块函数。

当第一次引用变量时，会调用该函数。具体的时机取决于变量被引用的阶段。这可以用于执行一些与变量评估无关的逻辑。例如，如果变量仅在*log_format* 指令中被引用，则其处理程序不会在日志阶段之前执行。这个处理程序可以在请求释放之前执行一些清理工作。

备注

由于 *js_set* 处理程序立即返回其结果，因此仅支持同步回调。因此，不支持异步回调，如 r.subrequest() 或 setTimeout()。

js_shared_dict_zone

语法	`js_shared_dict_zone zone=`*name:size* [`timeout=`*time*] [`type=`*string* | *number*] [`evict`];
默认	—
上下文	http

设置共享内存区域的名称和大小，该区域在工作进程之间保持键值字典的共享。

`type`	可选参数，允许将值类型重定义为 `number`；默认情况下，共享字典使用 `string` 作为键和值
`timeout`	可选参数，设置在共享字典条目从区域中移除之前的时间
`evict`	可选参数，当区域存储耗尽时，移除最旧的键值对

示例：

```
example.conf:
    # 创建一个 1MB 的字典，值为字符串，
    # 在 60 秒不活动后移除键值对：
    js_shared_dict_zone zone=foo:1M timeout=60s;

    # 创建一个 512KB 的字典，值为字符串，
    # 当区域溢出时移除最旧的键值对：
    js_shared_dict_zone zone=bar:512K timeout=30s evict;

    # 创建一个 32KB 的持久字典，值为数字：
    js_shared_dict_zone zone=num:32k type=number;
```

```
example.js:
    function get(r) {
        r.return(200, ngx.shared.foo.get(r.args.key));
    }
```

```
    function set(r) {
        r.return(200, ngx.shared.foo.set(r.args.key, r.args.value));
    }

    function delete(r) {
        r.return(200, ngx.shared.bar.delete(r.args.key));
    }

    function increment(r) {
        r.return(200, ngx.shared.num.incr(r.args.key, 2));
    }
```

js_var

语法	js_var *$variable* [*value*];
默认	—
上下文	http, server, location

声明一个 可写 变量。值可以包含文本、变量及其组合。该变量在重定向后不会被覆盖，不像通过 set 指令创建的变量。

请求参数

每个 HTTP njs 处理程序接收一个参数，即 请求 对象。

Limit Conn

该模块用于限制每个定义的关键字的连接数量，特别是来自单个 IP 地址的连接数量。

并非所有连接都会被计数。仅在请求被服务器处理且整个请求头已被读取的情况下，才会计数连接。

配置示例

```
http {
    limit_conn_zone $binary_remote_addr zone=addr:10m;

    ...

    server {

        ...

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

指令

limit_conn

语法	`limit_conn` *zone number*;
默认	—
上下文	http, server, location

为给定的键值设置共享内存区和最大允许的连接数。当超过此限制时，服务器将返回错误 作为请求的回复。例如，以下指令

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

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

仅允许每个 IP 地址同时建立一个连接。

> **备注**
>
> 在 HTTP/2 和 HTTP/3 中，每个并发请求被视为一个单独的连接。

可以有多个 `limit_conn` 指令。例如，以下配置将限制每个客户端 IP 地址对服务器的连接数量，同时限制对虚拟服务器的总连接数量：

```
limit_conn_zone $binary_remote_addr zone=perip:10m;
limit_conn_zone $server_name zone=perserver:10m;

server {
    ...
    limit_conn perip 10;
    limit_conn perserver 100;
}
```

这些指令仅在当前级别没有定义 `limit_conn` 指令时，从上一级配置级别继承。

limit_conn_dry_run

语法	`limit_conn_dry_run on | off;`
默认	`limit_conn_dry_run off;`
上下文	http, server, location

启用干运行模式。在此模式下，连接数量没有限制，但是在共享内存区 中，超出连接的数量仍然会被正

常统计。

limit_conn_log_level

语法	`limit_conn_log_level info | notice | warn | error;`
默认	`limit_conn_log_level error;`
上下文	http, server, location

设置服务器限制连接数量时所需的日志记录级别。

limit_conn_status

语法	`limit_conn_status` *code*;
默认	`limit_conn_status 503;`
上下文	http, server, location

设置对被拒绝请求的响应状态码。

limit_conn_zone

语法	`limit_conn_zone` *key* zone = *name:size*;
默认	—
上下文	http

为共享内存区设置参数，该区域将保留各种键的状态。特别是，状态包括当前的连接数量。键可以包含文本、变量及其组合。键值为空的请求不被计数。

使用示例：

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

在这里,客户端 IP 地址作为键。请注意,这里使用的是 `$binary_remote_addr` 变量,而不是 `$remote_addr`。

变量 `$remote_addr` 的大小可以从 7 到 15 字节不等。存储的状态在 32 位平台上占用 32 或 64 字节的内存，在 64 位平台上始终占用 64 字节。

变量 `$binary_remote_addr` 的大小对于 IPv4 地址始终为 4 字节，或对于 IPv6 地址始终为 16 字节。存储的状态在 32 位平台上始终占用 32 或 64 字节，在 64 位平台上始终占用 64 字节。

一个兆字节的区域可以保存大约 32000 个 32 字节的状态或大约 16000 个 64 字节的状态。如果区域存储空间耗尽，服务器将对所有进一步的请求返回错误。

内置变量

`$limit_conn_status`

保存连接数量限制的结果：`PASSED` 、`REJECTED` 或 `REJECTED_DRY_RUN`

Limit Req

该模块用于限制每个定义键的请求处理速率，特别是来自单个 IP 地址的请求处理速率。该限制是使用”漏桶” 方法实现的。

配置示例

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

    ...

    server {

        ...

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

指令

`limit_req`

语法	`limit_req zone=`*name* [`burst=`*number*] [nodelay | `delay=`*number*];
默认	—
上下文	http, server, location

设置共享内存区域和请求的最大突发大小。如果请求速率超过为区域配置的速率，则其处理将被延迟，以使请求以定义的速率进行处理。过多的请求会被延迟，直到其数量超过最大突发大小，此时请求将以错误 终止。默认情况下，最大突发大小等于零。例如，以下指令

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

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

平均允许每秒不超过 1 个请求，突发不超过 5 个请求。

如果不希望在请求受到限制时延迟过多的请求，则应使用参数 `nodelay`：

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

delay 参数指定过多请求变为延迟的限制。默认值为零，即所有过多的请求都将被延迟。

可以有多个 limit_req 指令。例如，以下配置将限制来自单个 IP 地址的请求处理速率，同时限制虚拟服务器的请求处理速率：

```
limit_req_zone $binary_remote_addr zone=perip:10m rate=1r/s;
limit_req_zone $server_name zone=perserver:10m rate=10r/s;

server {
    ...
    limit_req zone=perip burst=5 nodelay;
    limit_req zone=perserver burst=10;
}
```

这些指令仅在当前级别没有定义 limit_req 指令时，从上一级配置级别继承。

limit_req_dry_run

语法	limit_req_dry_run on | off;
默认	limit_req_dry_run off;
上下文	http, server, location

启用干运行模式。在此模式下，请求处理速率不受限制，但在共享内存区域 中，过多请求的数量仍然会被正常记录。

limit_req_log_level

语法	limit_req_log_level info | notice | warn | error;
默认	limit_req_log_level error;
上下文	http, server, location

设置服务器由于速率超限而拒绝处理请求或延迟请求处理的日志记录级别。延迟的日志记录级别比拒绝的日志记录级别低一级；例如，如果指定了 limit_req_log_level notice，则延迟将以 info 级别记录。

limit_req_status

语法	limit_req_status *code*;
默认	limit_req_status 503;
上下文	http, server, location

设置拒绝请求时返回的状态码。

limit_req_zone

语法	`limit_req_zone` *key* `zone=`*name:size* `rate=`*rate*;
默认	—
上下文	http

设置共享内存区域的参数，该区域将为各种键保留状态。特别是，状态存储当前的过多请求数量。键可以包含文本、变量及其组合。键值为空的请求不被计入。

使用示例：

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

在这里，状态保存在 10 兆字节的区域 `one` 中，该区域的平均请求处理速率不得超过每秒 1 个请求。

客户端 IP 地址作为键。请注意，这里使用的是 `$binary_remote_addr` 变量，而不是 `$remote_addr`。

对于 IPv4 地址，变量 `$binary_remote_addr` 的大小始终为 4 字节，对于 IPv6 地址，大小为 16 字节。存储的状态在 32 位平台上始终占用 64 字节，在 64 位平台上占用 128 字节。

一个兆字节的区域可以保存大约 16000 个 64 字节的状态或大约 8000 个 128 字节的状态。

如果区域存储已耗尽，将删除最近最少使用的状态。如果即便如此仍无法创建新状态，则请求将以错误终止。

`rate` 以每秒请求数（r/s）表示。如果希望速率低于每秒一个请求，则以每分钟请求数（r/m）表示。例如，每秒半个请求表示为 30r/m。

内置变量

`$limit_req_status`

保持限制请求处理速率的结果：`PASSED`、`DELAYED`、`REJECTED`、`DELAYED_DRY_RUN` 或 `REJECTED_DRY_RUN`

Log

该模块以指定格式写入请求日志。

请求在处理结束的 location 上下文中记录日志。如果在请求处理期间发生了内部重定向，则可能与原始 location 不同。

配置示例

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

access_log /spool/logs/angie-access.log compression buffer=32k;
```

指令

access_log

语法	`access_log` *path* [*format* [`buffer=`*size*] [`gzip=`*level*]] [`flush=`*time*] [`if=`*condition*]; `access_log off;`
默认	`access_log logs/access.log combined;`（路径取决于 `--http-log-path` 构建选项）
上下文	http, server, location, if in location, limit_except

设置 *path*、*format* 和缓冲日志写入的配置。可以在同一配置级别上指定多个日志。通过在第一个参数中指定 `"syslog:"` 前缀，可以配置日志记录到*syslog*。特殊值 `off` 取消当前级别上的所有 `access_log` 指令。如果未指定格式，则使用预定义的 `"combined"` 格式。

如果使用了 `buffer` 或 `gzip` 参数，则写入日志将被缓冲。

小心

缓冲区大小不得超过对磁盘文件的原子写入大小。对于 FreeBSD，这个大小是无限的。

启用缓冲时，数据将会写入文件：

- 如果下一行日志不适合缓冲区；
- 如果缓冲的数据比 flush 参数指定的时间更旧；
- 当工作进程重新打开日志文件 或关闭时。

如果使用了 `gzip` 参数，则缓冲的数据将在写入文件之前进行压缩。压缩级别可以设置在 1（最快，压缩较少）和 9（最慢，压缩最好）之间。默认情况下，缓冲区大小为 `64K` 字节，压缩级别设置为 `1`。由于数据是以原子块的形式进行压缩的，因此日志文件可以随时通过 `zcat` 解压缩或读取。

示例：

```
access_log /path/to/log.gz combined gzip flush=5m;
```

重要

要使 gzip 压缩工作，Angie 必须使用 zlib 库构建。

文件路径可以包含变量，但此类日志有一些限制：

- 使用工作进程凭据的用户 应该具有在包含此类日志的目录中创建文件的权限；
- 缓冲写入无效；
- 每次日志写入时，文件都被打开和关闭。然而，由于可以将频繁使用的文件的描述符存储在缓存中，因此在*open_log_file_cache* 指令的 `valid` 参数指定的时间内，写入旧文件仍然可以继续；

- 在每次日志写入时，检查请求的根目录是否存在，如果不存在，则不创建日志。因此，最好在同一配置级别上同时指定*root* 和*access_log*：

```
server {
    root       /spool/vhost/data/$host;
    access_log /spool/vhost/logs/$host;
    ...
```

if 参数启用条件日志记录。如果条件计算结果为 "0" 或空字符串，则请求将不会被记录。在以下示例中，响应代码为 2xx 和 3xx 的请求将不会被记录：

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

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

log_format

语法	log_format *name* [escape=default | json | none] *string* ...;
默认	log_format combined "...";
上下文	http

指定日志格式。

escape 参数允许设置 json 或 default 变量中的字符转义，默认使用 default 转义。none 值禁用转义。

对于 default 转义，字符""、"\" 和其他值小于 32 或大于 126 的字符将被转义为"\xXX"。如果未找到变量值，则将记录一个连字符"-"。

对于 json 转义，所有在 JSON 字符串中不允许的字符将被转义：字符""" 和"\" 被转义为"\"" 和"\\"，值小于 32 的字符被转义为"\n"、"\r"、"\t"、"\b"、"\f" 或"\u00XX"。

发送到客户端的头行具有前缀 sent_http_，例如 $sent_http_content_range。

配置始终包括预定义的 combined 格式：

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

open_log_file_cache

语法	open_log_file_cache max=*N* [inactive=*time*] [min_uses=*N*] [valid=*time*]; open_log_file_cache off;
默认	open_log_file_cache off;
上下文	http, server, location

定义一个缓存，用于存储包含变量名称的频繁使用日志的文件描述符。该指令具有以下参数：

max	设置缓存中描述符的最大数量；如果缓存已满，则关闭最近最少使用（LRU）描述符
inactive	设置在此时间内没有访问之后缓存描述符关闭的时间；默认为 10 秒
min_uses	设置在 inactive 参数定义的时间内文件使用的最小次数，以便让描述符在缓存中保持打开；默认为 1
valid	设置在此时间后检查文件是否仍然存在并具有相同名称；默认为 60 秒
off	禁用缓存

使用示例：

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

Map

该模块创建的变量值依赖于其他变量的值。

配置示例

```
map $http_host $name {
    hostnames;

    default       0;

    example.com   1;
    *.example.com 1;
    example.org   2;
    *.example.org 2;
    .example.net  3;
    wap.*         4;
}

map $http_user_agent $mobile {
    default       0;
    "~Opera Mini" 1;
}
```

指令

map

语法	map *string $variable* { ... }
默认	—
上下文	http

创建一个新变量。其值取决于第一个参数，该参数作为带变量的字符串指定，例如：

```
set $var1 "foo";
set $var2 "bar";

map $var1$var2 $new_variable {
    default "foobar_value";
}
```

这里，变量 `$new_variable` 的值将由两个变量 `$var1` 和 `$var2` 组成，或者在这些变量未定义时使用默认值。

备注

由于变量仅在使用时被评估，因此即使声明大量的"map" 变量也不会给请求处理增加额外的成本。

map 块中的参数指定源值和结果值之间的映射。

源值可以作为字符串或正则表达式指定。

字符串匹配时忽略大小写。

正则表达式应以 `~` 符号开头以进行区分大小写的匹配，或者以 `~*` 符号开头以进行不区分大小写的匹配。正则表达式可以包含命名和位置捕获，稍后可以在其他指令中与结果变量一起使用。

如果源值与下面描述的特殊参数之一匹配，则应以 `\` 符号为前缀。

结果值可以包含文本、变量及其组合。

还支持以下特殊参数：

`default` *value*	如果源值与指定的变体都不匹配，则设置结果值。当未指定 *default* 时，默认结果值将为空字符串。
`hostnames`	表示源值可以是带有前缀或后缀掩码的主机名。该参数应在值列表之前指定。

例如，

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

以下两个记录

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

可以合并为：

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

`include` *file*	包含一个包含值的文件。可以有多个包含。
`volatile`	表示该变量不可缓存。

如果源值与多个指定变体匹配，例如掩码和正则表达式都匹配，将选择第一个匹配的变体，优先顺序如下：

1. 没有掩码的字符串值
2. 带有前缀掩码的最长字符串值，例如 `*.example.com`
3. 带有后缀掩码的最长字符串值，例如 `mail.*`
4. 第一个匹配的正则表达式（按在配置文件中的出现顺序）
5. 默认值 (`default`)

map_hash_bucket_size

语法	`map_hash_bucket_size` *size*;
默认	`map_hash_bucket_size 32|64|128;`
上下文	http

设置*map* 变量哈希表的桶大小。默认值取决于处理器的缓存行大小。设置哈希表的详细信息请参见单独。

map_hash_max_size

语法	`map_hash_max_size` *size*;
默认	`map_hash_max_size 2048;`
上下文	http

设置*map* 变量哈希表的最大大小。设置哈希表的详细信息请参见单独。

Memcached

该模块用于从 memcached 服务器获取响应。键在*$memcached_key* 变量中设置。响应应该通过外部手段预先放入 memcached 中。

配置示例

```
server {
    location / {
        set             $memcached_key "$uri?$args";
        memcached_pass host:11211;
        error_page      404 502 504 = @fallback;
    }

    location @fallback {
        proxy_pass      http://backend;
    }
}
```

指令

memcached_bind

语法	`memcached_bind` *address* [`transparent`] | `off`;
默认值	—
上下文	http, server, location

使到 memcached 服务器的外发连接从指定的本地 IP 地址发起，带有可选的端口。参数值可以包含变量。特殊值 `off` 取消从上一级配置继承的 *memcached_bind* 指令的效果，这允许系统自动分配本地 IP 地址和端口。

`transparent` 参数允许到 memcached 服务器的外发连接从非本地 IP 地址发起，例如，从客户端的真实 IP 地址发起：

```
memcached_bind $remote_addr transparent;
```

为了使此参数生效,通常需要以超级用户 权限运行 Angie 工作进程。在 Linux 上,如果指定了 `transparent` 参数，则不需要此操作，因为工作进程从主进程继承 *CAP_NET_RAW* 能力。

重要

必须配置内核路由表以截获来自 memcached 服务器的网络流量。

memcached_buffer_size

语法	`memcached_buffer_size` *size*;
默认值	`memcached_buffer_size 4k|8k;`
上下文	http, server, location

设置用于读取从 memcached 服务器接收到的响应第一部分的缓冲区大小。响应一接收到就同步传递给客户端。

memcached_connect_timeout

语法	`memcached_connect_timeout` *time*;
默认值	`memcached_connect_timeout 60s;`
上下文	http, server, location

定义与 memcached 服务器建立连接的超时时间。需要注意的是，此超时时间通常不能超过 75 秒。

memcached_gzip_flag

语法	`memcached_gzip_flag` *flag*;
默认值	—
上下文	http, server, location

启用测试 memcached 服务器响应中的标志，并在标志设置时将响应头的”Content-Encoding” 字段设置为”gzip”。

memcached_next_upstream

语法	`memcached_next_upstream error` | `timeout` | `invalid_response` | `not_found` | `off` ...;
默认值	`memcached_next_upstream error timeout;`
上下文	http, server, location

指定在何种情况下请求应传递给上游池 中的下一个服务器：

`error`	在与服务器建立连接、传递请求或读取响应头时发生错误；
`timeout`	在与服务器建立连接、传递请求或读取响应头时发生超时；
`invalid_response`	服务器返回空或无效响应；
`not_found`	在服务器上未找到响应；
`off`	禁止请求传递给下一个服务器。

备注

应记住，只有在尚未向客户端发送任何内容时，才能将请求传递给下一个服务器。也就是说，如果在响应传输过程中发生错误或超时，则无法解决。

该指令还定义了与服务器通信的不成功尝试 的标准。

`error`, `timeout`, `invalid_response`	总是被视为不成功尝试，即使它们未在指令中指定
`not_found`	从不被视为不成功尝试

将请求传递给下一个服务器的尝试次数可以通过尝试次数 和时间 进行限制。

memcached_next_upstream_timeout

语法	`memcached_next_upstream_timeout` *time*;
默认值	`memcached_next_upstream_timeout 0;`
上下文	http, server, location

限制请求可以传递到下一个 服务器的时间。

`0`	关闭此限制

memcached_next_upstream_tries

语法	`memcached_next_upstream_tries` *number*;
默认值	`memcached_next_upstream_tries 0;`
上下文	http, server, location

限制请求传递到下一个 服务器的尝试次数。

`0`	关闭此限制

memcached_pass

语法	`memcached_pass` *address*;
默认值	—
上下文	location, if in location

设置 memcached 服务器地址。地址可以指定为域名或 IP 地址，以及端口：

```
memcached_pass localhost:11211;
```

或者指定为 UNIX 域套接字路径：

```
memcached_pass unix:/tmp/memcached.socket;
```

如果一个域名解析为多个地址，它们将以轮循方式使用。此外，地址可以指定为服务器组。

memcached_read_timeout

语法	`memcached_read_timeout` *time*;
默认值	`memcached_read_timeout 60s;`
上下文	http, server, location

定义从 memcached 服务器读取响应的超时时间。超时时间仅在两次连续读取操作之间设置，而不是用于整个响应的传输。如果 memcached 服务器在此时间内没有传输任何内容，则连接将关闭。

memcached_send_timeout

语法	`memcached_send_timeout` *time*;
默认值	`memcached_send_timeout 60s;`
上下文	http, server, location

设置向 memcached 服务器传输请求的超时时间。超时时间仅在两次连续写入操作之间设置，而不是用于整个请求的传输。如果 memcached 服务器在此时间内没有接收到任何内容，则连接将关闭。

memcached_socket_keepalive

语法	`memcached_socket_keepalive on | off;`
默认值	`memcached_socket_keepalive off;`
上下文	http, server, location

配置到 memcached 服务器的外发连接的"TCP keepalive" 行为。

""	默认情况下，操作系统的设置对套接字有效。
on	为套接字打开 *SO_KEEPALIVE* 套接字选项。

内置变量

$memcached_key

定义从 memcached 服务器获取响应的键。

Mirror

该模块通过创建后台镜像子请求来实现对原始请求的镜像。对镜像子请求的响应将被忽略。

配置示例

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

location = /mirror {
    internal;
    proxy_pass http://test_backend$request_uri;
}
```

指令

mirror

语法	mirror *uri* | off;
默认值	mirror off;
上下文	http, server, location

设置原始请求将被镜像到的 URI。可以在同一级配置中指定多个镜像。

mirror_request_body

语法	mirror_request_body on | off;
默认值	mirror_request_body on;
上下文	http, server, location

指示客户端请求体是否被镜像。当启用时，客户端请求体将在创建镜像子请求之前被读取。在这种情况下，由*proxy_request_buffering*、*fastcgi_request_buffering*、*scgi_request_buffering* 和*uwsgi_request_buffering* 指令设置的非缓冲客户端请求体代理将被禁用。

```
location / {
    mirror /mirror;
    mirror_request_body off;
    proxy_pass http://backend;
```

```
}

location = /mirror {
    internal;
    proxy_pass http://log_backend;
    proxy_pass_request_body off;
    proxy_set_header Content-Length "";
    proxy_set_header X-Original-URI $request_uri;
}
```

MP4

该模块为 MP4 文件提供伪流媒体的服务器端支持。这类文件通常具有.mp4、.m4v 或.m4a 文件扩展名。

伪流媒体与兼容的媒体播放器协同工作。播放器向服务器发送一个 HTTP 请求，请求中指定查询字符串参数的开始时间（简单命名为 start，单位为秒），服务器则回应以使其起始位置对应于请求的时间，例如：

```
http://example.com/elephants_dream.mp4?start=238.88
```

这允许在任意时间进行随机寻址，或从时间轴中间开始播放。

为了支持寻址，基于 H.264 的格式在所谓的"moov atom" 中存储元数据。它是文件的一部分，包含整个文件的索引信息。

为了开始播放，播放器首先需要读取元数据。这是通过发送一个带有 `start=0` 参数的特殊请求来完成的。许多编码软件将元数据插入到文件的末尾。这对于伪流媒体来说是不理想的，因为播放器必须在开始播放之前下载整个文件。如果元数据位于文件的开头，对于 Angie 来说，只需开始发送文件内容即可。如果元数据位于文件的末尾，Angie 必须读取整个文件并准备一个新的流，以便元数据在媒体数据之前。这涉及一些 CPU、内存和磁盘 I/O 的开销，因此最好提前 准备 原始文件以进行伪流媒体，而不是让 Angie 在每个这样的请求上都执行此操作。

该模块还支持 HTTP 请求的 `end` 参数，该参数设置播放的结束点。`end` 参数可以与 start 参数一起指定，也可以单独指定：

```
http://example.com/elephants_dream.mp4?start=238.88&end=555.55
```

对于包含非零 `start` 或 `end` 参数的匹配请求，Angie 将从文件中读取元数据，准备请求时间范围的流，并将其发送给客户端。这与上述描述的开销相同。

如果 `start` 参数指向非关键视频帧，则视频的开头将会损坏。为了解决此问题，视频可以 在 `start` 点之前添加关键帧以及它们之间的所有中间帧。这些帧将通过编辑列表从播放中隐藏。

如果匹配请求不包含 `start` 和 `end` 参数，则没有开销，文件将简单地作为静态资源发送。一些播放器也支持字节范围请求，因此不需要此模块。

当 从源代码构建时，该模块默认并不构建；应通过 `--with-http_mp4_module` 构建选项启用它。在来自我们的仓库的软件包和镜像中，该模块已包含在构建中。

> **小心**
>
> 如果之前使用了第三方的 mp4 模块，则应将其禁用。

对于 FLV 文件，提供了类似的伪流媒体支持：*FLV* 模块。

配置示例

```
location /video/ {
    mp4;
    mp4_buffer_size     1m;
    mp4_max_buffer_size 5m;
}
```

指令

mp4

语法	mp4;
默认	—
上下文	location

在周围位置启用模块处理。

mp4_buffer_size

语法	mp4_buffer_size *size*;
默认	mp4_buffer_size 512K;
上下文	http, server, location

设置用于处理 MP4 文件的缓冲区的初始大小。

mp4_max_buffer_size

语法	mp4_max_buffer_size *size*;
默认	mp4_max_buffer_size 10M;
上下文	http, server, location

在元数据处理期间，可能需要更大的缓冲区。其大小不得超过指定大小，否则 Angie 将返回 500（内部服务器错误）服务器错误，并记录以下消息：

> "/some/movie/file.mp4" mp4 moov atom is too large: 12583268, you may want to increase mp4_max_buffer_size

mp4_limit_rate

语法	`mp4_limit_rate on | off |` *factor*;
默认	`mp4_limit_rate off;`
上下文	http, server, location

对请求的 MP4 文件的传输进行速率限制。要计算限制，将 *factor* 乘以文件的平均比特率。

- `off` 值禁用速率限制。
- `on` 值设置 *factor* 为 `1.1`。
- 限制在达到由*mp4_limit_rate_after* 设置的值后应用。

请求是单独进行速率限制的：如果客户端打开两个连接，结果速率翻倍。因此，考虑使用*limit_conn* 和随附指令。

mp4_limit_rate_after

语法	`mp4_limit_rate_after` *time*;
默认	`mp4_limit_rate_after 60s;`
上下文	http, server, location

设置（以播放时间 为单位）传输的媒体数据量触发由*mp4_limit_rate* 设置的速率限制。

mp4_start_key_frame

语法	`mp4_start_key_frame on | off;`
默认	`mp4_start_key_frame off;`
上下文	http, server, location

强制输出视频始终以关键视频帧开始。如果开始参数不指向关键帧，则初始帧将通过 mp4 编辑列表隐藏。编辑列表受到主要播放器和浏览器的支持，如 Chrome、Safari、QuickTime 和 ffmpeg，Firefox 部分支持。

Perl

该模块用于在 Perl 中实现位置和变量处理器，并将 Perl 调用插入到 SSI 中。

当 从源代码构建时，该模块默认不构建；应通过 `--with-http_perl_module` 构建选项启用。

在我们的代码库中,该模块是 动态构建并作为一个名为 `angie-module-perl` 或 `angie-pro-module-perl` 的单独软件包提供。

重要

此模块需要 Perl 版本 5.6.1 或更高版本。C 编译器应与用于构建 Perl 的编译器兼容。

已知问题

该模块是实验性的，使用者需自担风险。

为了让 Perl 在重新配置期间重新编译修改过的模块，应使用 `-Dusemultiplicity=yes` 或 `-Dusethreads=yes` 参数构建。此外，为了减少 Perl 在运行时的内存泄漏，应使用 `-Dusemymalloc=no` 参数进行构建。要检查已构建 Perl 的这些参数的值（示例中指定了首选值），请运行：

```
$ perl -V:usemultiplicity -V:usemymalloc
usemultiplicity='define';
usemymalloc='n';
```

请注意，在使用新的 `-Dusemultiplicity=yes` 或 `-Dusethreads=yes` 参数重新构建 Perl 后，所有二进制 Perl 模块也必须重新构建——它们将不再与新的 Perl 兼容。

主进程和工作进程的大小在每次重新配置后可能会增长。如果主进程增长到不可接受的大小，可以在不更改可执行文件的情况下应用实时升级 程序。

当 Perl 模块执行长时间运行的操作时，例如解析域名、连接到其他服务器或查询数据库，分配给当前工作进程的其他请求将不会被处理。因此，建议仅执行具有可预测且短执行时间的操作，例如访问本地文件系统。

配置示例

```
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 模块：

```
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/>");

    if (-f $r->filename or -d _) {
        $r->print($r->uri, " exists!\n");
    }

    return OK;
}

1;
__END__
```

指令

perl

语法	`perl` *module* :: *function* | 'sub { ... }';
默认	—
上下文	location, limit_except

为给定位置设置一个 Perl 处理器。

perl_modules

语法	`perl_modules` *path*;
默认	—
上下文	http

为 Perl 模块设置额外路径。

perl_require

语法	`perl_require` *module*;
默认	—
上下文	http

定义将在每次重新配置时加载的模块名称。可以存在多个 *perl_require* 指令。

perl_set

语法	`perl_set` *$variable module* :: *function* | 'sub { ... }';
默认	—
上下文	http

为指定变量安装一个 Perl 处理器。

从 SSI 调用 Perl

调用 Perl 的 SSI 命令具有以下格式：

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

$r 请求对象方法

$r->args

返回请求参数。

$r->filename

返回与请求 URI 对应的文件名。

$r->has_request_body (*handler*)

如果请求中没有主体，则返回 0。如果有主体，则为请求设置指定的处理器并返回 1。在读取请求主体后，Angie 将调用指定的处理器。请注意，处理器函数应以引用方式传递。示例：

```
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 HTTP_BAD_REQUEST;
}

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->allow_ranges`

启用在发送响应时使用字节范围。

`$r->discard_request_body`

指示 Angie 丢弃请求主体。

`$r->header_in` (*field*)

返回指定客户端请求头字段的值。

`$r->header_only`

确定是否应将整个响应或仅其头部发送到客户端。

`$r->header_out` (*field, value*)

为指定响应头字段设置一个值。

`$r->internal_redirect (uri)`

对指定的 uri 进行内部重定向。实际重定向在 Perl 处理器执行完成后发生。该方法接受转义的 URI，并支持重定向到命名位置。

`$r->log_error (errno, message)`

将指定消息写入*error_log*。如果 *errno* 非零，错误代码及其描述将附加到消息中。

`$r->print (text, ...)`

将数据传递给客户端。

`$r->request_body`

如果客户端请求主体尚未写入临时文件，则返回客户端请求主体。为了确保客户端请求主体在内存中，其大小应通过*client_max_body_size* 限制，并使用*client_body_buffer_size* 设置足够的缓冲区大小。

`$r->request_body_file`

返回包含客户端请求主体的文件名称。处理后，应删除该文件。要始终将请求主体写入文件，应启用*client_body_in_file_only*。

`$r->request_method`

返回客户端请求的 HTTP 方法。

`$r->remote_addr`

返回客户端 IP 地址。

`$r->flush`

立即将数据发送给客户端。

`$r->sendfile (name [, offset [, length ]])`

将指定文件内容发送给客户端。可选参数指定数据传输的初始偏移量和长度。实际数据传输在 Perl 处理器完成后发生。

`$r->send_http_header ([type])`

将响应头发送给客户端。可选的类型参数设置”Content-Type” 响应头字段的值。如果值为空字符串，则将不发送”Content-Type” 头字段。

$r->status (*code*)

设置响应代码。

$r->sleep (*milliseconds, handler*)

设置指定的处理器，并在指定时间内停止请求处理。在此期间，Angie 继续处理其他请求。在指定时间经过后，Angie 将调用安装的处理器。请注意，处理器函数应以引用方式传递。为了在处理器之间传递数据，应使用*$r->variable()*。示例：

```
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->unescape (*text*)

解码以"%XX" 形式编码的文本。

`$r->uri`

返回请求 URI。

`$r->variable` (*name* [, *value*])

返回或设置指定变量的值。变量对每个请求都是本地的。

Prometheus

收集 Angie *metrics*，使用配置定义的模板，并以 Prometheus 格式返回由模板生成的指标。

> **⚠ 注意**
>
> 要收集这些指标，请在适当的上下文中使用以下指令启用共享内存区域：
>
> - 在*http_upstream* 或*stream_upstream* 中使用 `zone` 指令；
> - 使用*status_zone* 指令；
> - 在*resolver* 指令中使用 `status_zone` 参数。

配置示例

三个指标收集服务器共享内存区域的请求统计信息，合并到 `custom` 模板中，并在 `/p8s` 暴露：

```
http {

    prometheus_template custom {
        'angie_http_server_zones_requests_total{zone="$1"}' $p8s_value
            path=~^/http/server_zones/([^/]+)/requests/total$
            type=counter;

        'angie_http_server_zones_requests_processing{zone="$1"}' $p8s_value
            path=~^/http/server_zones/([^/]+)/requests/processing$
            type=gauge;

        'angie_http_server_zones_requests_discarded{zone="$1"}' $p8s_value
            path=~^/http/server_zones/([^/]+)/requests/discarded$
            type=counter;
    }

    # ...

    server {

        listen 80;

        location =/p8s {
            prometheus custom;
```

```
        }

        # ...

    }
}
```

Angie 有一个补充的 `prometheus_all.conf` 文件，定义了一些常用的指标，用作 `all` 模板：

文件内容 (Angie)

```
prometheus_template all {

angie_connections_accepted $p8s_value
    path=/connections/accepted
    type=counter
    'help=The total number of accepted client connections.';

angie_connections_dropped $p8s_value
    path=/connections/dropped
    type=counter
    'help=The total number of dropped client connections.';

angie_connections_active $p8s_value
    path=/connections/active
    type=gauge
    'help=The current number of active client connections.';

angie_connections_idle $p8s_value
    path=/connections/idle
    type=gauge
    'help=The current number of idle client connections.';


'angie_slabs_pages_used{zone="$1"}' $p8s_value
    path=~^/slabs/([^/]+)/pages/used$
    type=gauge
    'help=The number of currently used memory pages in a slab zone.';

'angie_slabs_pages_free{zone="$1"}' $p8s_value
    path=~^/slabs/([^/]+)/pages/free$
    type=gauge
    'help=The number of currently free memory pages in a slab zone.';


'angie_slabs_pages_slots_used{zone="$1",size="$2"}' $p8s_value
    path=~^/slabs/([^/]+)/slots/([^/]+)/used$
    type=gauge
```

```
    'help=The number of currently used memory slots of a specific size in a slab zone.';

'angie_slabs_pages_slots_free{zone="$1",size="$2"}' $p8s_value
    path=~^/slabs/([^/]+)/slots/([^/]+)/free$
    type=gauge
    'help=The number of currently free memory slots of a specific size in a slab zone.';

'angie_slabs_pages_slots_reqs{zone="$1",size="$2"}' $p8s_value
    path=~^/slabs/([^/]+)/slots/([^/]+)/reqs$
    type=counter
    'help=The total number of attempts to allocate a memory slot of a specific size in a slab␣
↪zone.';

'angie_slabs_pages_slots_fails{zone="$1",size="$2"}' $p8s_value
    path=~^/slabs/([^/]+)/slots/([^/]+)/fails$
    type=counter
    'help=The number of unsuccessful attempts to allocate a memory slot of a specific size in␣
↪a slab zone.';


'angie_resolvers_queries{zone="$1",type="$2"}' $p8s_value
    path=~^/resolvers/([^/]+)/queries/([^/]+)$
    type=counter
    'help=The number of queries of a specific type to resolve in a resolver zone.';

'angie_resolvers_sent{zone="$1",type="$2"}' $p8s_value
    path=~^/resolvers/([^/]+)/sent/([^/]+)$
    type=counter
    'help=The number of sent DNS queries of a specific type to resolve in a resolver zone.';

'angie_resolvers_responses{zone="$1",status="$2"}' $p8s_value
    path=~^/resolvers/([^/]+)/responses/([^/]+)$
    type=counter
    'help=The number of resolution results with a specific status in a resolver zone.';


'angie_http_server_zones_ssl_handshaked{zone="$1"}' $p8s_value
    path=~^/http/server_zones/([^/]+)/ssl/handshaked$
    type=counter
    'help=The total number of successful SSL handshakes in an HTTP server zone.';

'angie_http_server_zones_ssl_reuses{zone="$1"}' $p8s_value
    path=~^/http/server_zones/([^/]+)/ssl/reuses$
    type=counter
    'help=The total number of session reuses during SSL handshakes in an HTTP server zone.';

'angie_http_server_zones_ssl_timedout{zone="$1"}' $p8s_value
    path=~^/http/server_zones/([^/]+)/ssl/timedout$
```

```
    type=counter
    'help=The total number of timed-out SSL handshakes in an HTTP server zone.';

'angie_http_server_zones_ssl_failed{zone="$1"}' $p8s_value
    path=~^/http/server_zones/([^/]+)/ssl/failed$
    type=counter
    'help=The total number of failed SSL handshakes in an HTTP server zone.';


'angie_http_server_zones_requests_total{zone="$1"}' $p8s_value
    path=~^/http/server_zones/([^/]+)/requests/total$
    type=counter
    'help=The total number of client requests received in an HTTP server zone.';

'angie_http_server_zones_requests_processing{zone="$1"}' $p8s_value
    path=~^/http/server_zones/([^/]+)/requests/processing$
    type=gauge
    'help=The number of client requests currently being processed in an HTTP server zone.';

'angie_http_server_zones_requests_discarded{zone="$1"}' $p8s_value
    path=~^/http/server_zones/([^/]+)/requests/discarded$
    type=counter
    'help=The total number of client requests completed in an HTTP server zone without sending␣
↪a response.';


'angie_http_server_zones_responses{zone="$1",code="$2"}' $p8s_value
    path=~^/http/server_zones/([^/]+)/responses/([^/]+)$
    type=counter
    'help=The number of responses with a specific status in an HTTP server zone.';


'angie_http_server_zones_data_received{zone="$1"}' $p8s_value
    path=~^/http/server_zones/([^/]+)/data/received$
    type=counter
    'help=The total number of bytes received from clients in an HTTP server zone.';

'angie_http_server_zones_data_sent{zone="$1"}' $p8s_value
    path=~^/http/server_zones/([^/]+)/data/sent$
    type=counter
    'help=The total number of bytes sent to clients in an HTTP server zone.';


'angie_http_location_zones_requests_total{zone="$1"}' $p8s_value
    path=~^/http/location_zones/([^/]+)/requests/total$
    type=counter
    'help=The total number of client requests in an HTTP location zone.';
```

```
'angie_http_location_zones_requests_discarded{zone="$1"}' $p8s_value
    path=~^/http/location_zones/([^/]+)/requests/discarded$
    type=counter
    'help=The total number of client requests completed in an HTTP location zone without␣
↪sending a response.';


'angie_http_location_zones_responses{zone="$1",code="$2"}' $p8s_value
    path=~^/http/location_zones/([^/]+)/responses/([^/]+)$
    type=counter
    'help=The number of responses with a specific status in an HTTP location zone.';


'angie_http_location_zones_data_received{zone="$1"}' $p8s_value
    path=~^/http/location_zones/([^/]+)/data/received$
    type=counter
    'help=The total number of bytes received from clients in an HTTP location zone.';

'angie_http_location_zones_data_sent{zone="$1"}' $p8s_value
    path=~^/http/location_zones/([^/]+)/data/sent$
    type=counter
    'help=The total number of bytes sent to clients in an HTTP location zone.';


'angie_http_upstreams_peers_state{upstream="$1",peer="$2"}' $p8st_all_ups_state
    path=~^/http/upstreams/([^/]+)/peers/([^/]+)/state$
    type=gauge
    'help=The current state of an upstream peer in "HTTP": 1 - up, 2 - down, 3 - unavailable,␣
↪or 4 - recovering.';


'angie_http_upstreams_peers_selected_current{upstream="$1",peer="$2"}' $p8s_value
    path=~^/http/upstreams/([^/]+)/peers/([^/]+)/selected/current$
    type=gauge
    'help=The number of requests currently being processed by an upstream peer in "HTTP".';

'angie_http_upstreams_peers_selected_total{upstream="$1",peer="$2"}' $p8s_value
    path=~^/http/upstreams/([^/]+)/peers/([^/]+)/selected/total$
    type=counter
    'help=The total number of attempts to use an upstream peer in "HTTP".';


'angie_http_upstreams_peers_responses{upstream="$1",peer="$2",code="$3"}' $p8s_value
    path=~^/http/upstreams/([^/]+)/peers/([^/]+)/responses/([^/]+)$
    type=counter
    'help=The number of responses with a specific status received from an upstream peer in
↪"HTTP".';
```

```
'angie_http_upstreams_peers_data_sent{upstream="$1",peer="$2"}' $p8s_value
    path=~^/http/upstreams/([^/]+)/peers/([^/]+)/data/sent$
    type=counter
    'help=The total number of bytes sent to an upstream peer in "HTTP".';

'angie_http_upstreams_peers_data_received{upstream="$1",peer="$2"}' $p8s_value
    path=~^/http/upstreams/([^/]+)/peers/([^/]+)/data/received$
    type=counter
    'help=The total number of bytes received from an upstream peer in "HTTP".';


'angie_http_upstreams_peers_health_fails{upstream="$1",peer="$2"}' $p8s_value
    path=~^/http/upstreams/([^/]+)/peers/([^/]+)/health/fails$
    type=counter
    'help=The total number of unsuccessful attempts to communicate with an upstream peer in
↪"HTTP".';

'angie_http_upstreams_peers_health_unavailable{upstream="$1",peer="$2"}' $p8s_value
    path=~^/http/upstreams/([^/]+)/peers/([^/]+)/health/unavailable$
    type=counter
    'help=The number of times when an upstream peer in "HTTP" became "unavailable" due to␣
↪reaching the max_fails limit.';

'angie_http_upstreams_peers_health_downtime{upstream="$1",peer="$2"}' $p8s_value
    path=~^/http/upstreams/([^/]+)/peers/([^/]+)/health/downtime$
    type=counter
    'help=The total time (in milliseconds) that an upstream peer in "HTTP" was "unavailable".';


'angie_http_upstreams_keepalive{upstream="$1"}' $p8s_value
    path=~^/http/upstreams/([^/]+)/keepalive$
    type=gauge
    'help=The number of currently cached keepalive connections for an HTTP upstream.';


'angie_http_caches_responses{zone="$1",status="$2"}' $p8s_value
    path=~^/http/caches/([^/]+)/([^/]+)/responses$
    type=counter
    'help=The total number of responses processed in an HTTP cache zone with a specific cache␣
↪status.';

'angie_http_caches_bytes{zone="$1",status="$2"}' $p8s_value
    path=~^/http/caches/([^/]+)/([^/]+)/bytes$
    type=counter
    'help=The total number of bytes processed in an HTTP cache zone with a specific cache␣
↪status.';
```

```
'angie_http_caches_responses_written{zone="$1",status="$2"}' $p8s_value
    path=~^/http/caches/([^/]+)/([^/]+)/responses_written$
    type=counter
    'help=The total number of responses written to an HTTP cache zone with a specific cache␣
↪status.';

'angie_http_caches_bytes_written{zone="$1",status="$2"}' $p8s_value
    path=~^/http/caches/([^/]+)/([^/]+)/bytes_written$
    type=counter
    'help=The total number of bytes written to an HTTP cache zone with a specific cache status.
↪';


'angie_http_caches_size{zone="$1"}' $p8s_value
    path=~^/http/caches/([^/]+)/size$
    type=gauge
    'help=The current size (in bytes) of cached responses in an HTTP cache zone.';


'angie_http_caches_shards_size{zone="$1",path="$2"}' $p8s_value
    path=~^/http/caches/([^/]+)/shards/([^/]+)/size$
    type=gauge
    'help=The current size (in bytes) of cached responses in a shard path of an HTTP cache␣
↪zone.';


'angie_http_limit_conns{zone="$1",status="$2"}' $p8s_value
    path=~^/http/limit_conns/([^/]+)/([^/]+)$
    type=counter
    'help=The number of requests processed by an HTTP limit_conn zone with a specific result.';

'angie_http_limit_reqs{zone="$1",status="$2"}' $p8s_value
    path=~^/http/limit_reqs/([^/]+)/([^/]+)$
    type=counter
    'help=The number of requests processed by an HTTP limit_reqs zone with a specific result.';


'angie_stream_server_zones_ssl_handshaked{zone="$1"}' $p8s_value
    path=~^/stream/server_zones/([^/]+)/ssl/handshaked$
    type=counter
    'help=The total number of successful SSL handshakes in a stream server zone.';

'angie_stream_server_zones_ssl_reuses{zone="$1"}' $p8s_value
    path=~^/stream/server_zones/([^/]+)/ssl/reuses$
    type=counter
    'help=The total number of session reuses during SSL handshakes in a stream server zone.';

'angie_stream_server_zones_ssl_timedout{zone="$1"}' $p8s_value
```

```
    path=~^/stream/server_zones/([^/]+)/ssl/timedout$
    type=counter
    'help=The total number of timed-out SSL handshakes in a stream server zone.';

'angie_stream_server_zones_ssl_failed{zone="$1"}' $p8s_value
    path=~^/stream/server_zones/([^/]+)/ssl/failed$
    type=counter
    'help=The total number of failed SSL handshakes in a stream server zone.';


'angie_stream_server_zones_connections_total{zone="$1"}' $p8s_value
    path=~^/stream/server_zones/([^/]+)/connections/total$
    type=counter
    'help=The total number of client connections received in a stream server zone.';

'angie_stream_server_zones_connections_processing{zone="$1"}' $p8s_value
    path=~^/stream/server_zones/([^/]+)/connections/processing$
    type=gauge
    'help=The number of client connections currently being processed in a stream server zone.';

'angie_stream_server_zones_connections_discarded{zone="$1"}' $p8s_value
    path=~^/stream/server_zones/([^/]+)/connections/discarded$
    type=counter
    'help=The total number of client connections completed in a stream server zone without␣
↪establishing a session.';

'angie_stream_server_zones_connections_passed{zone="$1"}' $p8s_value
    path=~^/stream/server_zones/([^/]+)/connections/passed$
    type=counter
    'help=The total number of client connections in a stream server zone passed for handling␣
↪to a different listening socket.';


'angie_stream_server_zones_sessions{zone="$1",status="$2"}' $p8s_value
    path=~^/stream/server_zones/([^/]+)/sessions/([^/]+)$
    type=counter
    'help=The number of sessions finished with a specific status in a stream server zone.';


'angie_stream_server_zones_data_received{zone="$1"}' $p8s_value
    path=~^/stream/server_zones/([^/]+)/data/received$
    type=counter
    'help=The total number of bytes received from clients in a stream server zone.';

'angie_stream_server_zones_data_sent{zone="$1"}' $p8s_value
    path=~^/stream/server_zones/([^/]+)/data/sent$
    type=counter
    'help=The total number of bytes sent to clients in a stream server zone.';
```

```
'angie_stream_upstreams_peers_state{upstream="$1",peer="$2"}' $p8st_all_ups_state
    path=~^/stream/upstreams/([^/]+)/peers/([^/]+)/state$
    type=gauge
    'help=The current state of an upstream peer in "stream": 1 - up, 2 - down, 3 - unavailable,
↪ or 4 - recovering.';


'angie_stream_upstreams_peers_selected_current{upstream="$1",peer="$2"}' $p8s_value
    path=~^/stream/upstreams/([^/]+)/peers/([^/]+)/selected/current$
    type=gauge
    'help=The number of sessions currently being processed by an upstream peer in "stream".';

'angie_stream_upstreams_peers_selected_total{upstream="$1",peer="$2"}' $p8s_value
    path=~^/stream/upstreams/([^/]+)/peers/([^/]+)/selected/total$
    type=counter
    'help=The total number of attempts to use an upstream peer in "stream".';


'angie_stream_upstreams_peers_data_sent{upstream="$1",peer="$2"}' $p8s_value
    path=~^/stream/upstreams/([^/]+)/peers/([^/]+)/data/sent$
    type=counter
    'help=The total number of bytes sent to an upstream peer in "stream".';

'angie_stream_upstreams_peers_data_received{upstream="$1",peer="$2"}' $p8s_value
    path=~^/stream/upstreams/([^/]+)/peers/([^/]+)/data/received$
    type=counter
    'help=The total number of bytes received from an upstream peer in "stream".';


'angie_stream_upstreams_peers_health_fails{upstream="$1",peer="$2"}' $p8s_value
    path=~^/stream/upstreams/([^/]+)/peers/([^/]+)/health/fails$
    type=counter
    'help=The total number of unsuccessful attempts to communicate with an upstream peer in
↪"stream".';

'angie_stream_upstreams_peers_health_unavailable{upstream="$1",peer="$2"}' $p8s_value
    path=~^/stream/upstreams/([^/]+)/peers/([^/]+)/health/unavailable$
    type=counter
    'help=The number of times when an upstream peer in "stream" became "unavailable" due to␣
↪reaching the max_fails limit.';

'angie_stream_upstreams_peers_health_downtime{upstream="$1",peer="$2"}' $p8s_value
    path=~^/stream/upstreams/([^/]+)/peers/([^/]+)/health/downtime$
    type=counter
    'help=The total time (in milliseconds) that an upstream peer in "stream" was "unavailable".
↪';
```

```
}

map $p8s_value $p8st_all_ups_state {
    volatile;
    "up"           1;
    "down"         2;
    "unavailable"  3;
    "recovering"   4;
#    "unhealthy"    5;
#    "checking"     6;
#    "draining"     7;
    "busy"         8;
    default        0;
}
```

文件内容 (Angie PRO)

```
prometheus_template all {

angie_connections_accepted $p8s_value
    path=/connections/accepted
    type=counter
    'help=The total number of accepted client connections.';

angie_connections_dropped $p8s_value
    path=/connections/dropped
    type=counter
    'help=The total number of dropped client connections.';

angie_connections_active $p8s_value
    path=/connections/active
    type=gauge
    'help=The current number of active client connections.';

angie_connections_idle $p8s_value
    path=/connections/idle
    type=gauge
    'help=The current number of idle client connections.';


'angie_slabs_pages_used{zone="$1"}' $p8s_value
    path=~^/slabs/([^/]+)/pages/used$
    type=gauge
    'help=The number of currently used memory pages in a slab zone.';

'angie_slabs_pages_free{zone="$1"}' $p8s_value
    path=~^/slabs/([^/]+)/pages/free$
    type=gauge
```

```
    'help=The number of currently free memory pages in a slab zone.';


'angie_slabs_pages_slots_used{zone="$1",size="$2"}' $p8s_value
    path=~^/slabs/([^/]+)/slots/([^/]+)/used$
    type=gauge
    'help=The number of currently used memory slots of a specific size in a slab zone.';

'angie_slabs_pages_slots_free{zone="$1",size="$2"}' $p8s_value
    path=~^/slabs/([^/]+)/slots/([^/]+)/free$
    type=gauge
    'help=The number of currently free memory slots of a specific size in a slab zone.';

'angie_slabs_pages_slots_reqs{zone="$1",size="$2"}' $p8s_value
    path=~^/slabs/([^/]+)/slots/([^/]+)/reqs$
    type=counter
    'help=The total number of attempts to allocate a memory slot of a specific size in a slab␣
↪zone.';

'angie_slabs_pages_slots_fails{zone="$1",size="$2"}' $p8s_value
    path=~^/slabs/([^/]+)/slots/([^/]+)/fails$
    type=counter
    'help=The number of unsuccessful attempts to allocate a memory slot of a specific size in␣
↪a slab zone.';


'angie_resolvers_queries{zone="$1",type="$2"}' $p8s_value
    path=~^/resolvers/([^/]+)/queries/([^/]+)$
    type=counter
    'help=The number of queries of a specific type to resolve in a resolver zone.';

'angie_resolvers_sent{zone="$1",type="$2"}' $p8s_value
    path=~^/resolvers/([^/]+)/sent/([^/]+)$
    type=counter
    'help=The number of sent DNS queries of a specific type to resolve in a resolver zone.';

'angie_resolvers_responses{zone="$1",status="$2"}' $p8s_value
    path=~^/resolvers/([^/]+)/responses/([^/]+)$
    type=counter
    'help=The number of resolution results with a specific status in a resolver zone.';


'angie_http_server_zones_ssl_handshaked{zone="$1"}' $p8s_value
    path=~^/http/server_zones/([^/]+)/ssl/handshaked$
    type=counter
    'help=The total number of successful SSL handshakes in an HTTP server zone.';

'angie_http_server_zones_ssl_reuses{zone="$1"}' $p8s_value
```

```
    path=~^/http/server_zones/([^/]+)/ssl/reuses$
    type=counter
    'help=The total number of session reuses during SSL handshakes in an HTTP server zone.';

'angie_http_server_zones_ssl_timedout{zone="$1"}' $p8s_value
    path=~^/http/server_zones/([^/]+)/ssl/timedout$
    type=counter
    'help=The total number of timed-out SSL handshakes in an HTTP server zone.';

'angie_http_server_zones_ssl_failed{zone="$1"}' $p8s_value
    path=~^/http/server_zones/([^/]+)/ssl/failed$
    type=counter
    'help=The total number of failed SSL handshakes in an HTTP server zone.';


'angie_http_server_zones_requests_total{zone="$1"}' $p8s_value
    path=~^/http/server_zones/([^/]+)/requests/total$
    type=counter
    'help=The total number of client requests received in an HTTP server zone.';

'angie_http_server_zones_requests_processing{zone="$1"}' $p8s_value
    path=~^/http/server_zones/([^/]+)/requests/processing$
    type=gauge
    'help=The number of client requests currently being processed in an HTTP server zone.';

'angie_http_server_zones_requests_discarded{zone="$1"}' $p8s_value
    path=~^/http/server_zones/([^/]+)/requests/discarded$
    type=counter
    'help=The total number of client requests completed in an HTTP server zone without sending␣
↪a response.';


'angie_http_server_zones_responses{zone="$1",code="$2"}' $p8s_value
    path=~^/http/server_zones/([^/]+)/responses/([^/]+)$
    type=counter
    'help=The number of responses with a specific status in an HTTP server zone.';


'angie_http_server_zones_data_received{zone="$1"}' $p8s_value
    path=~^/http/server_zones/([^/]+)/data/received$
    type=counter
    'help=The total number of bytes received from clients in an HTTP server zone.';

'angie_http_server_zones_data_sent{zone="$1"}' $p8s_value
    path=~^/http/server_zones/([^/]+)/data/sent$
    type=counter
    'help=The total number of bytes sent to clients in an HTTP server zone.';
```

```
'angie_http_location_zones_requests_total{zone="$1"}' $p8s_value
    path=~^/http/location_zones/([^/]+)/requests/total$
    type=counter
    'help=The total number of client requests in an HTTP location zone.';

'angie_http_location_zones_requests_discarded{zone="$1"}' $p8s_value
    path=~^/http/location_zones/([^/]+)/requests/discarded$
    type=counter
    'help=The total number of client requests completed in an HTTP location zone without␣
↪sending a response.';


'angie_http_location_zones_responses{zone="$1",code="$2"}' $p8s_value
    path=~^/http/location_zones/([^/]+)/responses/([^/]+)$
    type=counter
    'help=The number of responses with a specific status in an HTTP location zone.';


'angie_http_location_zones_data_received{zone="$1"}' $p8s_value
    path=~^/http/location_zones/([^/]+)/data/received$
    type=counter
    'help=The total number of bytes received from clients in an HTTP location zone.';

'angie_http_location_zones_data_sent{zone="$1"}' $p8s_value
    path=~^/http/location_zones/([^/]+)/data/sent$
    type=counter
    'help=The total number of bytes sent to clients in an HTTP location zone.';


'angie_http_upstreams_peers_backup{upstream="$1",peer="$2"}' $p8st_all_ups_backup
    path=~^/http/upstreams/([^/]+)/peers/([^/]+)/backup$
    type=gauge
    'help=The HTTP upstream peer backup group level.';


'angie_http_upstreams_peers_state{upstream="$1",peer="$2"}' $p8st_all_ups_state
    path=~^/http/upstreams/([^/]+)/peers/([^/]+)/state$
    type=gauge
    'help=The current state of an upstream peer in "HTTP": 1 - up, 2 - down, 3 - unavailable,␣
↪4 - recovering, 5 - unhealthy, 6 - checking, or 7 - draining.';


'angie_http_upstreams_peers_selected_current{upstream="$1",peer="$2"}' $p8s_value
    path=~^/http/upstreams/([^/]+)/peers/([^/]+)/selected/current$
    type=gauge
    'help=The number of requests currently being processed by an upstream peer in "HTTP".';
```

```
'angie_http_upstreams_peers_selected_total{upstream="$1",peer="$2"}' $p8s_value
    path=~^/http/upstreams/([^/]+)/peers/([^/]+)/selected/total$
    type=counter
    'help=The total number of attempts to use an upstream peer in "HTTP".';


'angie_http_upstreams_peers_responses{upstream="$1",peer="$2",code="$3"}' $p8s_value
    path=~^/http/upstreams/([^/]+)/peers/([^/]+)/responses/([^/]+)$
    type=counter
    'help=The number of responses with a specific status received from an upstream peer in
↪"HTTP".';


'angie_http_upstreams_peers_data_sent{upstream="$1",peer="$2"}' $p8s_value
    path=~^/http/upstreams/([^/]+)/peers/([^/]+)/data/sent$
    type=counter
    'help=The total number of bytes sent to an upstream peer in "HTTP".';

'angie_http_upstreams_peers_data_received{upstream="$1",peer="$2"}' $p8s_value
    path=~^/http/upstreams/([^/]+)/peers/([^/]+)/data/received$
    type=counter
    'help=The total number of bytes received from an upstream peer in "HTTP".';


'angie_http_upstreams_peers_health_fails{upstream="$1",peer="$2"}' $p8s_value
    path=~^/http/upstreams/([^/]+)/peers/([^/]+)/health/fails$
    type=counter
    'help=The total number of unsuccessful attempts to communicate with an upstream peer in
↪"HTTP".';

'angie_http_upstreams_peers_health_unavailable{upstream="$1",peer="$2"}' $p8s_value
    path=~^/http/upstreams/([^/]+)/peers/([^/]+)/health/unavailable$
    type=counter
    'help=The number of times when an upstream peer in "HTTP" became "unavailable" due to␣
↪reaching the max_fails limit.';

'angie_http_upstreams_peers_health_downtime{upstream="$1",peer="$2"}' $p8s_value
    path=~^/http/upstreams/([^/]+)/peers/([^/]+)/health/downtime$
    type=counter
    'help=The total time (in milliseconds) that an upstream peer in "HTTP" was "unavailable".';

'angie_http_upstreams_peers_health_header_time{upstream="$1",peer="$2"}' $p8s_value
    path=~^/http/upstreams/([^/]+)/peers/([^/]+)/health/header_time$
    type=gauge
    'help=Average time (in milliseconds) to receive the response headers from an upstream peer␣
↪in "HTTP".';

'angie_http_upstreams_peers_health_response_time{upstream="$1",peer="$2"}' $p8s_value
```

```
    path=~^/http/upstreams/([^/]+)/peers/([^/]+)/health/response_time$
    type=gauge
    'help=Average time (in milliseconds) to receive the complete response from an upstream␣
↪peer in "HTTP".';

'angie_http_upstreams_peers_health_probes_count{upstream="$1",peer="$2"}' $p8s_value
    path=~^/http/upstreams/([^/]+)/peers/([^/]+)/health/probes/count$
    type=counter
    'help=The total number of probes for this peer.';

'angie_http_upstreams_peers_health_probes_fails{upstream="$1",peer="$2"}' $p8s_value
    path=~^/http/upstreams/([^/]+)/peers/([^/]+)/health/probes/fails$
    type=counter
    'help=The total number of failed probes for this peer.';


'angie_http_upstreams_keepalive{upstream="$1"}' $p8s_value
    path=~^/http/upstreams/([^/]+)/keepalive$
    type=gauge
    'help=The number of currently cached keepalive connections for an HTTP upstream.';


'angie_http_upstreams_backup_switch_active{upstream="$1"}' $p8s_value
    path=~^/http/upstreams/([^/]+)/backup_switch/active$
    type=gauge
    'help=The currently active HTTP upstream servers backup group level.';


'angie_http_upstreams_queue_queued{upstream="$1"}' $p8s_value
    path=~^/http/upstreams/([^/]+)/queue/queued$
    type=counter
    'help=The total number of queued requests for an HTTP upstream.';

'angie_http_upstreams_queue_waiting{upstream="$1"}' $p8s_value
    path=~^/http/upstreams/([^/]+)/queue/waiting$
    type=gauge
    'help=The number of requests currently waiting in an HTTP upstream queue.';

'angie_http_upstreams_queue_dropped{upstream="$1"}' $p8s_value
    path=~^/http/upstreams/([^/]+)/queue/dropped$
    type=counter
    'help=The total number of requests dropped from an HTTP upstream queue because the client␣
↪had prematurely closed the connection.';

'angie_http_upstreams_queue_timedout{upstream="$1"}' $p8s_value
    path=~^/http/upstreams/([^/]+)/queue/timedout$
    type=counter
    'help=The total number of requests timed out from an HTTP upstream queue.';
```

```
'angie_http_upstreams_queue_overflows{upstream="$1"}' $p8s_value
    path=~^/http/upstreams/([^/]+)/queue/overflows$
    type=counter
    'help=The total number of requests rejected by an HTTP upstream queue because the size␣
↪limit had been reached.';


'angie_http_caches_responses{zone="$1",status="$2"}' $p8s_value
    path=~^/http/caches/([^/]+)/([^/]+)/responses$
    type=counter
    'help=The total number of responses processed in an HTTP cache zone with a specific cache␣
↪status.';

'angie_http_caches_bytes{zone="$1",status="$2"}' $p8s_value
    path=~^/http/caches/([^/]+)/([^/]+)/bytes$
    type=counter
    'help=The total number of bytes processed in an HTTP cache zone with a specific cache␣
↪status.';

'angie_http_caches_responses_written{zone="$1",status="$2"}' $p8s_value
    path=~^/http/caches/([^/]+)/([^/]+)/responses_written$
    type=counter
    'help=The total number of responses written to an HTTP cache zone with a specific cache␣
↪status.';

'angie_http_caches_bytes_written{zone="$1",status="$2"}' $p8s_value
    path=~^/http/caches/([^/]+)/([^/]+)/bytes_written$
    type=counter
    'help=The total number of bytes written to an HTTP cache zone with a specific cache status.
↪';


'angie_http_caches_size{zone="$1"}' $p8s_value
    path=~^/http/caches/([^/]+)/size$
    type=gauge
    'help=The current size (in bytes) of cached responses in an HTTP cache zone.';


'angie_http_caches_shards_size{zone="$1",path="$2"}' $p8s_value
    path=~^/http/caches/([^/]+)/shards/([^/]+)/size$
    type=gauge
    'help=The current size (in bytes) of cached responses in a shard path of an HTTP cache␣
↪zone.';


'angie_http_limit_conns{zone="$1",status="$2"}' $p8s_value
    path=~^/http/limit_conns/([^/]+)/([^/]+)$
```

```
    type=counter
    'help=The number of requests processed by an HTTP limit_conn zone with a specific result.';

'angie_http_limit_reqs{zone="$1",status="$2"}' $p8s_value
    path=~^/http/limit_reqs/([^/]+)/([^/]+)$
    type=counter
    'help=The number of requests processed by an HTTP limit_reqs zone with a specific result.';


'angie_stream_server_zones_ssl_handshaked{zone="$1"}' $p8s_value
    path=~^/stream/server_zones/([^/]+)/ssl/handshaked$
    type=counter
    'help=The total number of successful SSL handshakes in a stream server zone.';

'angie_stream_server_zones_ssl_reuses{zone="$1"}' $p8s_value
    path=~^/stream/server_zones/([^/]+)/ssl/reuses$
    type=counter
    'help=The total number of session reuses during SSL handshakes in a stream server zone.';

'angie_stream_server_zones_ssl_timedout{zone="$1"}' $p8s_value
    path=~^/stream/server_zones/([^/]+)/ssl/timedout$
    type=counter
    'help=The total number of timed-out SSL handshakes in a stream server zone.';

'angie_stream_server_zones_ssl_failed{zone="$1"}' $p8s_value
    path=~^/stream/server_zones/([^/]+)/ssl/failed$
    type=counter
    'help=The total number of failed SSL handshakes in a stream server zone.';


'angie_stream_server_zones_connections_total{zone="$1"}' $p8s_value
    path=~^/stream/server_zones/([^/]+)/connections/total$
    type=counter
    'help=The total number of client connections received in a stream server zone.';

'angie_stream_server_zones_connections_processing{zone="$1"}' $p8s_value
    path=~^/stream/server_zones/([^/]+)/connections/processing$
    type=gauge
    'help=The number of client connections currently being processed in a stream server zone.';

'angie_stream_server_zones_connections_discarded{zone="$1"}' $p8s_value
    path=~^/stream/server_zones/([^/]+)/connections/discarded$
    type=counter
    'help=The total number of client connections completed in a stream server zone without␣
↪establishing a session.';

'angie_stream_server_zones_connections_passed{zone="$1"}' $p8s_value
    path=~^/stream/server_zones/([^/]+)/connections/passed$
```

```
    type=counter
    'help=The total number of client connections in a stream server zone passed for handling␣
↪to a different listening socket.';


'angie_stream_server_zones_sessions{zone="$1",status="$2"}' $p8s_value
    path=~^/stream/server_zones/([^/]+)/sessions/([^/]+)$
    type=counter
    'help=The number of sessions finished with a specific status in a stream server zone.';


'angie_stream_server_zones_data_received{zone="$1"}' $p8s_value
    path=~^/stream/server_zones/([^/]+)/data/received$
    type=counter
    'help=The total number of bytes received from clients in a stream server zone.';

'angie_stream_server_zones_data_sent{zone="$1"}' $p8s_value
    path=~^/stream/server_zones/([^/]+)/data/sent$
    type=counter
    'help=The total number of bytes sent to clients in a stream server zone.';


'angie_stream_upstreams_peers_backup{upstream="$1",peer="$2"}' $p8st_all_ups_backup
    path=~^/stream/upstreams/([^/]+)/peers/([^/]+)/backup$
    type=gauge
    'help=The "stream" upstream peer backup group level.';


'angie_stream_upstreams_peers_state{upstream="$1",peer="$2"}' $p8st_all_ups_state
    path=~^/stream/upstreams/([^/]+)/peers/([^/]+)/state$
    type=gauge
    'help=The current state of an upstream peer in "stream": 1 - up, 2 - down, 3 - unavailable,
↪ 4 - recovering, 5 - unhealthy, 6 - checking, or 7 - draining.';


'angie_stream_upstreams_peers_selected_current{upstream="$1",peer="$2"}' $p8s_value
    path=~^/stream/upstreams/([^/]+)/peers/([^/]+)/selected/current$
    type=gauge
    'help=The number of sessions currently being processed by an upstream peer in "stream".';

'angie_stream_upstreams_peers_selected_total{upstream="$1",peer="$2"}' $p8s_value
    path=~^/stream/upstreams/([^/]+)/peers/([^/]+)/selected/total$
    type=counter
    'help=The total number of attempts to use an upstream peer in "stream".';


'angie_stream_upstreams_peers_data_sent{upstream="$1",peer="$2"}' $p8s_value
    path=~^/stream/upstreams/([^/]+)/peers/([^/]+)/data/sent$
```

```
    type=counter
    'help=The total number of bytes sent to an upstream peer in "stream".';

'angie_stream_upstreams_peers_data_received{upstream="$1",peer="$2"}' $p8s_value
    path=~^/stream/upstreams/([^/]+)/peers/([^/]+)/data/received$
    type=counter
    'help=The total number of bytes received from an upstream peer in "stream".';

'angie_stream_upstreams_peers_data_pkt_sent{upstream="$1",peer="$2"}' $p8s_value
    path=~^/stream/upstreams/([^/]+)/peers/([^/]+)/data/pkt_sent$
    type=counter
    'help=The total number of packets sent to an upstream peer in "stream".';

'angie_stream_upstreams_peers_data_pkt_received{upstream="$1",peer="$2"}' $p8s_value
    path=~^/stream/upstreams/([^/]+)/peers/([^/]+)/data/pkt_received$
    type=counter
    'help=The total number of packets received from an upstream peer in "stream".';


'angie_stream_upstreams_peers_health_fails{upstream="$1",peer="$2"}' $p8s_value
    path=~^/stream/upstreams/([^/]+)/peers/([^/]+)/health/fails$
    type=counter
    'help=The total number of unsuccessful attempts to communicate with an upstream peer in
↪"stream".';

'angie_stream_upstreams_peers_health_unavailable{upstream="$1",peer="$2"}' $p8s_value
    path=~^/stream/upstreams/([^/]+)/peers/([^/]+)/health/unavailable$
    type=counter
    'help=The number of times when an upstream peer in "stream" became "unavailable" due to␣
↪reaching the max_fails limit.';

'angie_stream_upstreams_peers_health_downtime{upstream="$1",peer="$2"}' $p8s_value
    path=~^/stream/upstreams/([^/]+)/peers/([^/]+)/health/downtime$
    type=counter
    'help=The total time (in milliseconds) that an upstream peer in "stream" was "unavailable".
↪';

'angie_stream_upstreams_peers_health_connect_time{upstream="$1",peer="$2"}' $p8s_value
    path=~^/stream/upstreams/([^/]+)/peers/([^/]+)/health/connect_time$
    type=gauge
    'help=Average time (in milliseconds) to connect to an upstream peer in "stream".';

'angie_stream_upstreams_peers_health_first_byte_time{upstream="$1",peer="$2"}' $p8s_value
    path=~^/stream/upstreams/([^/]+)/peers/([^/]+)/health/first_byte_time$
    type=gauge
    'help=Average time (in milliseconds) to receive the first byte from an upstream peer in
↪"stream".';
```

```
'angie_stream_upstreams_peers_health_last_byte_time{upstream="$1",peer="$2"}' $p8s_value
    path=~^/stream/upstreams/([^/]+)/peers/([^/]+)/health/last_byte_time$
    type=gauge
    'help=Average time (in milliseconds) of the whole communication session with an upstream␣
↪peer in "stream".';


'angie_stream_upstreams_peers_health_probes_count{upstream="$1",peer="$2"}' $p8s_value
    path=~^/stream/upstreams/([^/]+)/peers/([^/]+)/health/probes/count$
    type=counter
    'help=The total number of probes for this peer.';

'angie_stream_upstreams_peers_health_probes_fails{upstream="$1",peer="$2"}' $p8s_value
    path=~^/stream/upstreams/([^/]+)/peers/([^/]+)/health/probes/fails$
    type=counter
    'help=The total number of failed probes for this peer.';


'angie_stream_upstreams_backup_switch_active{upstream="$1"}' $p8s_value
    path=~^/stream/upstreams/([^/]+)/backup_switch/active$
    type=gauge
    'help=The currently active "stream" upstream servers backup group level.';
}

map $p8s_value $p8st_all_ups_state {
    volatile;
    "up"           1;
    "down"         2;
    "unavailable"  3;
    "recovering"   4;
    "unhealthy"    5;
    "checking"     6;
    "draining"     7;
    "busy"         8;
    default        0;
}

map $p8s_value $p8st_all_ups_backup {
    volatile;
    "false"        0;
    "true"         1;
    default        $p8s_value;
}
```

使用方法:

```
http {
```

```
    include prometheus_all.conf;

    #...

    server {

        listen 80;

        location =/p8s {
            prometheus all;
        }

        # ...

    }
}
```

```
$ curl localhost/p8s

    # Angie Prometheus template "all"
    ...
```

指令

prometheus

Syntax	prometheus *template_name*;
默认	—
Context	location

为 location 上下文指定由*prometheus_template* 指令定义的模板处理程序。在请求时，此 location 计算并返回 Prometheus 格式的模板指标。

```
location =/p8s {
    prometheus custom;
}
```

```
$ curl localhost/p8s

    # Angie Prometheus template "custom"
    ...
```

prometheus_template

Syntax	`prometheus_template` *template_name* { ... }
默认	—
Context	http

定义 Angie 收集和导出的指标的命名模板，用于*prometheus* 指令。

备注

Angie 还包含一个现成的*all* 模板，其中包含一组最常用的指标。

可以包含任意数量的指标定义，每个定义的结构如下：*<metric_name>* *<variable>* [`path=`*<match_string>*] [`type=`*<type>*] [`help=`*<help>*]。

`metric_name`	设置指标名称，该名称将以 Prometheus 格式添加到响应中。可以包含可选的标签部分 (...)，例如： `http_requests_total{method="$1",code="$2"}` 标签值可以使用 Angie 变量；如果 *match_string* 定义为正则表达式，还可以使用该表达式中定义的捕获组。在获取由 *variable* 设置的指标值时，会评估这些变量和组。
`variable`	设置将被评估并作为指标值添加到响应中的变量名称。如果变量不存在或评估结果为空 (`""`)，则不会添加指标。

指标使用 *variable* 设置的值计算；成功评估后，指标将添加到响应中，例如：

```
'angie_time{version="$angie_version"}' $msec;
```

```
$ curl localhost/p8s

   angie_time{version="1.10.0"} 1695119820.562
```

`path=match_strin`	与 Angie API */status* 子树中所有指标的端点路径匹配，允许一次将多个指标实例添加到响应中。

在匹配过程中，路径带有前导斜杠但没有尾部斜杠，例如 `/angie/generation`；匹配不区分大小写。有两种匹配方法：

`path=exact_match`	通过逐字符比较检查。
`path=~regular_ex`	使用 PCRE 库检查；可以定义捕获组，用于 *metric_name* 字段的标签中。

如果 *match_string* 匹配任何路径，该路径上的 Angie 指标值将存储在*$p8s_value* 变量中，该变量可在指定 `path=` 时用于 *variable* 字段。

对于正则表达式，可能有多个匹配路径；对于 * 每个 * 匹配，指标都会添加到响应中。结合捕获组，这允许获取一系列具有相同名称但不同标签的指标，例如：

```
'angie_slabs_slots_free{zone="$1",size="$2"}' $p8s_value
    path=~^/slabs/([^/]+)/slots/([^/]+)/free$;
```

此定义为当前配置中存在的所有区域和所有大小添加指标：

```
angie_slabs_slots_free{zone="one",size="8"} 502
angie_slabs_slots_free{zone="one",size="16"} 249
angie_slabs_slots_free{zone="one",size="32"} 122
angie_slabs_slots_free{zone="one",size="128"} 22
angie_slabs_slots_free{zone="one",size="512"} 4
angie_slabs_slots_free{zone="two",size="8"} 311
...
```

如果没有匹配（使用任何匹配方法），则不会添加指标。

> **备注**
>
> `path=` 参数仅在 Angie 使用*API* 模块构建时可用。

`type=type`, `help=help`	分别设置指标的类型和帮助字符串，采用 Prometheus 格式，这些将与指标一起添加到响应中，不进行更改或验证。

内置变量

`http_prometheus` 模块有一个内置变量，当使用*prometheus_template* 指令定义的指标的 *match_string* 参数匹配 Angie API 的*/status* 部分中的指标路径时，该变量会接收其值。

`$p8s_value`

如果*prometheus_template* 中定义的指标的 *match_string* 匹配任何路径，位于该路径的 Angie 指标的值将存储在 `$p8s_value` 变量中。它旨在用于基于 `path=` 参数计算的指标定义中的 *variable* 字段。

存储在 `$p8s_value` 变量中的 Angie 指标值并不总是满足 Prometheus 格式的要求。在这种情况下，可以使用*map* 指令，例如将字符串转换为数字：

```
map $p8s_value $ups_state_n {
    up           0;
    unavailable  1;
    down         2;
    default      3;
}
```

```
prometheus_template main {
    'angie_http_upstreams_state{upstream="$1",peer="$2"}' $ups_state_n
        path=~^/http/upstreams/([^/]+)/peers/([^/]+)/state$;
}
```

如果 Angie 指标具有布尔值，即 `true` 或 `false`，变量将分别接收值 `"1"` 或 `"0"`；如果指标值为 `null`，变量将为 `"(null)"`。对于日期，使用整数 UNIX 纪元格式。

Proxy

允许将请求传递给另一个（代理）服务器。

配置示例

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

指令

proxy_bind

语法	`proxy_bind` *address* [`transparent`] | `off`;
默认	—
上下文	http, server, location

使发送到代理服务器的外部连接源自指定的本地 IP 地址，并可选择端口。参数值可以包含变量。特殊值 `off` 取消从先前配置级别继承的 *proxy_bind* 指令的效果，允许系统自动分配本地 IP 地址和端口。

`transparent` 参数允许从非本地 IP 地址发起的外部连接，例如，从客户端的真实 IP 地址：

```
proxy_bind $remote_addr transparent;
```

为了使该参数生效，通常需要以超级用户 权限运行 Angie 工作进程。在 Linux 上这是不需要的，因为如果指定了 `transparent` 参数，工作进程将从主进程继承 *CAP_NET_RAW* 权限。

重要

必须配置内核路由表以拦截来自代理服务器的网络流量。

proxy_buffer_size

语法	`proxy_buffer_size` *size*;
默认	`proxy_buffer_size 4k|8k;`
上下文	http, server, location

设置用于读取从代理服务器接收的响应的第一部分的缓冲区大小。该部分通常包含一个小的响应头。默认情况下，缓冲区大小等于一个内存页。这是 4K 或 8K，具体取决于平台。然而，它可以被设置得更小。

proxy_buffering

语法	`proxy_buffering on	off;`
默认	`proxy_buffering on;`
上下文	http, server, location

启用或禁用来自代理服务器的响应的缓冲。

`on`	Angie 尽快接收来自代理服务器的响应，并将其保存到由*proxy_buffer_size*和*proxy_buffers* 指令设置的缓冲区中。如果整个响应无法适应内存，部分响应可以保存到磁盘上的临时文件 中。写入临时文件受*proxy_max_temp_file_size*和*proxy_temp_file_write_size* 指令控制。
`off`	响应同步传递给客户端，立即传递给接收的响应。Angie 不会尝试从代理服务器读取整个响应。Angie 从服务器一次可以接收的数据的最大大小由*proxy_buffer_size* 指令设置。

缓冲也可以通过在”X-Accel-Buffering” 响应头字段中传递”yes” 或”no” 来启用或禁用。此功能可以通过使用*proxy_ignore_headers* 指令来禁用。

proxy_buffers

语法	`proxy_buffers` *number size*;
默认	`proxy_buffers 8 4k|8k;`
上下文	http, server, location

设置用于从代理服务器读取响应的缓冲区数量和大小，针对单个连接。

默认情况下，缓冲区大小等于一个内存页。这是 4K 或 8K，具体取决于平台。

proxy_busy_buffers_size

语法	`proxy_busy_buffers_size` *size*;
默认	`proxy_busy_buffers_size 8k|16k;`
上下文	http, server, location

当启用来自代理服务器的响应的缓冲 时，限制在响应尚未完全读取时可以忙于发送响应到客户端的缓冲区的总大小。在此期间，其余的缓冲区可以用于读取响应，并在需要时将部分响应缓冲到临时文件中。

默认情况下，大小由*proxy_buffer_size* 和*proxy_buffers* 指令设置的两个缓冲区的大小限制。

proxy_cache

语法	`proxy_cache` *zone* | `off` [`path=`*path*];
默认	`proxy_cache off;`
上下文	http, server, location

定义用于缓存的共享内存区域。该区域可以在配置中多次使用。参数值可以包含变量。

`off`	禁用从先前配置级别继承的缓存。

Added in version 1.2.0: PRO

在 Angie PRO 中，您可以指定多个*proxy_cache_path* 指令，它们共享相同的 `keys_zone` 值以实现 '缓存分片 '。如果您这样做，请设置 `path` 参数的*proxy_cache* 指令，引用此 `keys_zone`：

`path=`*path*	该值在后端的响应被 * 缓存 * 时确定，这意味着涉及到变量，包括存储响应某些信息的变量。 如果从缓存中获得响应，*path* 不会重新评估；因此，来自缓存的响应将保留其原始 *path* 直到将其从缓存中删除。

这允许通过将*map* 指令或脚本应用于来自后端的响应来选择缓存路径。一个 `Content-Type` 示例：

```
proxy_cache_path /cache/one keys_zone=zone:10m;
proxy_cache_path /cache/two keys_zone=zone;

map $upstream_http_content_type $cache {
   ~^text/  one;
   default  two;
}

server {
   ...
   location / {
```

```
        proxy_pass http://backend;
        proxy_cache zone path=/cache/$cache;
    }
}
```

这添加了两个缓存路径和一个变量映射来选择它们。如果 Content-Type 以 text/ 开头，则使用第一个路径；否则，使用第二个。

proxy_cache_background_update

语法	proxy_cache_background_update on | off;
默认	proxy_cache_background_update off;
上下文	http, server, location

允许启动后台子请求以更新过期的缓存项，同时将过期的缓存响应返回给客户端。

> **注意**
>
> 请注意，必须允许 使用过期的缓存响应进行更新。

proxy_cache_bypass

语法	proxy_cache_bypass ...;
默认	—
上下文	http, server, location

定义在什么条件下响应不会从缓存中获取。如果字符串参数的至少一个值不为空且不等于”0”，则响应将不会从缓存中获取：

```
proxy_cache_bypass $cookie_nocache $arg_nocache$arg_comment;
proxy_cache_bypass $http_pragma    $http_authorization;
```

可以与*proxy_no_cache* 指令一起使用。

proxy_cache_convert_head

语法	proxy_cache_convert_head on | off;
默认	proxy_cache_convert_head on;
上下文	http, server, location

启用或禁用将” HEAD” 方法转换为” GET” 以进行缓存。当禁用转换时，缓存键 应配置为包含*$request_method*。

proxy_cache_key

语法	proxy_cache_key *string*;
默认	proxy_cache_key $scheme$proxy_host$request_uri;
上下文	http, server, location

定义缓存的键，例如

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

默认情况下，该指令的值接近于字符串

```
proxy_cache_key $scheme$proxy_host$uri$is_args$args;
```

proxy_cache_lock

语法	proxy_cache_lock on | off;
默认	proxy_cache_lock off;
上下文	http, server, location

启用后，每次仅允许一个请求通过传递请求到代理服务器填充根据*proxy_cache_key* 指令标识的新缓存元素。其他相同缓存元素的请求将等待响应出现在缓存中，或等待该元素的缓存锁被释放，直到*proxy_cache_lock_timeout* 指令设置的时间。

proxy_cache_lock_age

语法	proxy_cache_lock_age *time*;
默认	proxy_cache_lock_age 5s;
上下文	http, server, location

如果传递给代理服务器的最后一个请求以填充新缓存元素的请求在指定时间内未完成，则可以传递另一个请求到代理服务器。

proxy_cache_lock_timeout

语法	proxy_cache_lock_timeout *time*;
默认	proxy_cache_lock_timeout 5s;
上下文	http, server, location

设置*proxy_cache_lock* 的超时。当时间到期时，请求将传递到代理服务器，但响应不会被缓存。

proxy_cache_max_range_offset

语法	`proxy_cache_max_range_offset` *number*;
默认值	—
上下文	http, server, location

为字节范围请求设置字节偏移量。如果范围超出该偏移量，则范围请求将转发给被代理服务器，响应将不被缓存。

proxy_cache_methods

语法	`proxy_cache_methods GET | HEAD | POST ...;`
默认值	`proxy_cache_methods GET HEAD;`
上下文	http, server, location

如果客户端请求的方法在此指令中列出，则响应将被缓存。"GET" 和"HEAD" 方法始终被添加到列表中，尽管建议显式指定它们。另请参见*proxy_no_cache* 指令。

proxy_cache_min_uses

语法	`proxy_cache_min_uses` *number*;
默认值	`proxy_cache_min_uses 1;`
上下文	http, server, location

设置响应被缓存之前的请求数量。

proxy_cache_path

语法	`proxy_cache_path` *path* [`levels=`*levels*] [`use_temp_path=on | off`] `keys_zone=`*name:size*[`:file=`*file*] [`inactive=`*time*] [`max_size=`*size*] [`min_free=`*size*] [`manager_files=`*number*] [`manager_sleep=`*time*] [`manager_threshold=`*time*] [`loader_files=`*number*] [`loader_sleep=`*time*] [`loader_threshold=`*time*];
默认值	—
上下文	http

设置缓存的 ‘路径 ‘和其他参数。缓存数据存储在文件中。缓存中的文件名是对缓存键 应用 MD5 函数的结果。

`levels`	定义缓存的层级：从 1 到 3，每层接受值 1 或 2。

例如，在以下配置中：

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

缓存中的文件名将如下所示：

```
/data/angie/cache/c/29/b7f54b2df7773722d382f4809d65029c
```

缓存响应首先写入临时文件，然后重命名该文件。临时文件和缓存可以放在不同的文件系统上。然而，请注意，在这种情况下，文件是在两个文件系统之间复制，而不是便宜的重命名操作。因此，建议对于任何给定的位置，缓存和存放临时文件的目录放在同一文件系统上。

`use_temp_path=on | off`	设置临时文件的目录
`on`	如果省略此参数或设置为 `on`，将使用为给定 *location* 设置的*proxy_temp_path* 指令的目录。
`off`	临时文件将直接放在缓存目录中。
`keys_zone`	配置共享内存区域的名称和大小，以存储所有活动的键和数据的信息。 一个兆字节的区域可以存储大约 8000 个键。 当在 `keys_zone` 中使用可选参数 *file* 时，Angie 会在主进程退出时将该区域的内容刷写到磁盘，并在下次启动 或二进制升级 后，尝试在相同的内存地址恢复该区域，以实现更可靠的数据持久化并加快缓存加载时间。 如果由于区域大小变化、二进制版本不兼容或其他原因导致无法恢复，Angie 将记录一条警报 (`failed to restore zone at address`)，并且不会使用区域恢复机制。不兼容的文件将被重命名为 `.old`；您可以选择删除它，或者还原其名称并将 Angie 恢复到最初创建该文件时的配置和版本。 **注意** 请确保 *file* 路径有效，且具有 Angie 所需的正确权限，同时防止未授权访问；相对路径是基于 前缀的。
`inactive`	在此参数指定的时间内未被访问的缓存数据将从缓存中移除，无论其新鲜度如何。默认设置为 10 分钟。

备注

Added in version 1.2.0: PRO

在 Angie PRO 中，允许多个共享相同 `keys_zone` 值的 *proxy_cache_path* 指令。仅第一个此类指令可以设置共享内存区域的大小。选择此类指令是通过相关*proxy_cache* 指令的 `path` 参数来完成的。

一个特殊的 ** 缓存管理器 ** 进程监视缓存的最大大小和文件系统中缓存的最小可用空间，当超过大小或可用空间不足时，它会删除最少最近使用的数据。数据以迭代的方式被删除。

max_size	最大缓存大小
min_free	文件系统中缓存的最小可用空间
manager_files	限制每次迭代中删除的项目数量 默认值为 100
manager_threshol	限制每次迭代的持续时间 默认值为 200 毫秒
manager_sleep	配置交互之间的暂停时间 默认值为 50 毫秒

在 Angie 启动一分钟后，特殊的 ** 缓存加载器 ** 进程会被激活。它会扫描文件系统中之前缓存的数据，并将这些信息加载到缓存区域中。该过程以迭代的方式进行；每次迭代处理由 `loader_files` 设置的有限数量的项目，确保不会超过 `loader_threshold`，然后根据 `loader_sleep` 设置的短暂间隔暂停，再继续处理下一批数据。这些迭代会持续进行，直到加载器处理完磁盘上所有现有的缓存条目：

loader_files	限制每次迭代中加载的项目数量 默认值为 100
loader_threshold	限制每次迭代的持续时间 默认值为 200 毫秒
loader_sleep	配置交互之间的暂停时间 默认值为 50 毫秒

备注

为 `keys_zone` 参数设置 *file* 路径不会影响缓存加载器的行为。

proxy_cache_revalidate

语法	`proxy_cache_revalidate on | off;`
默认值	`proxy_cache_revalidate off;`
上下文	http, server, location

启用使用”If-Modified-Since” 和”If-None-Match” 头字段的条件请求重新验证过期缓存项。

proxy_cache_use_stale

语法	`proxy_cache_use_stale error | timeout | invalid_header | updating | http_500 | http_502 | http_503 | http_504 | http_403 | http_404 | http_429 | off ...;`
默认值	`proxy_cache_use_stale off;`
上下文	http, server, location

确定在与被代理服务器通信时，可以在何种情况下使用过期的缓存响应。该指令的参数与*proxy_next_upstream* 指令的参数匹配。

error	如果无法选择处理请求的被代理服务器，则允许使用过期的缓存响应。
updating	附加参数，如果当前正在更新，则允许使用过期的缓存响应。这有助于在更新缓存数据时最小化对被代理服务器的访问。

使用过期的缓存响应也可以在响应头中直接启用，在响应过期后指定的秒数内：

- "Cache-Control" 头字段的 stale-while-revalidate 扩展允许在当前正在更新时使用过期的缓存响应。
- "Cache-Control" 头字段的 stale-if-error 扩展允许在出现错误时使用过期的缓存响应。

备注

这优先级低于使用指令参数。

为了在填充新的缓存元素时最小化对被代理服务器的访问，可以使用*proxy_cache_lock* 指令。

proxy_cache_valid

语法	proxy_cache_valid [*code* ...] *time*;
默认值	—
上下文	http, server, location

为不同的响应代码设置缓存时间。例如，以下指令

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

为响应代码 200 和 302 设置 10 分钟的缓存，为响应代码 404 设置 1 分钟的缓存。

如果仅指定缓存时间

```
proxy_cache_valid 5m;
```

则仅缓存 200、301 和 302 响应。

此外，可以指定 any 参数以缓存任何响应：

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

备注

缓存参数也可以直接在响应头中设置。这优先于使用指令设置的缓存时间。

- ”X-Accel-Expires” 头字段以秒为单位设置响应的缓存时间。零值禁用响应缓存。如果值以 @ 前缀开头，则设置一个绝对时间（自 Epoch 起的秒数），在此时间之前响应可以被缓存。
- 如果头中不包含”X-Accel-Expires” 字段，缓存参数可以在”Expires” 或”Cache-Control” 头字段中设置。
- 如果头中包含”Set-Cookie” 字段，则该响应将不被缓存。
- 如果头中包含”Vary” 字段且特殊值为”*”，则该响应将不被缓存。如果头中包含”Vary” 字段且值为其他值，则该响应将根据相应的请求头字段进行缓存。

可以使用*proxy_ignore_headers* 指令禁用对一个或多个这些响应头字段的处理。

proxy_connect_timeout

语法	`proxy_connect_timeout` *time*;
默认值	`proxy_connect_timeout 60s;`
上下文	http, server, location

定义与被代理服务器建立连接的超时时间。需要注意的是，这个超时时间通常不能超过 75 秒。

proxy_connection_drop

语法	`proxy_connection_drop` *time* | `on` | `off`;
默认值	`proxy_connection_drop off;`
上下文	http, server, location

配置在被代理服务器被从组中移除或被*reresolve* 过程或*API* 命令 `DELETE` 标记为永久不可用后，终止与其所有连接。

当处理客户端或被代理服务器的下一个读或写事件时，连接将被终止。

设置 ‘time‘启用连接终止超时; 当设置为 `on` 时，连接立即被断开。

proxy_cookie_domain

语法	`proxy_cookie_domain off;` `proxy_cookie_domain` *domain replacement*;
默认	`proxy_cookie_domain off;`
上下文	http, server, location

设置一个文本，用于更改代理服务器响应的”Set-Cookie” 头字段中的域属性。假设代理服务器返回的”Set-Cookie” 头字段的属性为”domain=localhost”。该指令

```
proxy_cookie_domain localhost example.org;
```

将该属性重写为”domain=example.org”。

domain 和 *replacement* 字符串开头的点和域属性会被忽略。匹配不区分大小写。

domain 和 *replacement* 字符串可以包含变量：

```
proxy_cookie_domain www.$host $host;
```

该指令也可以使用正则表达式指定。在这种情况下，*domain* 应以“~”符号开头。正则表达式可以包含命名和位置捕获，*replacement* 可以引用它们：

```
proxy_cookie_domain ~\.(?P<sl_domain>[-0-9a-z]+\.[a-z]+)$ $sl_domain;
```

可以在同一层级指定多个 *proxy_cookie_domain* 指令：

```
proxy_cookie_domain localhost example.org;
proxy_cookie_domain ~\.([a-z]+\.[a-z]+)$ $1;
```

如果多个指令可以应用于 cookie，则选择第一个匹配的指令。

`off` 参数取消了从上一个配置级别继承的 *proxy_cookie_domain* 指令的效果。

proxy_cookie_flags

语法	`proxy_cookie_flags off` | *cookie* [*flag* ...];
默认	`proxy_cookie_flags off;`
上下文	http, server, location

为 cookie 设置一个或多个标志。cookie 可以包含文本、变量及其组合。标志可以包含文本、变量及其组合。

`secure` 、`httponly` 、`samesite=strict` 、`samesite=lax` 、`samesite=none` 参数添加相应的标志。

`nosecure` 、`nohttponly` 、`nosamesite` 参数移除相应的标志。

cookie 也可以使用正则表达式指定。在这种情况下，cookie 应以“~”符号开头。

可以在同一配置级别指定多个 `proxy_cookie_flags` 指令：

```
proxy_cookie_flags one httponly;
proxy_cookie_flags ~ nosecure samesite=strict;
```

如果多个指令可以应用于 cookie，则选择第一个匹配的指令。在示例中，*httponly* 标志被添加到 cookie *one* ，对于所有其他 cookie，添加了 *samesite=strict* 标志，并删除了 *secure* 标志。

`off` 参数取消了从上一个配置级别继承的 *proxy_cookie_flags* 指令的效果。

proxy_cookie_path

语法	`proxy_cookie_path off;` `proxy_cookie_path` *path replacement*;
默认	`proxy_cookie_path off;`
上下文	http, server, location

设置一个文本，用于更改代理服务器响应中"Set-Cookie" 头字段的路径属性。假设代理服务器返回的"Set-Cookie" 头字段的属性为"path=/two/some/uri/"。该指令

```
proxy_cookie_path /two/ /;
```

将该属性重写为"path=/some/uri/"。

path 和 *replacement* 字符串可以包含变量：

```
proxy_cookie_path $uri /some$uri;
```

该指令也可以使用正则表达式指定。在这种情况下，*path* 应以 "~" 符号开头进行区分大小写匹配，或以 "~*" 符号开头进行不区分大小写匹配。正则表达式可以包含命名和位置捕获，*replacement* 可以引用它们：

```
proxy_cookie_path ~*^/user/([^/]+) /u/$1;
```

可以在同一层级指定多个 'proxy_cookie_path'指令：

```
proxy_cookie_path /one/ /;
proxy_cookie_path / /two/;
```

如果多个指令可以应用于 cookie，则选择第一个匹配的指令。

`off` 参数取消了从上一个配置级别继承的 *proxy_cookie_path* 指令的效果。

proxy_force_ranges

语法	`proxy_force_ranges on | off;`
默认	`proxy_force_ranges off;`
上下文	http, server, location

启用对代理服务器的缓存和未缓存响应的字节范围支持，而不考虑这些响应中的"Accept-Ranges" 字段。

proxy_headers_hash_bucket_size

语法	`proxy_headers_hash_bucket_size` *size*;
默认	`proxy_headers_hash_bucket_size 64;`
上下文	http, server, location

设置用于*proxy_hide_header* 和*proxy_set_header* 指令的哈希表的桶大小。哈希表的设置细节将在单独讨论。

proxy_headers_hash_max_size

语法	`proxy_headers_hash_max_size` *size*;
默认	`proxy_headers_hash_max_size 512;`
上下文	http, server, location

设置用于*proxy_hide_header* 和*proxy_set_header* 指令的哈希表的最大大小。哈希表的设置细节将在单独讨论。

proxy_hide_header

语法	`proxy_hide_header` *field*;
默认	—
上下文	http, server, location

默认情况下，Angie 不会将:samp:*Date*、`Server`、”X-Pad” 和:samp:*X-Accel-…*‘头字段从代理服务器的响应传递给客户端。*‘proxy_hide_header*‘指令设置将不被传递的附加字段。如果需要允许传递字段，可以使用*:ref:‘proxy_pass_header* 指令。

proxy_http_version

语法	`proxy_http_version 1.0 | 1.1 | 3;`
默认	`proxy_http_version 1.0;`
上下文	http, server, location, if in location, limit_except

设置代理的 HTTP 协议版本。默认情况下，使用版本 1.0。建议使用版本 1.1 或更高版本与保持连接 配合使用。

proxy_http3_hq

语法	`proxy_http3_hq on | off;`
默认	`proxy_http3_hq off;`
上下文	http, server

切换特殊的 `hq-interop` 协商模式，该模式用于*QUIC* 的 互操作性测试，Angie 依赖于此。

注意

仅在运行明确需要此模式的专用测试时启用此模式。

proxy_http3_max_concurrent_streams

语法	`proxy_http3_max_concurrent_streams` *number*;
默认	`proxy_http3_max_concurrent_streams 128;`
上下文	http, server

初始化 HTTP/3 和 QUIC 设置，并设置在连接 中并发 HTTP/3 请求流的最大数量。需要启用保持连接。

proxy_http3_max_table_capacity

语法	`proxy_http3_max_table_capacity` *number*;
默认	`proxy_http3_max_table_capacity 4096;`
上下文	http, server, location

设置代理连接的 动态表 容量。

备注

类似的*http3_max_table_capacity* 指令用于服务器连接。为了避免错误，当启用缓存代理时，禁用动态表的使用。

proxy_http3_stream_buffer_size

语法	`proxy_http3_stream_buffer_size` *size*;
默认	`proxy_http3_stream_buffer_size 64k;`
上下文	http, server

设置与*QUIC* 流 一起使用的读写缓冲区的大小。

proxy_ignore_client_abort

语法	`proxy_ignore_client_abort on | off;`
默认	`proxy_ignore_client_abort off;`
上下文	http, server, location

确定当客户端在未等待响应的情况下关闭连接时，是否应关闭与代理服务器的连接。

proxy_ignore_headers

语法	`proxy_ignore_headers` *field* ...;
默认	—
上下文	http, server, location

禁用处理来自代理服务器的某些响应头字段。可以忽略的字段包括：”X-Accel-Redirect”、”X-Accel-Expires”、”X-Accel-Limit-Rate”、”X-Accel-Buffering”、”X-Accel-Charset”、”Expires”、”Cache-Control”、”Set-Cookie” 和”Vary”。

如果未禁用，对这些头字段的处理具有以下效果：

- ”X-Accel-Expires”、”Expires”、”Cache-Control”、”Set-Cookie” 和”Vary” 设置响应的缓存 参数；
- ”X-Accel-Redirect” 执行对指定 URI 的内部 重定向；
- ”X-Accel-Limit-Rate” 设置对客户端响应传输的速率限制；
- ”X-Accel-Buffering” 启用或禁用对响应的缓冲；
- ”X-Accel-Charset” 设置响应的期望字符集。

proxy_intercept_errors

语法	`proxy_intercept_errors on | off;`
默认值	`proxy_intercept_errors off;`
上下文	http, server, location

确定是否应将响应代码大于或等于 300 的代理响应传递给客户端，还是应拦截并重定向到 Angie 进行处理，使用*error_page* 指令。

proxy_limit_rate

语法	`proxy_limit_rate` *rate*;
默认值	`proxy_limit_rate 0;`
上下文	http, server, location

限制从代理服务器读取响应的速度。*rate* 以字节每秒为单位指定，可以包含变量。

0	禁用速率限制

备注

限制是针对每个请求设置的，因此如果 Angie 同时打开两个与代理服务器的连接，则总体速率将是指定限制的两倍。限制仅在启用缓冲 来自代理服务器的响应时有效。

proxy_max_temp_file_size

语法	`proxy_max_temp_file_size` *size*;
默认值	`proxy_max_temp_file_size 1024m;`
上下文	http, server, location

当启用来自代理服务器的响应的缓冲 时，如果整个响应不适合由*proxy_buffer_size* 和*proxy_buffers* 指令设置的缓冲区，则可以将部分响应保存到临时文件中。此指令设置临时文件的最大大小。写入临时文件的数据大小由*proxy_temp_file_write_size* 指令设置。

0	禁用响应缓冲到临时文件

备注

此限制不适用于将被缓存 或存储在磁盘上 的响应。

proxy_method

语法	`proxy_method` *method*;
默认值	—
上下文	http, server, location

指定在转发给代理服务器的请求中使用的 HTTP 方法，而不是客户端请求中的方法。参数值可以包含变量。

proxy_next_upstream

语法	proxy_next_upstream error | timeout | invalid_header | http_500 | http_502 | http_503 | http_504 | http_403 | http_404 | http_429 | non_idempotent | off ...;
默认值	proxy_next_upstream error timeout;
上下文	http, server, location

指定在什么情况下请求应传递给*upstream* 组 中的下一个服务器:

error	在与服务器建立连接、传递请求或读取响应头时发生错误;
timeout	在与服务器建立连接、传递请求或读取响应头时发生超时;
invalid_header	服务器返回空或无效响应;
http_500	服务器返回代码为 500 的响应;
http_502	服务器返回代码为 502 的响应;
http_503	服务器返回代码为 503 的响应;
http_504	服务器返回代码为 504 的响应;
http_403	服务器返回代码为 403 的响应;
http_404	服务器返回代码为 404 的响应;
http_429	服务器返回代码为 429 的响应;
non_idempotent	通常，使用 ‘非幂等 <https://datatracker.ietf.org/doc/html/rfc7231#section-4-2-2>‘_ 方法的请求（POST，LOCK，PATCH）在向上游服务器发送请求后不会传递给下一个服务器；启用此选项可明确允许重试此类请求;
off	禁止将请求传递给下一个服务器。

备注

应当记住，只有在尚未向客户端发送任何内容的情况下，才可以将请求传递给下一个服务器。也就是说，如果在传输响应的过程中发生错误或超时，则无法修复此问题。

该指令还定义了与服务器通信的不成功尝试的定义。

error timeout invalid_header	始终被视为不成功的尝试，即使它们未在指令中指定
http_500 http_502 http_503 http_504 http_429	仅在指令中指定时被视为不成功的尝试
http_403 http_404	从不被视为不成功的尝试

将请求传递给下一个服务器的尝试可以通过次数 和时间 限制。

proxy_next_upstream_timeout

语法	`proxy_next_upstream_timeout` *time*;
默认值	`proxy_next_upstream_timeout 0;`
上下文	http, server, location

限制可以传递给下一个 服务器的请求的时间。

0	关闭此限制

proxy_next_upstream_tries

语法	`proxy_next_upstream_tries` *number*;
默认值	`proxy_next_upstream_tries 0;`
上下文	http, server, location

限制将请求传递给下一个 服务器的可能尝试次数。

0	关闭此限制

proxy_no_cache

语法	`proxy_no_cache` *string* ...;
默认值	—
上下文	http, server, location

定义响应不被保存到缓存的条件。如果字符串参数的至少一个值不为空且不等于”0”，则响应将不会被保存：

```
proxy_no_cache $cookie_nocache $arg_nocache$arg_comment;
proxy_no_cache $http_pragma    $http_authorization;
```

可以与*proxy_cache_bypass* 指令一起使用。

proxy_pass

语法	proxy_pass *uri*;
默认值	—
上下文	location, if in location, limit_except

设置代理服务器的协议和地址以及可选的 URI，以便将位置映射到该 URI。可以指定的协议有 `http` 或 `https`。地址可以指定为域名或 IP 地址，并可选地指定端口：

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

或作为在单词 `unix` 后指定的 UNIX 域套接字路径，并用冒号括起来：

```
proxy_pass http://unix:/tmp/backend.socket:/uri/;
```

如果域名解析为多个地址，则将以轮询方式使用所有这些地址。此外，可以将地址指定为服务器组。

参数值可以包含变量。在这种情况下，如果地址指定为域名，则在描述的服务器组中搜索该名称，如果未找到，则使用*resolver* 确定。

请求 URI 的传递方式如下：

- 如果指定了 ** 带 URI 的 ** *proxy_pass* 指令，则在将请求传递给服务器时，将与位置匹配的规范化的 请求 URI 部分替换为指令中指定的 URI：

```
location /name/ {
    proxy_pass http://127.0.0.1/remote/;
}
```

- 如果指定了 ** 不带 URI 的 ** *proxy_pass*，则请求 URI 将以客户端发送的原始请求的形式传递给服务器，或者在处理更改的 URI 时传递完整的规范请求 URI：

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

在某些情况下，要替换的请求 URI 部分无法确定：

- 当使用正则表达式指定 *location* 时，以及在命名的 *location* 内部。

在这些情况下，应当不带 URI 指定 *proxy_pass*。

- 当在代理的 *location* 内部使用*rewrite* 指令更改 URI，并且此配置将用于处理请求（*break*）：

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

在这种情况下，指令中指定的 URI 将被忽略，完整的更改请求 URI 将传递给服务器。

- 当在 *proxy_pass* 中使用变量时：

```
location /name/ {
    proxy_pass http://127.0.0.1$request_uri;
}
```

在这种情况下，如果在指令中指定了 URI，则将按原样传递给服务器，替换原始请求 URI。

WebSocket 代理需要特殊配置。

proxy_pass_header

语法	`proxy_pass_header` *field* ...;
默认值	—
上下文	http, server, location

允许将否则禁用的 头字段从代理服务器传递给客户端。

proxy_pass_request_body

语法	`proxy_pass_request_body on | off;`
默认	`proxy_pass_request_body on;`
上下文	http, server, location

指示是否将原始请求体传递给被代理的服务器。

```
location /x-accel-redirect-here/ {
    proxy_method GET;
    proxy_pass_request_body off;
    proxy_set_header Content-Length "";

    proxy_pass ...;
}
```

另请参见*proxy_set_header* 和*proxy_pass_request_headers* 指令。

proxy_pass_request_headers

语法	`proxy_pass_request_headers on | off;`
默认	`proxy_pass_request_headers on;`
上下文	http, server, location

指示是否将原始请求的头字段传递给被代理的服务器。

```
location /x-accel-redirect-here/ {
    proxy_method GET;
    proxy_pass_request_headers off;
    proxy_pass_request_body off;

    proxy_pass ...;
}
```

另请参见*proxy_set_header* 和*proxy_pass_request_body* 指令。

proxy_pass_trailers

语法	proxy_pass_trailers on | off;
默认	proxy_pass_trailers off;
上下文	http, server, location

允许将来自被代理服务器的尾部字段传递给客户端。

在 HTTP/1.1 中，尾部部分是 显式启用的。

```
location / {
    proxy_http_version 1.1;
    proxy_set_header Connection "te";
    proxy_set_header TE "trailers";
    proxy_pass_trailers on;

    proxy_pass ...;
}
```

proxy_quic_active_connection_id_limit

语法	proxy_quic_active_connection_id_limit *number*;
默认	proxy_quic_active_connection_id_limit 2;
上下文	http, server

设置*QUIC* `active_connection_id_limit` 传输参数值。这是每个服务器可以维护的最大活跃 连接 ID 数量。

proxy_quic_gso

语法	proxy_quic_gso on | off;
默认	proxy_quic_gso off;
上下文	http, server

切换以使用*QUIC* 优化的批量模式发送数据，利用 (通用分段卸载)。

proxy_quic_host_key

语法	`proxy_quic_host_key` *file*;
默认	—
上下文	http, server

设置一个 *file*，其中包含与*QUIC* 一起使用的秘密密钥，用于加密 无状态重置 和 地址验证 令牌。默认情况下，每次重启时会生成一个随机密钥。使用旧密钥生成的令牌将不被接受。

proxy_read_timeout

语法	`proxy_read_timeout` *time*;
默认	`proxy_read_timeout 60s;`
上下文	http, server, location

定义从被代理服务器读取响应的超时时间。超时仅在两个连续读取操作之间设置，而不适用于整个响应的传输。如果被代理服务器在此时间内未传输任何内容，则连接将被关闭。

proxy_redirect

语法	`proxy_redirect default`; `proxy_redirect off`; `proxy_redirect` *redirect replacement*;
默认	`proxy_redirect default;`
上下文	http, server, location

设置在被代理服务器响应的”Location” 和”Refresh” 头字段中应更改的文本。

假设被代理服务器返回了头字段：

```
Location: http://localhost:8000/two/some/uri/
```

指令

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

将把该字符串重写为：

```
Location: http://frontend/one/some/uri/
```

在 *redirect* 字符串中可以省略服务器名称：

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

那么主服务器的名称和端口（如果不同于 80）将被插入。

由 `default` 参数指定的默认替换使用*location* 和*proxy_pass* 指令的参数。因此，下面两个配置是等效的：

```
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/;
```

小心

如果在*proxy_pass* 中使用变量，则不允许使用 `default` 参数。

replacement 字符串可以包含变量：

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

redirect 也可以包含变量：

```
proxy_redirect http://$proxy_host:8000/ /;
```

该指令可以使用正则表达式指定。在这种情况下，*redirect* 应该以"~" 符号开头以进行区分大小写的匹配，或者以"~*" 符号开头以进行不区分大小写的匹配。正则表达式可以包含命名和位置捕获，*replacement* 可以引用它们：

```
proxy_redirect ~^(http://[^:]+):\d+(/.+)$ $1$2;
proxy_redirect ~*/user/([^/]+)/(.+)$      http://$1.example.com/$2;
```

可以在同一级别指定多个 *proxy_redirect* 指令：

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

如果可以将多个指令应用于被代理服务器响应的头字段，则将选择第一个匹配的指令。

`off` 参数取消了从先前配置级别继承的 *proxy_redirect* 指令的效果。

使用该指令，还可以将主机名添加到被代理服务器发出的相对重定向中：

```
proxy_redirect / /;
```

proxy_request_buffering

语法	`proxy_request_buffering on | off;`
默认	`proxy_request_buffering on;`
上下文	http, server, location

启用或禁用客户端请求体的缓冲。

`on`	在将请求发送到被代理服务器之前，会读取 整个请求体。
`off`	请求体会在接收时立即发送到被代理服务器。在这种情况下，如果 Angie 已经开始发送请求体，则请求无法传递给下一个服务器。

当使用 HTTP/1.1 分块传输编码发送原始请求体时，无论指令值如何，请求体都将被缓冲，除非 HTTP/1.1 被启用 进行代理。

proxy_send_lowat

语法	`proxy_send_lowat` *size*;
默认	`proxy_send_lowat 0;`
上下文	http, server, location

如果指令设置为非零值，Angie 将尝试通过使用*kqueue* 方法的 *NOTE_LOWAT* 标志或指定大小的 *SO_SNDLOWAT* 套接字选项来最小化对被代理服务器的发送操作次数。

备注

此指令在 Linux、Solaris 和 Windows 上会被忽略。

proxy_send_timeout

语法	`proxy_send_timeout` *time*;
默认	`proxy_send_timeout 60s;`
上下文	http, server, location

设置向被代理服务器传输请求的超时时间。超时仅在两个连续写入操作之间设置，而不适用于整个请求的传输。如果被代理服务器在此时间内未接收到任何内容，则连接将被关闭。

proxy_set_body

语法	`proxy_set_body` *value*;
默认	—
上下文	http, server, location

允许重新定义传递给被代理服务器的请求体。值可以包含文本、变量及其组合。

proxy_set_header

语法	`proxy_set_header` *field value*;
默认	`proxy_set_header Host $proxy_host;`
上下文	http, server, location

允许重新定义或附加字段到请求头传递 给被代理服务器。值可以包含文本、变量及其组合。如果当前级别没有定义 *proxy_set_header* 指令，则这些指令将从先前的配置级别继承。默认情况下，仅重定义两个字段：

```
proxy_set_header Host       $proxy_host;
proxy_set_header Connection close;
```

如果启用了缓存，原始请求的头字段"If-Modified-Since"、"If-Unmodified-Since"、"If-None-Match"、"If-Match"、"Range" 和"If-Range" 不会传递给被代理服务器。

未更改的"Host" 请求头字段可以这样传递：

```
proxy_set_header Host       $http_host;
```

不过，如果客户端请求头中不存在此字段，则不会传递任何内容。在这种情况下，最好使用*$host* 变量 - 其值等于"Host" 请求头字段中的服务器名称，或者如果该字段不存在，则为主服务器名称：

```
proxy_set_header Host       $host;
```

此外，可以将服务器名称与被代理服务器的端口一起传递：

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

如果头字段的值为空字符串，则此字段将不会被传递到被代理服务器：

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

proxy_socket_keepalive

语法	`proxy_socket_keepalive on | off;`
默认	`proxy_socket_keepalive off;`
上下文	http, server, location

配置与被代理服务器的外发连接的"TCP keepalive" 行为。

`""`	默认情况下，操作系统的设置对套接字生效。
`on`	为套接字开启 *SO_KEEPALIVE* 套接字选项。

proxy_ssl_certificate

语法	`proxy_ssl_certificate` *file* [*file*];
默认	—
上下文	http, server, location

指定用于身份验证被代理 HTTPS 服务器的 PEM 格式证书文件。文件名中可以使用变量。

Added in version 1.2.0.

当*proxy_ssl_ntls* 启用时，该指令接受两个参数而不是一个：

```
location /proxy {
    proxy_ssl_ntls  on;

    proxy_ssl_certificate      sign.crt enc.crt;
    proxy_ssl_certificate_key  sign.key enc.key;

    proxy_ssl_ciphers "ECC-SM2-WITH-SM4-SM3:ECDHE-SM2-WITH-SM4-SM3:RSA";

    proxy_pass https://backend:443;
}
```

proxy_ssl_certificate_cache

语法	`proxy_ssl_certificate_cache off;` `proxy_ssl_certificate_cache max=`*N* [`inactive=`*time*] [`valid=`*time*];
默认值	`proxy_ssl_certificate_cache off;`
上下文	http, server, location

定义一个用于存储通过变量指定的*SSL* 证书 和私钥 的缓存。

该指令支持以下参数：

- `max` —设置缓存中元素的最大数量。缓存溢出时，最少使用（LRU）的元素将被移除。
- `inactive` —指定在多长时间未访问后移除元素。默认值为 10 秒。
- `valid` —指定缓存元素被认为有效并可重复使用的时间。默认值为 60 秒。超过此时间后，证书将重新加载或重新验证。
- `off` —禁用缓存。

示例：

```
proxy_ssl_certificate       $proxy_ssl_server_name.crt;
proxy_ssl_certificate_key   $proxy_ssl_server_name.key;
proxy_ssl_certificate_cache max=1000 inactive=20s valid=1m;
```

proxy_ssl_certificate_key

语法	`proxy_ssl_certificate_key` *file* [*file*];
默认	—
上下文	http, server, location

指定用于身份验证被代理 HTTPS 服务器的 PEM 格式私钥文件。

可以指定值"engine:*name*:id" 代替文件，从给定名称的 OpenSSL 引擎加载具有指定 id 的密钥。

文件名中可以使用变量。

Added in version 1.2.0.

当*proxy_ssl_ntls* 启用时，该指令接受两个参数而不是一个：

```
location /proxy {
    proxy_ssl_ntls  on;

    proxy_ssl_certificate      sign.crt enc.crt;
    proxy_ssl_certificate_key  sign.key enc.key;

    proxy_ssl_ciphers "ECC-SM2-WITH-SM4-SM3:ECDHE-SM2-WITH-SM4-SM3:RSA";

    proxy_pass https://backend:443;
}
```

proxy_ssl_ciphers

语法	`proxy_ssl_ciphers` *ciphers*;
默认	`proxy_ssl_ciphers DEFAULT;`
上下文	http, server, location

指定对被代理 HTTPS 服务器请求的启用密码。密码以 OpenSSL 库可以理解的格式指定。

密码列表取决于已安装的 OpenSSL 版本。可以使用 `openssl ciphers` 命令查看完整列表。

注意

当使用 OpenSSL 时，`proxy_ssl_ciphers` 指令 * 不会 * 配置 TLS 1.3 的加密套件。如需配置 TLS 1.3 的加密套件，请使用*proxy_ssl_conf_command* 指令，它是为支持高级 SSL 配置而添加的。

- 在 LibreSSL 中，可以使用 `proxy_ssl_ciphers` 配置 TLS 1.3 加密套件。
- 在 BoringSSL 中，无法配置 TLS 1.3 加密套件。

proxy_ssl_conf_command

语法	`proxy_ssl_conf_command` *name value*;
默认	—
上下文	http, server, location

在与被代理 HTTPS 服务器建立连接时设置任意 OpenSSL 配置 commands。

重要

当使用 OpenSSL 1.0.2 或更高版本时支持该指令。要在 OpenSSL 中配置 TLS 1.3 密码套件，请使用 `ciphersuites` 命令。

可以在同一层上指定多个 *proxy_ssl_conf_command* 指令。如果当前层没有定义 *proxy_ssl_conf_command* 指令，则这些指令将从上一个配置层继承。

小心

请注意，直接配置 OpenSSL 可能会导致意外行为。

proxy_ssl_crl

语法	`proxy_ssl_crl` *file*;
默认	—
上下文	http, server, location

指定包含被吊销证书 (CRL) 的 PEM 格式文件，用于验证 被代理 HTTPS 服务器的证书。

proxy_ssl_name

语法	proxy_ssl_name *name*;
默认	proxy_ssl_name $proxy_host;
上下文	http, server, location

允许覆盖用于验证 被代理 HTTPS 服务器证书的服务器名称，并在与被代理 HTTPS 服务器建立连接时通过*SNI* 传递。

默认情况下，将使用*proxy_pass* URL 的主机部分。

proxy_ssl_ntls

Added in version 1.2.0.

语法	proxy_ssl_ntls on | off;
默认	proxy_ssl_ntls off;
上下文	http, server

使用 TongSuo 库启用对 NTLS 的客户端支持。

```
location /proxy {
    proxy_ssl_ntls  on;

    proxy_ssl_certificate      sign.crt enc.crt;
    proxy_ssl_certificate_key  sign.key enc.key;

    proxy_ssl_ciphers "ECC-SM2-WITH-SM4-SM3:ECDHE-SM2-WITH-SM4-SM3:RSA";

    proxy_pass https://backend:443;
}
```

重要

使用 *--with-ntls* 构建选项构建 Angie，并链接与 NTLS 启用的 SSL 库

```
./configure --with-openssl=../Tongsuo-8.3.0 \
            --with-openssl-opt=enable-ntls  \
            --with-ntls
```

proxy_ssl_password_file

语法	`proxy_ssl_password_file` *file*;
默认	—
上下文	http, server, location

指定一个文件，其中包含密钥 的密码，每个密码单独占一行。在加载密钥时，会依次尝试这些密码。

proxy_ssl_protocols

语法	`proxy_ssl_protocols` [SSLv2] [SSLv3] [TLSv1] [TLSv1.1] [TLSv1.2] [TLSv1.3];
默认值	`proxy_ssl_protocols TLSv1.2 TLSv1.3;`
上下文	http, server, location

在 1.2.0 版本发生变更: `TLSv1.3` 参数已添加到默认集。

启用对被代理 HTTPS 服务器请求的指定协议。

proxy_ssl_server_name

语法	`proxy_ssl_server_name on | off;`
默认	`proxy_ssl_server_name off;`
上下文	http, server, location

启用或禁用通过 服务器名称指示 TLS 扩展（SNI，RFC 6066）传递*proxy_ssl_name* 指令设置的服务器名称，在与被代理 HTTPS 服务器建立连接时。

proxy_ssl_session_reuse

语法	`proxy_ssl_session_reuse on | off;`
默认	`proxy_ssl_session_reuse on;`
上下文	http, server, location

确定在与被代理服务器工作时是否可以重用 SSL 会话。如果日志中出现错误"*SSL3_GET_FINISHED:digest check failed*"，请尝试禁用会话重用。

proxy_ssl_trusted_certificate

语法	`proxy_ssl_trusted_certificate` *file*;
默认	—
上下文	http, server, location

指定一个文件，其中包含用于验证 被代理 HTTPS 服务器证书的受信任 CA 证书，格式为 PEM。

proxy_ssl_verify

语法	`proxy_ssl_verify on	off;`
默认	`proxy_ssl_verify off;`
上下文	http, server, location

启用或禁用对被代理 HTTPS 服务器证书的验证。

proxy_ssl_verify_depth

语法	`proxy_ssl_verify_depth` *number*;
默认	`proxy_ssl_verify_depth 1;`
上下文	http, server, location

设置被代理 HTTPS 服务器证书链中的验证深度。

proxy_store

语法	`proxy_store on	off |` *string*;
默认	`proxy_store off;`
上下文	http, server, location

启用将文件保存到磁盘。

`on`	使用与指令*alias* 或*root* 对应的路径保存文件
`off`	禁用文件保存

文件名可以通过带有变量的 `string` 显式设置：

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

文件的修改时间根据收到的”Last-Modified” 响应头字段进行设置。响应首先被写入临时文件，然后该文件被重命名。临时文件和持久存储可以放在不同的文件系统上。然而，请注意，在这种情况下，文件是在两个文件系统之间复制，而不是简单的重命名操作。因此，建议对于任何给定的 *location*，保存的文件和由*proxy_temp_path* 指令设置的临时文件目录放在同一文件系统上。

此指令可用于创建静态不可更改文件的本地副本，例如：

```
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/;
}
```

或者像这样：

```
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;
}
```

proxy_store_access

语法	proxy_store_access 用户: 权限...;
默认	proxy_store_access user:rw;
上下文	http, server, location

设置新创建文件和目录的访问权限，例如：

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

如果指定了任何 *group* 或 *all* 访问权限，则可以省略用户权限：

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

proxy_temp_file_write_size

语法	`proxy_temp_file_write_size` *size*;
默认	`proxy_temp_file_write_size 8k|16k;`
上下文	http, server, location

限制在启用从代理服务器到临时文件的响应缓冲时，每次写入临时文件的数据大小。默认情况下，大小由*proxy_buffer_size* 和*proxy_buffers* 指令设置的两个缓冲区限制。临时文件的最大大小由*proxy_max_temp_file_size* 指令设置。

proxy_temp_path

语法	`proxy_temp_path` *path* [*level1* [*level2* [*level3*]]]‘;
默认	`proxy_temp_path proxy_temp;`（路径取决于 `--http-proxy-temp-path` 构建选项）
上下文	http, server, location

定义一个目录，用于存储从代理服务器接收的数据的临时文件。可以在指定目录下使用最多三层的子目录层次。例如，在以下配置中

```
proxy_temp_path /spool/angie/proxy_temp 1 2;
```

临时文件可能如下所示：

```
/spool/angie/proxy_temp/7/45/00000123457
```

另请参见*proxy_cache_path* 指令的 use_temp_path 参数。

内置变量

http_proxy 模块支持可用于使用*proxy_set_header* 指令构造头的内置变量：

$proxy_host

代理服务器的名称和端口，如*proxy_pass* 指令中指定的;

$proxy_port

代理服务器的端口，如*proxy_pass* 指令中指定的，或协议的默认端口;

$proxy_add_x_forwarded_for

带有*$remote_addr* 变量的”X-Forwarded-For” 客户端请求头字段，二者之间用逗号分隔。如果客户端请求头中没有”X-Forwarded-For” 字段，则*$proxy_add_x_forwarded_for* 变量等于*$remote_addr* 变量。

Random Index

该模块处理以斜杠字符 (/) 结尾的请求，并在目录中随机选择一个文件作为索引文件提供服务。该模块在 `http_index` 模块之前处理。

当 从源代码构建时，该模块默认不会被构建；需要通过 `--with-http_random_index_module` 构建选项来启用。

在从 我们的仓库获取的软件包和镜像中，该模块已包含在构建中。

配置示例

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

指令

random_index

语法	`random_index on | off;`
默认值	`random_index off;`
上下文	location

启用或禁用在周围位置中模块的处理。

RealIP

该模块用于将客户端地址和可选端口更改为指定头字段中发送的地址和端口。

当 从源代码构建时，该模块默认未构建；需要通过 `--with-http_realip_module` 构建选项进行启用。

在来自 我们的仓库的包和镜像中，该模块已包含在构建中。

配置示例

```
set_real_ip_from  192.168.1.0/24;
set_real_ip_from  192.168.2.1;
set_real_ip_from  2001:0db8::/32;
real_ip_header    X-Forwarded-For;
real_ip_recursive on;
```

指令

set_real_ip_from

语法	`set_real_ip_from` *address* | *CIDR* | `unix:`;
默认	—
上下文	http, server, location

定义已知发送正确替换地址的可信地址。如果指定了特殊值 `unix:`，则所有 UNIX 域套接字将被信任。可信地址也可以使用主机名指定。

real_ip_header

语法	`real_ip_header` *field* | `X-Real-IP` | `X-Forwarded-For` | `proxy_protocol`;
默认	`real_ip_header X-Real-IP;`
上下文	http, server, location

定义请求头字段，其值将用于替换客户端地址。

包含可选端口的请求头字段值也用于替换客户端端口。地址和端口应根据 RFC 3986 进行指定。

`proxy_protocol` 参数将客户端地址更改为来自 PROXY 协议头的地址。必须通过在*listen* 指令中设置 *proxy_protocol* 参数来预先启用 PROXY 协议。

real_ip_recursive

语法	`real_ip_recursive on` | `off`;
默认	`real_ip_recursive off;`
上下文	http, server, location

如果禁用递归搜索，则与可信地址之一匹配的原始客户端地址被请求头字段中由*real_ip_header* 指令定义的最后一个地址替换。如果启用递归搜索，则与可信地址之一匹配的原始客户端地址被请求头字段中最后一个非可信地址替换。

内置变量

`$realip_remote_addr`

保存原始客户端地址

`$realip_remote_port`

保存原始客户端端口

Referer

该模块用于阻止具有无效"Referer" 头字段值的请求访问站点。应注意，伪造具有适当"Referer" 字段值的请求非常容易，因此该模块的目的不是彻底阻止此类请求，而是阻止由常规浏览器发送的大量请求流。还应考虑，即使是有效请求，常规浏览器也可能不发送"Referer" 字段。

配置示例

```
valid_referers none blocked server_names
               *.example.com example.* www.example.org/galleries/
               ~\.google\.;

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

指令

referer_hash_bucket_size

语法	`referer_hash_bucket_size` *size*;
默认值	`referer_hash_bucket_size 64;`
上下文	server, location

设置有效引用者哈希表的桶大小。有关设置哈希表的详细信息，请参阅单独的文档。

referer_hash_max_size

语法	`referer_hash_max_size` *size*;
默认值	`referer_hash_max_size 2048;`
上下文	server, location

设置有效引用者哈希表的最大大小。有关设置哈希表的详细信息，请参阅单独的文档。

valid_referers

语法	`valid_referers none` | `blocked` | `server_names` | *string* ...;
默认值	—
上下文	server, location

指定将导致内置*$invalid_referer* 变量被设置为空字符串的"Referer" 请求头字段值。否则，变量将被设置为"1"。匹配搜索不区分大小写。

参数可以如下：

none	请求头中缺少"Referer" 字段；
blocked	请求头中存在"Referer" 字段，但其值已被防火墙或代理服务器删除；此类值是指不以 http:// 或 https:// 开头的字符串；
server_names	"Referer" 请求头字段包含一个服务器名称；
arbitrary string	定义一个服务器名称和一个可选的 URI 前缀。服务器名称可以在开头或结尾加上"*"。在检查过程中，会忽略"Referer" 字段中的服务器端口；
regular expression	第一个符号应为"~"。应注意，表达式将与 http:// 或 https:// 之后的文本进行匹配。

示例：

```
valid_referers none blocked server_names
               *.example.com example.* www.example.org/galleries/
               ~\.google\.;
```

内置变量

$invalid_referer

如果"Referer" 请求头字段值被认为是有效 的，则为空字符串，否则为"1"。

Rewrite

该模块用于使用 PCRE 正则表达式更改请求 URI，返回重定向，并有条件地选择配置。

break、*if*、*return*、*rewrite* 和*set* 指令按以下顺序处理：

- 在 *server* 级别上指定的此模块的指令按顺序执行；
- 重复执行：
  - 基于请求 URI 搜索 *location*；
  - 在找到的位置内按顺序执行此模块指定的指令；
  - 如果请求 URI 被重写，则重复此循环，但不超过 10 次。

指令

break

语法	break;
默认值	—
上下文	server, location, if

停止处理当前的 *http_rewrite* 指令集。

如果在 *location* 内指定了指令，则请求的进一步处理将在该 *location* 中继续。

示例：

```
if ($slow) {
    limit_rate 10k;
    break;
}
```

if

语法	`if` (*condition*) { ... }
默认值	—
上下文	server, location

指定的条件被评估。如果为真，则执行大括号内指定的模块指令，并将请求分配给 *if* 指令内的配置。*if* 指令内的配置从先前的配置级别继承。

条件可以是以下任意一种：

- 变量名；如果变量的值为空字符串或”0”，则为假；
- 使用”=” 和”!=” 运算符将变量与字符串进行比较；
- 使用”~”（区分大小写匹配）和”~*”（不区分大小写匹配）运算符将变量与正则表达式进行匹配。正则表达式可以包含捕获，在 $1..$9 变量中可以重复使用。还提供了负运算符”!~” 和”!~*”。如果正则表达式包含”}” 或”;” 字符，则整个表达式应放在单引号或双引号中。
- 使用”-f” 和”!-f” 运算符检查文件是否存在；
- 使用”-d” 和”!-d” 运算符检查目录是否存在；
- 使用”-e” 和”!-e” 运算符检查文件、目录或符号链接是否存在；
- 使用”-x” 和”!-x” 运算符检查可执行文件。

示例：

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

if ($http_cookie ~* "id=([^;]+)(?:;|$)") {
    set $id $1;
}

if ($request_method = POST) {
    return 405;
}

if ($slow) {
    limit_rate 10k;
```

```
}

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

备注

$invalid_referer 内置变量的值由*valid_referers* 指令设置。

return

语法	`return` *code* [*text*]; `return` *code URL*; `return` *URL*;
默认值	—
上下文	server, location, if

停止处理并将指定的 `code` 返回给客户端。非标准代码 444 在不发送响应头的情况下关闭连接。

可以指定重定向 URL（对于代码 301、302、303、307 和 308）或响应体文本（对于其他代码）。响应体文本和重定向 URL 可以包含变量。作为特例，可以将重定向 URL 指定为本服务器的本地 URI，在这种情况下，完整的重定向 URL 根据请求方案（*$scheme*）以及*server_name_in_redirect* 和*port_in_redirect* 指令形成。

此外，可以将代码 302 的临时重定向 URL 作为唯一参数指定。该参数应以 `http://`、`https://` 或"*$scheme*" 字符串开头。URL 可以包含变量。

另请参阅*error_page* 指令。

rewrite

语法	`rewrite` *regex replacement* [*flag*];
默认值	—
上下文	server, location, if

如果指定的正则表达式匹配请求 URI，则 URI 将按 `replacement` 字符串中指定的方式更改。*rewrite* 指令按其在配置文件中的出现顺序顺序执行。可以使用 `flags` 终止指令的进一步处理。如果 `replacement` 字符串以 `http://`、`https://` 或"*$scheme*" 开头，处理将停止并返回重定向给客户端。

可选的 `flag` 参数可以是以下之一：

last	停止处理当前的 *http_rewrite* 指令集，并开始搜索与更改后的 URI 匹配的新 *location*；
break	停止处理当前的 *http_rewrite* 指令集，如同 *break* 指令；
redirect	返回临时重定向，状态码为 302；如果 replacement 字符串不以 http://、https:// 或”*$scheme*” 开头，则使用该参数；
permanent	返回永久重定向，状态码为 301。

完整的重定向 URL 根据请求方案（*$scheme*）以及*server_name_in_redirect* 和*port_in_redirect* 指令形成。

示例：

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

但是，如果这些指令放在”/download/” 位置内，last 标志应替换为 break，否则 Angie 将循环 10 次并返回 500 错误：

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

如果 replacement 字符串包含新的请求参数，则先前的请求参数将附加在后面。如果这不是所希望的，可以在替换字符串末尾放置问号，以避免附加它们，例如：

```
rewrite ^/users/(.*)$ /show?user=$1? last;
```

如果正则表达式包含”}” 或”;” 字符，则整个表达式应放在单引号或双引号中。

rewrite_log

语法	rewrite_log on | off;
默认值	rewrite_log off;
上下文	http, server, location, if

启用或禁用将 *http_rewrite* 模块指令处理结果记录到*error_log* 的 *notice* 级别。

set

语法	`set` *$variable value*;
默认值	—
上下文	server, location, if

为指定的变量设置值。该值可以包含文本、变量及其组合。

uninitialized_variable_warn

语法	`uninitialized_variable_warn on | off;`
默认值	`uninitialized_variable_warn on;`
上下文	http, server, location, if

控制是否记录关于未初始化变量的警告。

内部实现

http_rewrite 模块指令在配置阶段编译为内部指令，这些指令在请求处理期间被解释。解释器是一个简单的虚拟栈机器。

例如，指令

```
location /download/ {
    if ($forbidden) {
        return 403;
    }

    if ($slow) {
        limit_rate 10k;
    }

    rewrite ^/(download/.*)/media/(.*)\..*$ /$1/mp3/$2.mp3 break;
}
```

将被转换为以下指令：

```
variable $forbidden
check against zero
    return 403
    end of code
variable $slow
check against zero
match of regular expression
copy "/"
copy $1
```

```
copy "/mp3/"
copy $2
copy ".mp3"
end of regular expression
end of code
```

请注意，上述*limit_rate* 指令没有指令，因为它与 *http_rewrite* 模块无关。为 *if* 块创建了一个单独的配置。如果条件为真，则请求将分配给此配置，其中 *limit_rate* 等于 10k。

指令

```
rewrite ^/(download/.*)/media/(.*)\..*$ /$1/mp3/$2.mp3 break;
```

如果将正则表达式中的第一个斜杠放在括号内，则可以减少一个指令：

```
rewrite ^(/download/.*)/media/(.*)\..*$ $1/mp3/$2.mp3 break;
```

相应的指令将如下所示：

```
match of regular expression
copy $1
copy "/mp3/"
copy $2
copy ".mp3"
end of regular expression
end of code
```

SCGI

允许将请求传递给 SCGI 服务器。

配置示例

```
location / {
    include   scgi_params;
    scgi_pass localhost:9000;
}
```

指令

scgi_bind

语法	scgi_bind *address* [transparent] | off;
默认	—
上下文	http, server, location

使发往 SCGI 服务器的出站连接从指定的本地 IP 地址（可选端口）发起。参数值可以包含变量。特殊值 `off` 取消从前一个配置级别继承的 *scgi_bind* 指令的效果，这允许系统自动分配本地 IP 地址和端口。

`transparent` 参数允许发往 SCGI 服务器的出站连接从非本地 IP 地址发起，例如，从客户端的真实 IP 地址：

```
scgi_bind $remote_addr transparent;
```

为了使此参数有效，通常需要以超级用户 权限运行 Angie 工作进程。在 Linux 上则不需要，因为如果指定了 `transparent` 参数，工作进程会从主进程继承 *CAP_NET_RAW* 能力。

重要

需要配置内核路由表以拦截来自 SCGI 服务器的网络流量。

scgi_buffer_size

语法	`scgi_buffer_size` *size*;
默认	`scgi_buffer_size 4k|8k;`
上下文	http, server, location

设置用于读取从 SCGI 服务器接收到的响应第一部分的缓冲区大小。这部分通常包含一个小的响应头。默认情况下，缓冲区大小等于一个内存页面。这是 4K 或 8K，具体取决于平台。虽然可以将其设置得更小。

scgi_buffering

语法	`scgi_buffering` *size*;
默认	`scgi_buffering on;`
上下文	http, server, location

启用或禁用从 SCGI 服务器返回响应的缓冲。

`on`	Angie 尽快从 SCGI 服务器接收响应，并将其保存到由*scgi_buffer_size* 和*scgi_buffers* 指令设置的缓冲区中。如果整个响应无法全部装入内存，则部分响应可以保存到磁盘上的*temporary file*。写入临时文件受*scgi_max_temp_file_size* 和*scgi_temp_file_write_size* 指令的控制。
`off`	响应以同步方式传递给客户端，立即在接收到时传递。Angie 不会尝试从 SCGI 服务器读取整个响应。Angie 一次从服务器可以接收的数据的最大大小由*scgi_buffer_size* 指令设置。

缓冲也可以通过在"X-Accel-Buffering" 响应头字段中传递"yes" 或"no" 来启用或禁用。此功能可以使用*scgi_ignore_headers* 指令禁用。

scgi_buffers

语法	`scgi_buffers` *number size*;
默认	`scgi_buffers 8 4k	8k;`
上下文	http, server, location

设置用于从 SCGI 服务器读取响应的缓冲区的数量和大小，针对单个连接。

默认情况下，缓冲区大小等于一个内存页面。这是 4K 或 8K，具体取决于平台。

scgi_busy_buffers_size

语法	`scgi_busy_buffers_size` *size*;
默认	`scgi_busy_buffers_size 8k	16k;`
上下文	http, server, location

当*buffering* 从 SCGI 服务器的响应被启用时，限制可以忙于将响应发送给客户端的缓冲区的总大小，同时响应尚未完全读取。与此同时，其余的缓冲区可以用于读取响应，并在必要时将部分响应缓冲到临时文件。

默认情况下，大小受*scgi_buffer_size* 和*scgi_buffers* 指令设置的两个缓冲区的大小限制。

scgi_cache

语法	`scgi_cache` *zone*	`off`;
默认	`scgi_cache off;`
上下文	http, server, location

定义用于缓存的共享内存区域。相同的区域可以在多个地方使用。参数值可以包含变量。

`off`	禁用从前一个配置级别继承的缓存。

scgi_cache_background_update

语法	`scgi_cache_background_update on`	`off`;
默认	`scgi_cache_background_update off;`
上下文	http, server, location

允许在返回过期缓存项的同时，启动后台子请求以更新该缓存项。

> **⚠ 注意**
>
> 请注意，必须允许 在更新时使用过期的缓存响应。

scgi_cache_bypass

语法	scgi_cache_bypass ...;
默认	—
上下文	http, server, location

定义不从缓存中获取响应的条件。如果字符串参数的至少一个值不为空且不等于"0"，则响应不会从缓存中获取：

```
scgi_cache_bypass $cookie_nocache $arg_nocache$arg_comment;
scgi_cache_bypass $http_pragma    $http_authorization;
```

可以与*scgi_no_cache* 指令一起使用。

scgi_cache_key

语法	scgi_cache_key *string*;
默认	—
上下文	http, server, location

定义缓存的键，例如：

```
scgi_cache_key localhost:9000$request_uri;
```

scgi_cache_lock

语法	scgi_cache_lock on | off;
默认	scgi_cache_lock off;
上下文	http, server, location

启用时，将只允许一个请求一次填充由*scgi_cache_key* 指令标识的新缓存元素，方法是将请求传递给 SCGI 服务器。同一缓存元素的其他请求将等待响应出现在缓存中或等待此元素的缓存锁释放，直到*scgi_cache_lock_timeout* 指令设置的时间为止。

scgi_cache_lock_age

语法	`scgi_cache_lock_age` *time*;
默认	`scgi_cache_lock_age 5s;`
上下文	http, server, location

如果传递给 SCGI 服务器用于填充新缓存元素的最后一个请求在指定时间内未完成，则可以再向 SCGI 服务器发出一个请求。

scgi_cache_lock_timeout

语法	`scgi_cache_lock_timeout` *time*;
默认	`scgi_cache_lock_timeout 5s;`
上下文	http, server, location

为*scgi_cache_lock* 设置超时。当时间到达时，请求将被传递给 SCGI 服务器，但响应将不会被缓存。

scgi_cache_max_range_offset

语法	`scgi_cache_max_range_offset` *number*;
默认	—
上下文	http, server, location

为字节范围请求设置字节偏移量。如果范围超出偏移量，则范围请求将被传递给 SCGI 服务器，响应将不会被缓存。

scgi_cache_methods

语法	`scgi_cache_methods GET | HEAD | POST ...;`
默认	`scgi_cache_methods GET HEAD;`
上下文	http, server, location

如果客户端请求方法在此指令中列出，则响应将被缓存。“GET”和“HEAD”方法始终被添加到列表中，尽管建议明确指定它们。另见*scgi_no_cache* 指令。

scgi_cache_min_uses

语法	`scgi_cache_min_uses` *number*;
默认	`scgi_cache_min_uses 1;`
上下文	http, server, location

设置在响应被缓存之前的请求数量。

scgi_cache_path

语法	scgi_cache_path *path* [levels=*levels*] [use_temp_path=on | off] keys_zone=*name:size* [inactive=*time*] [max_size=*size*] [min_free=*size*] [manager_files=*number*] [manager_sleep=*time*] [manager_threshold=*time*] [loader_files=*number*] [loader_sleep=*time*] [loader_threshold=*time*];
默认	—
上下文	http

设置缓存的路径和其他参数。缓存数据存储在文件中。缓存中的文件名是对*cache key* 应用 MD5 函数的结果。

`levels` 参数定义了缓存的层级，从 1 到 3，每个层级接受值 1 或 2。例如，在以下配置中：

```
scgi_cache_path /data/angie/cache levels=1:2 keys_zone=one:10m;
```

缓存中的文件名将如下所示：

```
/data/angie/cache/c/29/b7f54b2df7773722d382f4809d65029c
```

缓存响应首先写入一个临时文件，然后该文件被重命名。临时文件和缓存可以放在不同的文件系统上。然而，请注意，在这种情况下，文件是跨两个文件系统复制而不是便宜的重命名操作。因此，建议对于任何给定的位置，缓存和存放临时文件的目录应放在同一个文件系统上。

临时文件的目录是基于 `use_temp_path` 参数设置的。

`on`	如果此参数被省略或设置为 on，则将使用为给定 *location* 设置的*scgi_temp_path* 指令所指定的目录。
`off`	临时文件将直接放入缓存目录中。

此外，所有活动的键和数据的信息存储在一个共享内存区域，其名称和大小由 `keys_zone` 参数配置。一个兆字节的区域可以存储大约 8000 个键。

在 `inactive` 参数指定的时间内未访问的缓存数据将从缓存中移除，而不管其新鲜度。

默认情况下，*inactive* 设置为 10 分钟。

一个特殊的 **缓存管理器**进程监视最大缓存大小和缓存文件系统上最小的可用空间，当大小超出或可用空间不足时，它会移除最少最近使用的数据。数据是以迭代的方式移除的。

max_size	最大缓存大小
min_free	缓存文件系统上最小的可用空间
manager_files	限制在一个迭代中要删除的项目数量 默认值：100
manager_threshol	限制一个迭代的持续时间 默认值：200 毫秒
manager_sleep	配置迭代之间的暂停 默认值：50 毫秒

在 Angie 启动后的一分钟，特殊的 **缓存加载器**进程被激活。它将存储在文件系统中的先前缓存的数据的信息加载到缓存区域。加载也是以迭代方式进行的。

loader_files	限制在一个迭代中要加载的项目数量 默认值：100
loader_threshold	限制一个迭代的持续时间 默认值：200 毫秒
loader_sleep	配置迭代之间的暂停 默认值：50 毫秒

scgi_cache_revalidate

语法	scgi_cache_revalidate on | off;
默认	scgi_cache_revalidate off;
上下文	http, server, location

启用使用"If-Modified-Since" 和"If-None-Match" 头字段的条件请求来重新验证过期的缓存项。

scgi_cache_use_stale

语法	scgi_cache_use_stale error | timeout | invalid_header | updating | http_500 | http_502 | http_503 | http_504 | http_403 | http_404 | http_429 | off ...;
默认	scgi_cache_use_stale off;
上下文	http, server, location

确定在与 SCGI 服务器通信时可以在何种情况下使用过期的缓存响应。指令的参数与*scgi_next_upstream* 指令的参数相匹配。

error	如果无法选择处理请求的 SCGI 服务器，则允许使用过期的缓存响应。
updating	额外参数，允许使用过期的缓存响应如果它正在被更新。这允许在更新缓存数据时最小化对 SCGI 服务器的访问。

使用过期的缓存响应也可以在响应头中直接启用，指定响应变得过期后允许的秒数：

- "Cache-Control" 头字段的 stale-while-revalidate 扩展允许在其当前被更新时使用过期的缓存响应。
- "Cache-Control" 头字段的 stale-if-error 扩展允许在发生错误时使用过期的缓存响应。

备注

这优先级低于使用指令参数。

为了最小化在填充新缓存元素时对 SCGI 服务器的访问，可以使用*scgi_cache_lock* 指令。

scgi_cache_valid

语法	`scgi_cache_valid` [*code* ...] *time*;
默认	—
上下文	http, server, location

为不同的响应代码设置缓存时间。例如，以下指令

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

为响应代码 200 和 302 设置 10 分钟的缓存，为响应代码 404 设置 1 分钟的缓存。

如果只指定缓存时间

```
scgi_cache_valid 5m;
```

则仅缓存 200、301 和 302 响应。

此外，可以指定 `any` 参数以缓存任何响应：

```
scgi_cache_valid 200 302 10m;
scgi_cache_valid 301      1h;
scgi_cache_valid any      1m;
```

备注

缓存的参数也可以直接在响应头中设置。这优先级高于使用指令设置的缓存时间。

- "X-Accel-Expires" 头字段以秒为单位设置响应的缓存时间。零值禁用响应的缓存。如果值以 @ 前缀开头，则设置响应可能被缓存的绝对时间（自 Epoch 起的秒数）。
- 如果头中不包含"X-Accel-Expires" 字段，则可以在"Expires" 或"Cache-Control" 的头字段中设置缓存参数。

- 如果头中包含"Set-Cookie" 字段，则该响应不会被缓存。
- 如果头中包含特殊值为"*" 的"Vary" 字段，则该响应不会被缓存。如果头中包含其他值的"Vary" 字段，则该响应将基于相应的请求头字段进行缓存。

可以使用*scgi_ignore_headers* 指令禁用对这些响应头字段之一或多个的处理。

scgi_connect_timeout

语法	`scgi_connect_timeout` *time*;
默认	`scgi_connect_timeout 60s;`
上下文	http, server, location

定义与 SCGI 服务器建立连接的超时时间。需要注意的是，此超时时间通常不能超过 75 秒。

scgi_connection_drop

语法	`scgi_connection_drop` *time* | `on` | `off`;
默认	`scgi_connection_drop off;`
上下文	http, server, location

在代理服务器被移除出组或被*reresolve* 进程或*API* 命令 `DELETE` 标记为永久不可用后，启用终止与代理服务器的所有连接。

在处理客户端或代理服务器的下一个读或写事件时，连接将被终止。

设置 *time* 启用连接终止超时；设置为 `on` 时，连接将立即被断开。

scgi_force_ranges

语法	`scgi_force_ranges off`;
默认	`scgi_force_ranges off;`
上下文	http, server, location

启用对 SCGI 服务器的缓存和未缓存响应的字节范围支持，而不管这些响应中的"Accept-Ranges" 字段。

scgi_hide_header

语法	`scgi_hide_header` *field*;
默认	—
上下文	http, server, location

默认情况下，Angie 不会将 SCGI 服务器响应中的 `Status` 和 `X-Accel-...` 头字段传递给客户端。*:samp:'scgi_hide_header* 指令设置将不会被传递的其他字段。如果需要允许传递字段，则可以使

用*scgi_pass_header* 指令。

scgi_ignore_client_abort

语法	`scgi_ignore_client_abort on	off;`
默认	`scgi_ignore_client_abort off;`
上下文	http, server, location

确定当客户端在等待响应时关闭连接时，是否应关闭与 SCGI 服务器的连接。

scgi_ignore_headers

语法	`scgi_ignore_headers` *field* ...;
默认	—
上下文	http, server, location

禁用处理来自 SCGI 服务器的某些响应头字段。可以忽略的字段包括：”X-Accel-Redirect”、”X-Accel-Expires”、”X-Accel-Limit-Rate”、”X-Accel-Buffering”、”X-Accel-Charset”、”Expires”、”Cache-Control”、”Set-Cookie” 和”Vary”。

如果没有被禁用，这些头字段的处理将产生以下效果：

- ”X-Accel-Expires”、”Expires”、”Cache-Control”、”Set-Cookie” 和”Vary” 设置响应缓存 的参数；
- ”X-Accel-Redirect” 执行对指定 URI 的内部 重定向；
- ”X-Accel-Limit-Rate” 设置响应传输到客户端的速率限制；
- ”X-Accel-Buffering” 启用或禁用响应的缓冲；
- ”X-Accel-Charset” 设置响应的期望字符集。
  - * 语法
    * `scgi_intercept_errors on | off;`
  - * 默认值
    * `scgi_intercept_errors off;`
  - * 上下文
    * http, server, location

确定是否应该将状态码大于或等于 300 的 SCGI 响应传递给客户端，或者拦截并重定向到 Angie 进行处理，使用*error_page* 指令。

scgi_limit_rate

语法	`scgi_limit_rate` *rate*;
默认值	`scgi_limit_rate 0;`
上下文	http, server, location

限制从 SCGI 服务器读取响应的速度。*rate* 以每秒字节为单位，并且可以包含变量。

0	禁用速率限制

备注

限制是按请求设置的，因此如果 Angie 同时打开两个与 SCGI 服务器的连接，则整体速率将是指定限制的两倍。该限制仅在启用 SCGI 服务器响应的*buffering* 时有效。

scgi_max_temp_file_size

语法	`scgi_max_temp_file_size` *size*;
默认值	`scgi_max_temp_file_size 1024m;`
上下文	http, server, location

当启用 SCGI 服务器响应的*buffering* 时，如果整个响应无法适合由*scgi_buffer_size* 和*scgi_buffers* 指令设置的缓冲区，则响应的一部分可以保存到临时文件中。此指令设置临时文件的最大大小。一次写入临时文件的数据大小由*scgi_temp_file_write_size* 指令设置。

0	禁用响应缓冲到临时文件

备注

此限制不适用于将被缓存或存储在磁盘 的响应。

scgi_next_upstream

语法	`scgi_next_upstream error | timeout | invalid_header | http_500 | http_503 | http_403 | http_404 | http_429 | non_idempotent | off` ...;
默认值	`scgi_next_upstream error timeout;`
上下文	http, server, location

指定在何种情况下请求应传递给*upstream pool* 中的下一个服务器：

error	在与服务器建立连接、传递请求或读取响应头时发生错误；
timeout	在与服务器建立连接、传递请求或读取响应头时发生超时；
invalid_header	服务器返回了空的或无效的响应；
http_500	服务器返回了状态码 500 的响应；
http_503	服务器返回了状态码 503 的响应；
http_403	服务器返回了状态码 403 的响应；
http_404	服务器返回了状态码 404 的响应；
http_429	服务器返回了状态码 429 的响应；
non_idempotent	通常，带有 非幂等 方法的请求（POST、LOCK、PATCH）在向上游服务器发送请求后不会传递给下一个服务器；启用此选项明确允许重试此类请求；
off	禁用将请求传递给下一个服务器。

备注

应注意，只有在尚未向客户端发送任何内容时，才能将请求传递给下一个服务器。也就是说，如果在响应传输过程中发生错误或超时，则无法进行修复。

该指令还定义了什么被视为与服务器通信的:ref: 不成功尝试 *<max_fails>*。

error timeout invalid_header	始终被视为不成功的尝试，即使在指令中未指定
http_500 http_503 http_429	仅在指令中指定时被视为不成功的尝试
http_403 http_404	永远不被视为不成功的尝试

将请求传递给下一个服务器可能会受到尝试次数 和时间 的限制。

scgi_next_upstream_timeout

语法	scgi_next_upstream_timeout *time*;
默认值	scgi_next_upstream_timeout 0;
上下文	http, server, location

限制可以传递给下一个 服务器的请求的时间。

0	关闭此限制

scgi_next_upstream_tries

语法	`scgi_next_upstream_tries` *number*;
默认值	`scgi_next_upstream_tries 0;`
上下文	http, server, location

限制将请求传递给下一个 服务器的可能尝试次数。

`0`	关闭此限制

scgi_no_cache

语法	`scgi_no_cache` *string* ...;
默认值	—
上下文	http, server, location

定义在何种情况下响应将不会被保存到缓存。如果字符串参数的至少一个值不为空并且不等于"0"，则响应将不会被保存：

```
scgi_no_cache $cookie_nocache $arg_nocache$arg_comment;
scgi_no_cache $http_pragma    $http_authorization;
```

可以与*scgi_cache_bypass* 指令一起使用。

scgi_param

语法	`scgi_param` *parameter value* [`if_not_empty`];
默认值	—
上下文	http, server, location

设置应传递给 SCGI 服务器的参数。值可以包含文本、变量及其组合。这些指令仅在当前级别没有定义 scgi_param 指令的情况下从上一个配置级别继承。

标准 CGI 环境变量 应作为 SCGI 头提供，请参见分发中提供的 scgi_params 文件：

```
location / {
    include scgi_params;
#    ...
}
```

如果指令与 `if_not_empty` 一起指定，则仅当该参数的值不为空时，才会将该参数传递给服务器：

```
scgi_param HTTPS $https if_not_empty;
```

scgi_pass

语法	scgi_pass *uri*;
默认值	—
上下文	location, if in location, limit_except

设置 SCGI 服务器的地址。地址可以指定为域名或 IP 地址，以及一个可选的端口：

```
scgi_pass localhost:9000;
```

或指定为在单词 `unix` 之后并用冒号括起来的 UNIX 域套接字路径：

```
scgi_pass unix:/tmp/scgi.socket;
```

如果一个域名解析到多个地址，则将以轮询的方式使用它们。此外，地址可以指定为服务器组。如果使用了组，则不能与其一起指定端口；相反，单独为组内的每个服务器指定端口。

参数值可以包含变量。在这种情况下，如果将地址指定为域名，则将在描述的服务器组中搜索该名称，如果未找到，则使用*resolver* 确定。

scgi_pass_header

语法	scgi_pass_header *field* ...;
默认值	—
上下文	http, server, location

允许将来自 SCGI 服务器的否则被禁用 的头字段传递给客户端。

scgi_pass_request_body

语法	scgi_pass_request_body on | off;
默认值	scgi_pass_request_body on;
上下文	http, server, location

指示是否将原始请求体传递给 SCGI 服务器。另请参见*scgi_pass_request_headers* 指令。

scgi_pass_request_headers

语法	scgi_pass_request_headers on | off;
默认值	scgi_pass_request_headers on;
上下文	http, server, location

指示是否将原始请求的头字段传递给 SCGI 服务器。另请参见*scgi_pass_request_body* 指令。

scgi_read_timeout

语法	`scgi_read_timeout` *time*;
默认值	`scgi_read_timeout 60s;`
上下文	http, server, location

定义从 SCGI 服务器读取响应的超时时间。超时仅在两个连续读取操作之间设定，而不是针对整个响应的传输。如果 SCGI 服务器在此时间内没有传输任何内容，则连接将关闭。

scgi_request_buffering

语法	`scgi_request_buffering on | off;`
默认值	`scgi_request_buffering on;`
上下文	http, server, location

启用或禁用客户端请求体的缓冲。

`on`	在将请求发送到 SCGI 服务器之前，从客户端读取 整个请求体。
`off`	请求体在接收时立即发送到 SCGI 服务器。在这种情况下，如果 Angie 已经开始发送请求体，则请求无法传递给下一个服务器。

当使用 HTTP/1.1 分块传输编码发送原始请求体时，无论指令值如何，请求体都会被缓冲。

scgi_send_timeout

语法	`scgi_send_timeout` 时间;
默认	`scgi_send_timeout 60s;`
上下文	http, server, location

设置向 SCGI 服务器传输请求的超时。超时仅在两个连续的写操作之间设置，而不是针对整个请求的传输。如果 SCGI 服务器在此时间内未接收到任何内容，则连接将被关闭。

scgi_socket_keepalive

语法	`scgi_socket_keepalive on | off;`
默认	`scgi_socket_keepalive off;`
上下文	http, server, location

配置与 SCGI 服务器的出站连接的"TCP keepalive" 行为。

""	默认情况下，操作系统的设置对套接字生效。
on	为套接字开启 *SO_KEEPALIVE* 套接字选项。

scgi_store

语法	scgi_store on | off | 字符串;
默认	scgi_store off;
上下文	http, server, location

启用将文件保存到磁盘。

on	保存与指令*alias* 或*root* 相对应的路径的文件。
off	禁用保存文件。

文件名可以通过使用带有变量的 *字符串*显式设置：

```
scgi_store /data/www$original_uri;
```

文件的修改时间根据收到的"Last-Modified" 响应头字段设置。响应首先写入临时文件，然后重命名该文件。临时文件和持久存储可以放在不同的文件系统上。然而，请注意，在这种情况下，文件是在两个文件系统之间复制，而不是便宜的重命名操作。因此，建议对于任何给定的 *location*，保存的文件和由*scgi_temp_path* 指令设置的临时文件目录放在同一文件系统上。

此指令可用于创建静态不可更改文件的本地副本，例如：

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

location /fetch/ {
    internal;

    scgi_pass          backend:9000;
    ...

    scgi_store         on;
    scgi_store_access  user:rw group:rw all:r;
    scgi_temp_path     /data/temp;

    alias              /data/www/;
}
```

scgi_store_access

语法	scgi_store_access 用户: 权限...;
默认	scgi_store_access user:rw;
上下文	http, server, location

设置新创建的文件和目录的访问权限，例如：

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

如果指定了任何 group 或 all 访问权限，则用户权限可以省略：

```
scgi_store_access group:rw all:r;
```

scgi_temp_file_write_size

语法	scgi_temp_file_write_size 大小;
默认	scgi_temp_file_write_size 8k|16k;
上下文	http, server, location

限制在启用从 SCGI 服务器到临时文件的响应缓冲时，一次写入临时文件的数据大小。默认情况下，大小由*scgi_buffer_size* 和*scgi_buffers* 指令设置的两个缓冲区限制。临时文件的最大大小由*scgi_max_temp_file_size* 指令设置。

scgi_temp_path

语法	scgi_temp_path 路径 [*level1* [*level2* [*level3*]]]‘;
默认	scgi_temp_path scgi_temp; (路径取决于 --http-scgi-temp-path 构建选项)
上下文	http, server, location

定义一个目录，用于存储从 SCGI 服务器接收到的临时文件。可以在指定目录下使用最多三级子目录层次。例如，在以下配置中

```
scgi_temp_path /spool/angie/scgi_temp 1 2;
```

临时文件可能看起来像这样：

```
/spool/angie/scgi_temp/7/45/00000123457
```

另请参见*scgi_cache_path* 指令的 use_temp_path 参数。

Secure Link

该模块用于检查请求链接的真实性，保护资源免受未经授权的访问，并限制链接的生命周期。

请求链接的真实性通过比较请求中传递的校验和值和为请求计算的值来验证。如果链接有有限的生命周期且时间已过期，则该链接被视为过时。这些检查的状态在*$secure_link* 变量中可用。

该模块提供两种替代操作模式。第一种模式通过*secure_link_secret* 指令启用，用于检查请求链接的真实性以及保护资源免受未经授权的访问。第二种模式通过*secure_link* 和*secure_link_md5* 指令启用，也用于限制链接的生命周期。

当 从源代码构建时，此模块默认不构建；应使用 `--with-http_secure_link_module` 构建选项启用。

在来自 我们的仓库的软件包和镜像中，模块包含在构建中。

指令

secure_link

语法	`secure_link` 表达式;
默认值	—
上下文	http, server, location

定义一个包含变量的字符串，用于从中提取链接的校验和值和生命周期。

表达式中使用的变量通常与请求相关联；请参阅下面的示例。

从字符串中提取的校验和值与由*secure_link_md5* 指令定义的表达式的 MD5 哈希值进行比较。

如果校验和不匹配，则*$secure_link* 变量被设置为空字符串。如果校验和匹配，则检查链接的生命周期。

如果链接有有限的生命周期且时间已过期，则*$secure_link* 变量被设置为 `0`。否则，它被设置为 `1`。请求中传递的 MD5 哈希值以 base64url 编码。

如果链接有有限的生命周期，则过期时间以自纪元（1970 年 1 月 1 日 00:00:00 GMT）以来的秒数设置。该值在表达式中指定于 MD5 哈希之后，并以逗号分隔。请求中传递的过期时间可通过*$secure_link_expires* 变量用于*secure_link_md5* 指令。如果未指定过期时间，则链接具有无限的生命周期。

secure_link_md5

语法	`secure_link_md5` 表达式;
默认值	—
上下文	http, server, location

定义一个表达式，其 MD5 哈希值将被计算并与请求中传递的值进行比较。

表达式应包含链接（资源）的受保护部分和一个秘密成分。如果链接有有限的生命周期，表达式还应包含*$secure_link_expires*。

为了防止未经授权的访问，表达式可以包含一些关于客户端的信息，例如其地址和浏览器版本。

示例：

```
location /s/ {
    secure_link $arg_md5,$arg_expires;
    secure_link_md5 "$secure_link_expires$uri$remote_addr secret";

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

    if ($secure_link = "0") {
        return 410;
    }

#    ...
}
```

链接"/s/link?md5=_e4Nc3iduzkWRm01TBBNYw&expires=2147483647" 限制了对客户端 IP 地址为 127.0.0.1 的"/s/link" 的访问。该链接也有限的生命周期，直到 2038 年 1 月 19 日（GMT）。

在 UNIX 上，可以通过以下命令获取 md5 请求参数值：

```
echo -n '2147483647/s/link127.0.0.1 secret' | \
   openssl md5 -binary | openssl base64 | tr +/ -_ | tr -d =
```

secure_link_secret

语法	secure_link_secret 单词;
默认值	—
上下文	location

定义一个秘密单词，用于检查请求链接的真实性。

请求链接的完整 URI 如下所示：

```
/prefix/hash/link
```

其中 hash 是为链接和秘密单词连接而计算的 MD5 哈希的十六进制表示，prefix 是没有斜杠的任意字符串。

如果请求链接通过了真实性检查，则*$secure_link* 变量被设置为从请求 URI 提取的链接。否则，*$secure_link* 变量被设置为空字符串。

示例：

```
location /p/ {
    secure_link_secret secret;

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

    rewrite ^ /secure/$secure_link;
}

location /secure/ {
    internal;
}
```

请求"/p/5e814704a28d9bc1914ff19fa0c4a00a/link" 将被内部重定向到"/secure/link"。

在 UNIX 上，可以通过以下命令获取此示例的哈希值：

```
echo -n 'linksecret' | openssl md5 -hex
```

内置变量

`$secure_link`

链接检查的状态。具体值取决于所选的操作模式。

`$secure_link_expires`

在请求中传递的链接的生命周期；仅用于*secure_link_md5* 指令。

Slice

该模块是一个过滤器，将请求拆分为子请求，每个子请求返回特定范围的响应。该过滤器提供了更有效的大响应缓存。

当 从源代码构建时，默认情况下不会构建此模块；它需要通过 `--with-http_slice_module` 构建选项启用。

在来自 我们仓库的包和镜像中，该模块已包含在构建中。

配置示例

```
location / {
    slice             1m;
    proxy_cache       cache;
    proxy_cache_key   $uri$is_args$args$slice_range;
    proxy_set_header  Range $slice_range;
    proxy_cache_valid 200 206 1h;
```

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

在此示例中，响应被分割为 1 兆字节可缓存的切片。

指令

slice

语法	slice *size*;
默认值	slice 0;
上下文	http, server, location

设置切片的大小。零值将禁用将响应分割为切片。

> **警告**
>
> 请注意，值过低可能导致过多的内存使用和打开大量文件。

为了使子请求返回所需的范围，应该将*$slice_range* 变量作为"Range" 请求头字段传递给代理服务器。如果启用了缓存，应将*$slice_range* 添加到缓存键 中，并应启用 对状态码为 206 的响应的缓存。

内置变量

$slice_range

以 HTTP 字节范围 格式表示的当前切片范围，例如，`bytes=0-1048575`。

Split Clients

该模块用于 A/B 测试、金丝雀发布或其他需要将一部分客户端路由到一个服务器或配置，同时将其余部分路由到其他地方的场景。

配置示例

```
http {
    split_clients "${remote_addr}AAA" $variant {
                  0.5%               .one;
                  2.0%               .two;
                  *                  "";
    }

    server {
        location / {
            index index${variant}.html;
```

指令

split_clients

语法	split_clients *string* *$variable* { ... }
默认值	—
上下文	http

通过哈希 *string* 创建一个 *$variable*；*string* 中的变量被替换，替换结果被哈希，然后将哈希值用于选择 *$variable* 的字符串值。

哈希函数使用 MurmurHash2 （32 位），其整个值范围（0 到 4294967295）按出现顺序映射到桶中；百分比决定了桶的大小。通配符 (*) 可以出现在最后；不属于其他桶的哈希值会映射到其指定的值。

示例：

```
split_clients "${remote_addr}AAA" $variant {
               0.5%               .one;
               2.0%               .two;
               *                  "";
}
```

在这里，替换 $*remote_addr*AAA 字符串后，哈希值分布如下：

- 值从 0 到 21474835（0.5%）产生 .one；
- 值从 21474836 到 107374180（2%）产生 .two；
- 值从 107374181 到 4294967295（所有其他值）产生 "" （空字符串）。

SSI

该模块是一个过滤器，用于处理通过它传递的响应中的 SSI（服务器端包含）命令。

配置示例

```
location / {
    ssi on;
#    ...
}
```

指令

ssi

语法	ssi on | off;
默认值	ssi off;
上下文	http, server, location, if in location

启用或禁用响应中 SSI 命令的处理。

ssi_last_modified

语法	`ssi_last_modified on | off;`
默认值	`ssi_last_modified off;`
上下文	http, server, location

允许在 SSI 处理过程中保留原始响应中的"Last-Modified" 头字段，以便于响应缓存。

默认情况下，头字段会被移除，因为响应的内容在处理过程中被修改，可能包含动态生成的元素或与原始响应独立更改的部分。

ssi_min_file_chunk

语法	`ssi_min_file_chunk` *size*;
默认值	`ssi_min_file_chunk 1k;`
上下文	http, server, location

设置存储在磁盘上的响应部分的最小大小，从该大小开始，使用*sendfile* 发送它们是有意义的。

ssi_silent_errors

语法	`ssi_silent_errors on | off;`
默认值	`ssi_silent_errors off;`
上下文	http, server, location

如果启用，在 SSI 处理期间发生错误时，将抑制输出"*[an error occurred while processing the directive]*" 字符串。

ssi_types

语法	`ssi_types` mime-*type* ...;
默认值	`ssi_types text/html;`
上下文	http, server, location

启用对具有指定 MIME 类型的响应中 SSI 命令的处理，除了 `text/html`。特殊值"*" 匹配任何 MIME 类型。

ssi_value_length

语法	`ssi_value_length` *length*;
默认值	`ssi_value_length 256;`
上下文	http, server, location

设置 SSI 命令中参数值的最大长度。

SSI 命令

SSI 命令具有以下通用格式：

```
<!--# command parameter1=value1 parameter2=value2 ... -->
```

支持以下命令：

block

定义一个可以作为 *include* 命令中的存根使用的块。块可以包含其他 SSI 命令。该命令具有以下参数：

name

块名称。

示例：

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

config

设置在 SSI 处理期间使用的一些参数，即：

errmsg

在 SSI 处理期间发生错误时输出的字符串。默认情况下，输出以下字符串：

```
`[an error occurred while processing the directive]`
```

timefmt

传递给 `strftime()` 函数的格式字符串，用于输出日期和时间。默认情况下，使用以下格式：

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

"%s" 格式适合以秒为单位输出时间。

`echo`

输出变量的值。该命令具有以下参数：

`var`

变量名称。

`encoding`

编码方法。可能的值包括 `none`、`url` 和 `entity`。默认使用 `entity`。

`default`

一个非标准参数，设置在变量未定义时输出的字符串。默认输出 (none)。

该命令

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

替换以下命令序列：

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

`if`

执行条件包含。支持以下命令：

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

当前仅支持一层嵌套。该命令具有以下参数：

`expr`

表达式。表达式可以是：

- 变量存在性检查：

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

- 变量与文本的比较：

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

- 变量与正则表达式的比较：

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

如果 *text* 包含变量，则会替换它们的值。正则表达式可以包含位置捕获和命名捕获，稍后可以通过变量使用，例如：

```
<!--# if expr="$name = /(.+)@(?P<domain>.+)/" -->
  <!--# echo var="1" -->
  <!--# echo var="domain" -->
<!--# endif -->
```

include

将另一个请求的结果包含到响应中。该命令具有以下参数：

file

指定包含的文件，例如：

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

virtual

指定包含的请求，例如：

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

在一页上指定的多个请求，由代理或 FastCGI/uwsgi/SCGI/gRPC 服务器处理的请求会并行运行。如果需要顺序处理，则应使用 *wait* 参数。

stub

一个非标准参数，命名包含请求结果为空主体或请求处理期间发生错误时输出的块，例如：

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

替换块内容在包含请求上下文中处理。

`wait`

一个非标准参数，指示在继续 SSI 处理之前等待请求完全完成，例如：

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

`set`

一个非标准参数，指示将请求处理的成功结果写入指定变量，例如：

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

响应的最大大小由*subrequest_output_buffer_size* 指令设置：

```
location /remote/ {
    subrequest_output_buffer_size 64k;
#    ...
}
```

`set`

设置变量的值。该命令具有以下参数：

`var`

变量名称。

`value`

变量值。如果赋值中包含变量，则会替换它们的值。

内置变量

`$date_local`

当前时间（本地时区）。格式由 *config* 命令中的 *timefmt* 参数设置。

`$date_gmt`

当前时间（GMT）。格式由 *config* 命令中的 *timefmt* 参数设置。

SSL

提供 HTTPS 所需的支持。

当 从源代码构建时，此模块默认并未构建；应通过 `--with-http_ssl_module` 构建选项启用。

在 我们的仓库中的包和映像中，该模块包含在构建中。

重要

此模块需要 OpenSSL 库。

配置示例

为了降低处理器负载，建议

- 将工作进程数量 设置为处理器的数量，
- 启用保持连接，
- 启用共享 会话缓存，
- 禁用内置 会话缓存，
- 并可能增加会话生命周期 （默认 5 分钟）：

```
worker_processes auto;

http {

    # ...

    server {
        listen              443 ssl;
        keepalive_timeout   70;

        ssl_protocols       TLSv1.2 TLSv1.3;
        ssl_ciphers         AES128-SHA:AES256-SHA:RC4-SHA:DES-CBC3-SHA:RC4-MD5;
        ssl_certificate     /usr/local/angie/conf/cert.pem;
        ssl_certificate_key /usr/local/angie/conf/cert.key;
        ssl_session_cache   shared:SSL:10m;
        ssl_session_timeout 10m;

        # ...
    }
```

指令

ssl_buffer_size

语法	`ssl_buffer_size` *size*;
默认	`ssl_buffer_size 16k;`
上下文	http, server

设置发送数据所用的缓冲区大小。

默认情况下，缓冲区大小为 16k，这对应于发送大响应时的最小开销。为了最小化首次字节时间，使用更小的值可能是有益的，例如：

```
ssl_buffer_size 4k;
```

ssl_certificate

语法	ssl_certificate *file* [*file*];
默认	—
上下文	http, server

指定给定虚拟服务器的 PEM 格式证书文件。如果需要指定中间证书，除了主证书外，应在同一文件中按照以下顺序指定：主证书在前，中间证书在后。可以在同一文件中放置 PEM 格式的私钥。

此指令可以多次指定，以加载不同类型的证书，例如 RSA 和 ECDSA：

```
server {
    listen              443 ssl;
    server_name         example.com;

    ssl_certificate     example.com.rsa.crt;
    ssl_certificate_key example.com.rsa.key;

    ssl_certificate     example.com.ecdsa.crt;
    ssl_certificate_key example.com.ecdsa.key;

    # ...
}
```

只有 OpenSSL 1.0.2 或更高版本支持不同证书的独立证书链。在较旧版本中，只能使用一个证书链。

重要

在使用 OpenSSL 1.0.2 或更高版本时，可以在文件名中使用变量：

```
ssl_certificate     $ssl_server_name.crt;
ssl_certificate_key $ssl_server_name.key;
```

请注意，使用变量意味着每次 SSL 握手时都将加载证书，这可能会对性能产生负面影响。

可以指定值"data:*$variable*" 来代替 `file`，这会从变量中加载证书，而无需使用中间文件。请注意，不当使用此语法可能会带来安全隐患，例如将私钥数据写入错误日志。

重要

需要记住的是，由于 HTTPS 协议的限制，出于最大互操作性的考虑，虚拟服务器应监听不同的 *IP* 地址。

Added in version 1.2.0: 如果启用了 *ssl_ntls*，则该指令可以接受两个参数（密钥的签名部分和加密部分）而不是一个：

```
listen ... ssl;

ssl_ntls  on;

# 双 NTLS 证书
ssl_certificate      sign.crt enc.crt;
ssl_certificate_key  sign.key enc.key;

# 可以与常规 RSA 证书一起使用
ssl_certificate  rsa.crt;
ssl_certificate  rsa.key;
```

ssl_certificate_cache

语法	`ssl_certificate_cache off;` `ssl_certificate_cache max=`*N* [`inactive=`*time*] [`valid=`*time*];
默认值	`ssl_certificate_cache off;`
上下文	http, server

定义一个用于存储通过变量指定的 *SSL* 证书 和私钥 的缓存。

该指令支持以下参数：

- `max` —设置缓存中元素的最大数量。缓存溢出时，最少使用（LRU）的元素将被移除。
- `inactive` —指定在多长时间未访问后移除元素。默认值为 10 秒。
- `valid` —指定缓存元素被认为有效并可重复使用的时间。默认值为 60 秒。超过此时间后，证书将重新加载或重新验证。
- `off` —禁用缓存。

示例：

```
ssl_certificate       $ssl_server_name.crt;
ssl_certificate_key   $ssl_server_name.key;
ssl_certificate_cache max=1000 inactive=20s valid=1m;
```

ssl_certificate_key

语法	`ssl_certificate_key` *file* [*file*];
默认	—
上下文	http, server

指定给定虚拟服务器的 PEM 格式私钥文件。

重要

在使用 OpenSSL 1.0.2 或更高版本时，可以在文件名中使用变量。

可以指定值 `engine:`name`:id` 来代替 `file`，这会从指定的 OpenSSL 引擎名称加载带有指定 *id* 的私钥。

可以指定值"data:*$variable*" 来代替 `file`，这会从变量中加载私钥，而无需使用中间文件。请注意，不当使用此语法可能会带来安全隐患，例如将私钥数据写入错误日志。

Added in version 1.2.0: 如果启用了*ssl_ntls*，则该指令可以接受两个参数（密钥的签名部分和加密部分）而不是一个：

```
listen ... ssl;

ssl_ntls  on;

# 双 NTLS 证书
ssl_certificate      sign.crt enc.crt;
ssl_certificate_key  sign.key enc.key;

# 可以与常规 RSA 证书一起使用
ssl_certificate  rsa.crt;
ssl_certificate  rsa.key;
```

ssl_ciphers

语法	`ssl_ciphers` *ciphers*;
默认	`ssl_ciphers HIGH:!aNULL:!MD5;`
上下文	http, server

指定启用的密码套件。密码套件以 OpenSSL 库理解的格式指定，例如：

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

密码列表取决于已安装的 OpenSSL 版本。可以使用 `openssl ciphers` 命令查看完整列表。

注意

当使用 OpenSSL 时，`ssl_ciphers` 指令 * 不会 * 配置 TLS 1.3 的加密套件。如需配置 TLS 1.3 的加密套件，请使用*ssl_conf_command* 指令，它是为支持高级 SSL 配置而添加的。

- 在 LibreSSL 中，可以使用 `ssl_ciphers` 配置 TLS 1.3 加密套件。
- 在 BoringSSL 中，无法配置 TLS 1.3 加密套件。

ssl_client_certificate

语法	`ssl_client_certificate` *file*;
默认	—
上下文	http, server

指定用于验证 客户端证书和 OCSP 响应的 PEM 格式受信任 CA 证书文件，如果启用了*ssl_stapling*。

证书列表将发送给客户端。如果不希望这样，可以使用*ssl_trusted_certificate* 指令。

ssl_conf_command

语法	`ssl_conf_command` *name value*;
默认	—
上下文	http, server

设置任意 OpenSSL 配置命令。

重要

在使用 OpenSSL 1.0.2 或更高版本时支持该指令。要在 OpenSSL 中配置 TLS 1.3 密码套件，请使用 `ciphersuites` 命令。

可以在同一级别上指定多个 *ssl_conf_command* 指令：

```
ssl_conf_command Options PrioritizeChaCha;
ssl_conf_command Ciphersuites TLS_CHACHA20_POLY1305_SHA256;
```

如果当前级别没有定义 *ssl_conf_command* 指令，则这些指令会从上一个配置级别继承。

小心

直接配置 OpenSSL 可能导致意外行为。

ssl_crl

语法	`ssl_crl` *file*;
默认	—
上下文	http, server

指定用于验证 客户端证书的 PEM 格式吊销证书文件（CRL）。

ssl_dhparam

语法	`ssl_dhparam` *file*;
默认	—
上下文	http, server

指定用于 DHE 密码套件的 DH 参数文件。

默认情况下未设置参数，因此不会使用 DHE 密码套件。

ssl_early_data

语法	`ssl_early_data on | off;`
默认	`ssl_early_data off;`
上下文	http, server

启用或禁用 TLS 1.3 早期数据。

在早期数据中发送的请求会受到 重放攻击 的影响。为了保护应用层免受此类攻击，应使用*$ssl_early_data* 变量。

```
proxy_set_header Early-Data $ssl_early_data;
```

重要

在使用 OpenSSL 1.1.1 或更高版本以及 BoringSSL 时支持该指令。

ssl_ecdh_curve

语法	`ssl_ecdh_curve` *curve*;
默认	`ssl_ecdh_curve auto;`
上下文	http, server

指定 ECDHE 密码套件的曲线。

重要

在使用 OpenSSL 1.0.2 或更高版本时，可以指定多个曲线，例如：

```
ssl_ecdh_curve prime256v1:secp384r1;
```

特殊值 `auto` 指示 Angie 在使用 OpenSSL 1.0.2 或更高版本时使用 OpenSSL 库内置的列表，或者在较旧版本中使用 *prime256v1*。

重要

在使用 OpenSSL 1.0.2 或更高版本时，该指令设置服务器支持的曲线列表。因此，为了使 ECDSA 证书能够正常工作，重要的是要包括证书中使用的曲线。

ssl_ntls

Added in version 1.2.0.

语法	`ssl_ntls on | off;`
默认	`ssl_ntls off;`
上下文	http, server

启用服务器端对 NTLS 的支持，使用 TongSuo 库。

```
listen ... ssl;
ssl_ntls  on;
```

重要

使用 *--with-ntls* 构建选项构建 Angie，并链接到启用 NTLS 的 SSL 库。

```
./configure --with-openssl=../Tongsuo-8.3.0 \
            --with-openssl-opt=enable-ntls  \
            --with-ntls
```

ssl_ocsp

语法	`ssl_ocsp on | off | leaf;`
默认	`ssl_ocsp off;`
上下文	http, server

启用客户端证书链的 OCSP 验证。*:samp:‘leaf* 参数仅启用客户端证书的验证。

为了使 OCSP 验证正常工作，*ssl_verify_client* 指令应设置为 *:samp:‘on* 或 *:samp:‘optional*。

要解析 OCSP 响应者主机名，*resolver* 指令也应被指定。

示例：

```
ssl_verify_client on;
ssl_ocsp          on;
resolver          127.0.0.53;
```

ssl_ocsp_cache

语法	`ssl_ocsp_cache off` | [shared:*name*:*size*];
默认	`ssl_ocsp_cache off;`
上下文	http, server

设置存储客户端证书状态以进行 OCSP 验证的缓存的名称和大小。该缓存在所有工作进程之间共享。可以在多个虚拟服务器中使用相同名称的缓存。

`off` 参数禁止使用缓存。

ssl_ocsp_responder

语法	`ssl_ocsp_responder` *uri*;
默认	—
上下文	http, server

重写在 ”Authority Information Access” 证书扩展中指定的 OCSP 响应者的 URI，以进行验证 客户端证书。

仅支持 `http://` OCSP 响应者：

```
ssl_ocsp_responder http://ocsp.example.com/;
```

ssl_password_file

语法	`ssl_password_file` *file*;
默认	—
上下文	http, server

指定一个包含秘密密钥 的文件，每个密码短语在单独的一行中指定。加载密钥时将依次尝试密码短语。

示例：

```
http {
    ssl_password_file /etc/keys/global.pass;
    ...

    server {
        server_name www1.example.com;
        ssl_certificate_key /etc/keys/first.key;
    }

    server {
        server_name www2.example.com;

        # 也可以使用命名管道代替文件
        ssl_password_file /etc/keys/fifo;
        ssl_certificate_key /etc/keys/second.key;
    }
}
```

ssl_prefer_server_ciphers

语法	ssl_prefer_server_ciphers on | off;
默认	ssl_prefer_server_ciphers off;
上下文	http, server

指定在使用 SSLv3 和 TLS 协议时，服务器密码应优于客户端密码。

ssl_protocols

语法	ssl_protocols [SSLv2] [SSLv3] [TLSv1] [TLSv1.1] [TLSv1.2] [TLSv1.3];
默认值	ssl_protocols TLSv1.2 TLSv1.3;
上下文	http, server

在 1.2.0 版本发生变更: TLSv1.3 参数添加到默认集合中。

启用指定的协议。

重要

TLSv1.1 和 TLSv1.2 参数仅在使用 OpenSSL 1.0.1 或更高版本时有效。

TLSv1.3 参数仅在使用 OpenSSL 1.1.1 或更高版本时有效。

ssl_reject_handshake

语法	ssl_reject_handshake on | off;
默认	ssl_reject_handshake off;
上下文	http, server

如果启用，:*ref:'server* 块中的 SSL 握手将被拒绝。

例如，在以下配置中，拒绝与服务器名称不是 *example.com* 的 SSL 握手：

```
server {
    listen               443 ssl default_server;
    ssl_reject_handshake on;
}

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

ssl_session_cache

语法	ssl_session_cache off | none | [builtin[:*size*]] [shared:*name*:*size*];
默认	ssl_session_cache none;
上下文	http, server

设置存储会话参数的缓存的类型和大小。缓存可以是以下任何类型：

off	严格禁止使用会话缓存：Angie 明确告诉客户端会话不能重用。
none	温和不允许使用会话缓存：Angie 告诉客户端会话可以重用，但实际上不在缓存中存储会话参数。
builtin	OpenSSL 中内置的缓存；仅由一个工作进程使用。缓存大小以会话为单位指定。如果未给定大小，则等于 20480 会话。使用内置缓存可能导致内存碎片。
shared	在所有工作进程之间共享的缓存。缓存大小以字节为单位指定；一兆字节可以存储大约 4000 个会话。每个共享缓存应有一个任意名称。可以在多个虚拟服务器中使用相同名称的缓存。它还用于自动生成、存储和定期轮换 TLS 会话票据密钥，除非使用*ssl_session_ticket_key* 指令显式配置。

可以同时使用两种缓存类型，例如：

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

但仅使用共享缓存而不使用内置缓存应该更高效。

ssl_session_ticket_key

语法	`ssl_session_ticket_key` *file*;
默认	—
上下文	http, server

设置用于加密和解密 TLS 会话票据的秘密密钥文件。如果必须在多个服务器之间共享相同的密钥，则需要该指令。默认情况下，使用随机生成的密钥。

如果指定了多个密钥，则仅使用第一个密钥来加密 TLS 会话票据。这允许配置密钥轮换，例如：

```
ssl_session_ticket_key current.key;
ssl_session_ticket_key previous.key;
```

该文件必须包含 80 或 48 字节的随机数据，可以使用以下命令创建：

```
openssl rand 80 > ticket.key
```

根据文件大小，使用 AES256（对于 80 字节密钥）或 AES128（对于 48 字节密钥）进行加密。

ssl_session_tickets

语法	`ssl_session_tickets on | off;`
默认	`ssl_session_tickets on;`
上下文	http, server

启用或禁用通过 TLS 会话票据 的会话恢复。

ssl_session_timeout

语法	`ssl_session_timeout` *time*;
默认	`ssl_session_timeout 5m;`
上下文	http, server

指定客户端可以重用会话参数的时间。

ssl_stapling

语法	`ssl_stapling on | off;`
默认	`ssl_stapling off;`
上下文	http, server

启用或禁用服务器的 OCSP 响应的 stapling。示例：

```
ssl_stapling on;
resolver 127.0.0.53;
```

为了使 OCSP stapling 生效，服务器证书颁发者的证书应为已知。如果*ssl_certificate* 文件不包含中间证书，则服务器证书颁发者的证书应在*ssl_trusted_certificate* 文件中存在。

> **注意**
>
> 为了解析 OCSP 响应者主机名，*resolver* 指令也应被指定。

ssl_stapling_file

语法	`ssl_stapling_file` *file*;
默认	—
上下文	http, server

当设置时，密封的 OCSP 响应将从指定文件中获取，而不是查询服务器证书中指定的 OCSP 响应者。

该文件应为由 `openssl ocsp` 命令生成的 DER 格式。

ssl_stapling_responder

语法	`ssl_stapling_responder uri;`
默认	—
上下文	http, server

重写在 "Authority Information Access" 证书扩展中指定的 OCSP 响应者的 URI。

仅支持 `http://` OCSP 响应者：

```
ssl_stapling_responder http://ocsp.example.com/;
```

ssl_stapling_verify

语法	`ssl_stapling_verify on | off;`
默认	`ssl_stapling_verify off;`
上下文	http, server

启用或禁用服务器对 OCSP 响应的验证。

为了使验证正常工作，服务器证书颁发者的证书、根证书和所有中间证书应使用*ssl_trusted_certificate* 指令配置为可信。

ssl_trusted_certificate

语法	`ssl_trusted_certificate` *file*;
默认	—
上下文	http, server

指定用于验证 客户端证书和 OCSP 响应的 PEM 格式的受信任 CA 证书文件，如果启用了*ssl_stapling*。

与*ssl_client_certificate* 设置的证书不同，这些证书的列表不会发送给客户端。

ssl_verify_client

语法	`ssl_verify_client on | off | optional | optional_no_ca;`
默认值	`ssl_verify_client off;`
上下文	http, server

启用客户端证书的验证。验证结果存储在*$ssl_client_verify* 变量中。

`optional`	请求客户端证书，并在证书存在时对其进行验证。
`optional_no_ca`	请求客户端证书，但不要求其由受信任的 CA 证书签名。这是为在 Angie 之外的服务执行实际证书验证的情况而设计的。

ssl_verify_depth

语法	`ssl_verify_depth` *number*;
默认值	`ssl_verify_depth 1;`
上下文	http, server

设置客户端证书链的验证深度。

错误处理

`http_ssl` 模块支持多个非标准错误代码，可用于通过*error_page* 指令进行重定向：

495	在客户端证书验证期间发生错误；
496	客户端未提供所需的证书；
497	向 HTTPS 端口发送了常规请求。

重定向在请求被完全解析后发生，此时变量如*$request_uri*、*$uri*、*$args* 等已可用。

指定要在代理服务器响应头的”Set-Cookie” 字段的路径属性中更改的文本。重定向在请求完全解析后进行，此时变量如*$request_uri*、*$uri*、*$args* 和其他变量已可用。

内置变量

http_ssl 模块支持内置变量：

`$ssl_alpn_protocol`

返回 SSL 握手期间 ALPN 选择的协议，否则返回空字符串。

`$ssl_cipher`

返回用于建立 SSL 连接的密码套件名称。

`$ssl_ciphers`

返回客户端支持的密码套件列表。已知的密码套件以名称列出，未知的以十六进制显示，例如：

```
AES128-SHA:AES256-SHA:0x00ff
```

重要

该变量仅在使用 OpenSSL 1.0.2 或更高版本时完全支持。使用旧版本时，该变量仅在新会话中可用，并且仅列出已知的密码套件。

`$ssl_client_escaped_cert`

返回已建立 SSL 连接的客户端证书的 PEM 格式（URL 编码）；

`$ssl_client_fingerprint`

返回已建立 SSL 连接的客户端证书的 SHA1 指纹；

`$ssl_client_i_dn`

返回根据 RFC 2253 的已建立 SSL 连接的客户端证书的” 颁发者 DN” 字符串；

`$ssl_client_i_dn_legacy`

返回已建立 SSL 连接的客户端证书的” 颁发者 DN” 字符串。

`$ssl_client_raw_cert`

返回已建立 SSL 连接的客户端证书的 PEM 格式。

`$ssl_client_s_dn`

返回根据 RFC 2253 的已建立 SSL 连接的客户端证书的” 主体 DN” 字符串。

`$ssl_client_s_dn_legacy`

返回已建立 SSL 连接的客户端证书的" 主体 DN" 字符串。

`$ssl_client_serial`

返回已建立 SSL 连接的客户端证书的序列号。

`$ssl_client_v_end`

返回客户端证书的结束日期。

`$ssl_client_v_remain`

返回客户端证书到期前的天数。

`$ssl_client_v_start`

返回客户端证书的开始日期。

`$ssl_client_verify`

返回客户端证书验证的结果：`SUCCESS`、`FAILED:reason` 和 `NONE` （如果未提供证书）。

`$ssl_curve`

返回用于 SSL 握手密钥交换过程的协商曲线。已知的曲线以名称列出，未知的以十六进制显示，例如：

```
prime256v1
```

重要

该变量仅在使用 OpenSSL 3.0 或更高版本时支持。使用旧版本时，该变量值将为空字符串。

`$ssl_curves`

返回客户端支持的曲线列表。已知的曲线以名称列出，未知的以十六进制显示，例如：

```
0x001d:prime256v1:secp521r1:secp384r1
```

重要

该变量仅在使用 OpenSSL 1.0.2 或更高版本时支持。使用旧版本时，该变量值将为空字符串。

该变量仅对新会话可用。

`$ssl_early_data`

如果使用 TLS 1.3 *early* 数据且握手尚未完成，则返回"1"，否则返回""。

`$ssl_protocol`

返回已建立 SSL 连接的协议。

`$ssl_server_cert_type`

根据服务器证书和密钥的类型，取值 `RSA`、`DSA`、`ECDSA`、`ED448`、`ED25519`、`SM2`、`RSA-PSS` 或 `unknown`。

`$ssl_server_name`

返回通过 SNI 请求的服务器名称；

`$ssl_session_id`

返回已建立 SSL 连接的会话标识符。

`$ssl_session_reused`

如果 SSL 会话被重用，则返回 `r`，否则返回""。

Stub Status

该模块提供对基本服务器状态信息的访问。

当从源代码中 构建时，默认不会构建此模块；应通过 `--with-http_stub_status_module` 构建选项来启用。

在来自 我们的仓库的软件包和镜像中，该模块已包含在构建中。

配置示例

```
location = /basic_status {
    stub_status;
}
```

此配置创建了一个简单的网页，显示基本状态信息，如下所示：

```
Active connections: 291
server accepts handled requests
 16630948 16630948 31070465
Reading: 6 Writing: 179 Waiting: 106
```

指令

stub_status

语法	stub_status;
默认值	—
上下文	server, location

状态信息将可从所在位置访问。

数据

提供以下状态信息：

`Active connections`

当前活跃客户端连接数，包括等待连接。

`accepts`

接受的客户端连接总数。

`handled`

处理的连接总数。通常，该参数值与 accepts 相同，除非达到某些资源限制（例如，*worker_connections* 限制）。

`requests`

客户端请求总数。

`Reading`

当前正在读取请求头的连接数。

`Writing`

当前正在向客户端写回响应的连接数。

`Waiting`

当前空闲等待请求的客户端连接数。

内置变量

`$connections_active`

与 *Active connections* 的值相同。

`$connections_reading`

与 *Reading* 的值相同。

`$connections_writing`

与 *Writing* 的值相同。

`$connections_waiting`

与 *Waiting* 的值相同。

Sub

该模块是一个过滤器，通过将一个指定字符串替换为另一个字符串来修改响应。

当从源代码 构建时，此模块默认不被构建；它应通过 `--with-http_sub_module` 构建选项来启用。

在来自 我们的存储库的软件包和镜像中，该模块已包含在构建中。

配置示例

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

指令

sub_filter

语法	`sub_filter` *string replacement*;
默认值	—
上下文	http, server, location

设置要替换的字符串和替换字符串。匹配要替换的字符串时不区分大小写。要替换的字符串和替换字符串可以包含变量。在同一配置级别上可以指定多个 `sub_filter` 指令。这些指令仅在当前级别未定义 `sub_filter` 指令的情况下才会从上一级配置继承。

sub_filter_last_modified

语法	`sub_filter_last_modified on | off;`
默认值	`sub_filter_last_modified off;`
上下文	http, server, location

允许在替换期间保留原始响应的”Last-Modified” 头字段，以便于响应缓存。

默认情况下，由于响应内容在处理过程中被修改，头字段会被移除。

sub_filter_once

语法	sub_filter_once on | off;
默认值	sub_filter_once on;
上下文	http, server, location

指示是否仅查找每个要替换的字符串一次或重复查找。

sub_filter_types

语法	sub_filter_types mime-*type* ...;
默认值	sub_filter_types text/html;
上下文	http, server, location

除了 text/html 外，还在指定 MIME 类型的响应中启用字符串替换。特殊值"*" 匹配任何 MIME 类型。

Upstream

该模块用于定义可以通过*proxy_pass*、*fastcgi_pass*、*uwsgi_pass*、*scgi_pass*、*memcached_pass* 和*grpc_pass* 指令引用的服务器组。

配置示例

```
upstream backend {
    zone backend 1m;
    server backend1.example.com       weight=5;
    server backend2.example.com:8080;
    server backend3.example.com       service=_example._tcp resolve;
    server unix:/tmp/backend3;

    server backup1.example.com:8080   backup;
    server backup2.example.com:8080   backup;
}

resolver 127.0.0.53 status_zone=resolver;

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

指令

backup_switch (PRO)

Added in version 1.9.0: PRO

语法	`backup_switch permanent`[=*time*];
默认值	—
上下文	upstream

该指令为出现的位置 `upstream` 启用主动备用功能。当请求无法在主服务器组中选择服务器，并转而使用备用组时，如果定义了此指令，备用组将成为 活动组。后续请求将优先尝试使用活动组中的服务器。

当 `permanent` 被定义且没有指定 *time* 值时，一旦选择，备用组将一直保持活跃，不会自动重新评估。如果设置了 *time* 限制，那么在指定的时间间隔 后，活动状态将超时，平衡器会重新评估主服务器组，如果服务器健康，则切换回主组。

示例：

```
upstream my_backend {
    server primary1.example.com;
    server primary2.example.com;

    server backup1.example.com backup;
    server backup2.example.com backup;

    backup_switch permanent=2m;
}
```

如果负载均衡器从主服务器切换到备用组，所有后续请求将在 2 分钟内由该备用组处理。2 分钟过后，负载均衡器会重新评估主服务器，如果发现服务器健康，则重新激活主服务器组。

bind_conn (PRO)

语法	`bind_conn` *value*;
默认	—
上下文	upstream

当 *value* 被设置为变量字符串并变为非空值 `""` 和 `"0"` 之外的值时，启用将服务器连接绑定到客户端。

> **注意**
>
> `bind_conn` 指令必须在所有设置负载均衡方法的指令之后使用；否则，它将不起作用。如果同时使用*sticky*，`bind_conn` 应该出现在 `sticky` 之后。

注意

使用指令时，配置*http_proxy* 模块以允许保持连接，例如：

```
proxy_http_version 1.1;
proxy_set_header Connection "";
```

该指令的一个典型用例是代理 NTLM 认证的连接，其中客户端在协商开始时应该绑定到服务器：

```
map $http_authorization   $ntlm {
    ~*^N(?:TLM|egotiate)  1;
}

upstream ntlm_backend {
    server 127.0.0.1:8080;
    bind_conn $ntlm;
}

server {
    # ...
    location / {
        proxy_pass http://ntlm_backend;
        proxy_http_version 1.1;
        proxy_set_header Connection "";
        # ...
    }
}
```

feedback (PRO)

Added in version 1.6.0: PRO

语法	`feedback` *variable* [`inverse`] [`factor=`*number*] [`account=`*condition_variable*] [`last_byte`];
默认	—
上下文	upstream

启用基于反馈的负载均衡机制用于 `upstream`。它动态调整负载均衡决策，将每个节点的权重乘以其平均反馈值，该值受随时间变化的 *variable* 的影响，并受可选条件的制约。

接受以下参数：

variable	反馈值来源的变量。它应表示性能或健康指标，并旨在通过节点在头字段或其他方式提供。 在每次从节点的响应中评估该值，并根据 inverse 和 factor 设置纳入滚动平均值。
inverse	如果设置，反馈值被反向解释，即较低的值表示更好的性能。
factor	计算平均值时反馈值的权重因子。有效值为 0 到 99 之间的整数。默认值—90。 使用 exponential moving average 公式计算平均反馈。 因子越大，新的值对平均值的影响越小；如果因子设置为 90，结果将有 90% 来自先前的值，仅 10% 来自新值。
account	指定一个条件变量控制应包括哪些响应进行计算。仅当响应的条件变量不为 "" 或 "0" 时，平均值才会使用反馈值进行更新。 **备注** 默认情况下，来自*probes* 的响应不包括在计算中；将*$upstream_probe* 变量与 account 结合使用可包含这些响应，或甚至排除其他所有响应。
last_byte	允许在接收到完整响应后处理来自上游服务器的反馈，而不是仅在接收到头部后处理。

示例：

```
upstream backend {

    zone backend 1m;

    feedback $feedback_value factor=80 account=$condition_value;

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

map $upstream_http_custom_score $feedback_value {
    "high"                      100;
    "medium"                    75;
    "low"                       50;
    default                     10;
}

map $upstream_probe $condition_value {
    "high_priority" "1";
    "low_priority"  "0";
    default         "1";
}
```

这将基于从响应头字段获取的特定分数将服务器响应分类为不同的反馈级别，并且还添加了一个从*$upstream_probe* 映射的条件，以仅考虑来自 high_priority 探测的响应或常规客户端请求的响应。

hash

语法	`hash` *key* [`consistent`];
默认	—
上下文	upstream

指定服务器组的负载均衡方法，其中客户端和服务器的映射基于哈希键值。键可以包含文本、变量及其组合。请注意，从组中添加或移除服务器可能会导致大多数键映射到不同的服务器。该方法与 Cache::Memcached Perl 库兼容。

如果指定了 `consistent` 参数，则将使用 ketama 一致性哈希方法。该方法确保在向组中添加或移除服务器时，只有少量键会被重新映射到不同的服务器。这有助于提高缓存服务器的缓存命中率。该方法与设置为 160 的 ketama_points 参数的 Cache::Memcached::Fast Perl 库兼容。

ip_hash

语法	`ip_hash`;
默认	—
上下文	upstream

指定应使用负载均衡方法，其中请求根据客户端 IP 地址在服务器之间分配。客户端 IPv4 地址的前三个八位字节或整个 IPv6 地址被用作哈希键。该方法确保来自同一客户端的请求将始终传递到同一服务器，除非该服务器不可用。在后一种情况下，客户端请求将传递到另一服务器。很可能，它将始终是同一服务器。

如果需要临时移除其中一台服务器，应使用 `down` 参数进行标记，以保持当前的客户端 IP 地址哈希。

```
upstream backend {
    ip_hash;

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

keepalive

语法	`keepalive` *connections*;
默认	—
上下文	upstream

激活与上游服务器的连接缓存。

`connections` 参数设置每个工作进程中保存在缓存中的最大空闲保持连接数。当超过该数字时，将关闭最近最少使用的连接。

备注

特别需要指出的是，*keepalive* 指令并不限制 Angie 工作进程可以打开的与上游服务器的总连接数。连接参数应设置为足够小的数字，以便让上游服务器处理新的传入连接。

注意

`keepalive` 指令必须在所有设置负载均衡方法的指令之后使用；否则，它将不起作用。

带有保持连接的 memcached 上游的示例配置：

```
upstream memcached_backend {
    server 127.0.0.1:11211;
    server 10.0.0.2:11211;

    keepalive 32;
}

server {
    #...

    location /memcached/ {
        set $memcached_key $uri;
        memcached_pass memcached_backend;
    }

}
```

对于 HTTP，*proxy_http_version* 指令应设置为”1.1”，并清除”Connection” 头字段：

```
upstream http_backend {
    server 127.0.0.1:8080;

    keepalive 16;
}

server {
    #...

    location /http/ {
        proxy_pass http://http_backend;
        proxy_http_version 1.1;
        proxy_set_header Connection "";
```

```
    #    ...
    }
}
```

备注

另外，可以通过将"Connection: Keep-Alive" 头字段传递给上游服务器来使用 HTTP/1.0 持久连接，尽管不建议使用这种方法。

对于 FastCGI 服务器，需要设置*fastcgi_keep_conn* 以使保持连接工作：

```
upstream fastcgi_backend {
    server 127.0.0.1:9000;

    keepalive 8;
}

server {
    #...

    location /fastcgi/ {
        fastcgi_pass fastcgi_backend;
        fastcgi_keep_conn on;
    #    ...
    }
}
```

备注

SCGI 和 uwsgi 协议没有定义保持连接的语义。

keepalive_requests

语法	`keepalive_requests` *number*;
默认	`keepalive_requests 1000;`
上下文	upstream

设置可以通过一个保持连接提供的最大请求数。达到最大请求数后，连接将被关闭。

定期关闭连接是必要的，以释放每个连接的内存分配。因此，使用过高的最大请求数可能会导致过度的内存使用，不推荐这样做。

keepalive_time

语法	`keepalive_time` *time*;
默认	`keepalive_time 1h;`
上下文	upstream

限制通过一个保持连接处理请求的最大时间。达到此时间后，连接将在后续请求处理完成后关闭。

keepalive_timeout

语法	`keepalive_timeout` *timeout*;
默认	`keepalive_timeout 60s;`
上下文	upstream

设置与上游服务器的空闲保持连接保持打开状态的超时时间。

least_conn

语法	`least_conn`;
默认值	—
上下文	upstream

指定一组应该使用负载均衡方法，其中请求将传递给活动连接数最少的服务器，同时考虑服务器的权重。如果有多个这样的服务器，则按权重轮询的负载均衡方法依次尝试。

least_time (PRO)

语法	`least_time header` | `last_byte` [`factor=`*number*] [`account=`*condition_variable*];
默认值	—
上下文	upstream

指定该组应使用负载均衡方法，其中活动服务器收到请求的机会与其平均响应时间成反比；响应时间越少，服务器接收的请求越多。

`header`	指令仅考虑响应头。
`last_byte`	指令使用接收整个响应的平均时间。

Added in version 1.7.0: PRO

`factor`	与*response_time_factor (PRO)* 具有相同的目的，如果设置则覆盖。
`account`	指定一个条件变量控制哪些响应应包含在计算中。平均值仅在响应的条件变量不是 `""` 或 `"0"` 时更新。 **备注** 默认情况下，来自*probes* 的响应不包括在计算中；将*$upstream_probe* 变量与 `account` 结合使用可以包括这些响应或甚至排除其他所有响应。

相应的移动平均值，经过 `factor` 和 `account` 调整，也作为 `header_time` 和 `response_time` 显示在服务器的 `health` 对象中，位于 API 中的上游指标。

queue (PRO)

Added in version 1.4.0: PRO

语法	`queue` *number* [`timeout=` 时间];
默认值	—
上下文	upstream

如果在第一次尝试时无法将代理服务器分配给请求（例如，在短暂服务中断期间或当负载激增达到*max_conns* 限制时），请求不会被拒绝；相反，Angie 尝试将其排队以进行处理。

指令中的数字设置了*worker process* 的队列中请求的最大数量。如果队列已满，将向客户端返回 `502 (Bad Gateway)` 错误。

备注

proxy_next_upstream 指令的逻辑也适用于排队请求。具体来说，如果为请求选择了一个服务器但无法将其交给它，请求可能会返回到队列中。

如果在 `timeout` 设置的时间 内未选择服务器来处理排队请求（默认值为 60 秒），将向客户端返回 `502 (Bad Gateway)` 错误。提前关闭连接的客户端请求也会从队列中移除；在*API* 中有通过队列传递的请求计数器。

注意

`queue` 指令必须在所有设置负载均衡方法的指令之后使用；否则，它将无法工作。

random

语法	`random` [`two`];
默认值	—
上下文	upstream

指定一组应该使用负载均衡方法，其中请求将传递给随机选择的服务器，同时考虑服务器的权重。

可选的 `two` 参数指示 Angie 随机选择两个服务器，然后使用指定的方法选择一个服务器。默认方法是*least_conn*，它将请求传递给活动连接数最少的服务器。

response_time_factor (PRO)

语法	`response_time_factor` *number*;
默认值	`response_time_factor 90;`
上下文	upstream

如果使用*least_time (PRO)* 负载均衡方法，设置 ** 前 ** 值的平滑因子用于使用 指数移动平均 公式计算的平均响应时间。

number 越大，新值对平均值的影响越小；如果 *number* 设置为 90，结果将有 90% 来自于之前的值，仅有 10% 来自于新值。允许的范围是 0 到 99（包括）。

相应的移动平均值作为 `header_time`（仅头部）和`:samp:`response_time`（整个响应）在服务器的`:samp:`health` 对象中显示，位于 API 中的上游指标。

备注

计算仅考虑成功响应；什么被视为不成功的响应由*proxy_next_upstream*, *fastcgi_next_upstream*, *uwsgi_next_upstream*, *scgi_next_upstream*, *memcached_next_upstream*, 和*grpc_next_upstream* 指令定义。此外，只有在所有头部被接收和处理后，`header_time` 才会更新，只有在接收到整个响应后，`response_time` 才会更新。

server

语法	`server` *address* [*parameters*];
默认值	—
上下文	upstream

定义服务器的地址和其他参数。地址可以指定为域名或 IP 地址，带可选端口，或作为 UNIX 域套接字路径，指定在 `unix:` 前缀后。如果未指定端口，则使用 80 端口。解析为多个 IP 地址的域名一次定义多个服务器。

可以定义以下参数：

`weight`=*number*	设置服务器的权重；默认为 1。
`max_conns`=*number*	限制到代理服务器的最大同时活动连接数。默认值为 `0`，表示没有限制。如果服务器组不驻留在共享内存 中，则限制适用于每个工作进程。

备注

如果启用了空闲*keepalive* 连接、多个*workers* 和共享内存，则对代理服务器的活动和空闲连接的总数可能会超过 *max_conns* 值。

`max_fails`=*number* —设置应在*fail_timeout* 设置的持续时间内发生的与服务器通信的失败尝试次数，以将服务器视为不可用；然后在相同的持续时间后重试。

不成功的尝试被视为由*proxy_next_upstream*，*fastcgi_next_upstream*，*uwsgi_next_upstream*，*scgi_next_upstream*，*memcached_next_upstream*，和*grpc_next_upstream* 指令定义。

当达到 `max_fails` 时，服务器也会被*upstream_probe (PRO)* 探针视为不健康；它将不会接收客户端请求，直到探针重新认为它健康。

备注

如果一个组中的 `server` 指令解析为多个服务器，则其 `max_fails` 设置适用于每个服务器。

如果在解析所有 `server` 指令后，上游仅包含一个服务器，则 `max_fails` 设置将无效并被忽略。

`max_fails=1`	不成功尝试的默认次数
`max_fails=0`	禁用尝试的计数

`fail_timeout`=*time* —设置在多长时间内应发生与服务器通信的失败尝试次数 (*max_fails*) 才能将服务器视为不可用。然后，服务器在同样的时间内保持不可用状态然后才会重试。

默认情况下，这设置为 10 秒。

备注

如果一个组中的 `server` 指令解析为多个服务器，则其 `fail_timeout` 设置适用于每个服务器。

如果在解析所有 `server` 指令后，上游仅包含一个服务器，则 `fail_timeout` 设置将无效并被忽略。

`backup`	将服务器标记为备用服务器。当主服务器不可用时，将传递请求。 如果配置了*backup_switch (PRO)*，其主动备用逻辑也会被应用。
`down`	将服务器标记为永久不可用。
`drain` (PRO)	将服务器标记为排水状态；这意味着它仅接收来自之前通过*sticky* 绑定的会话的请求。否则，它的行为类似于 `down`。

小心

参数 `backup` 不能与*hash*、*ip_hash* 和*random* 负载均衡方法一起使用。

`down` 和 `drain` 参数是互斥的。

Added in version 1.1.0.

`resolve`	启用监控与域名对应的 IP 地址列表的更改，无需重新加载配置即可更新它。该组应该存储在共享内存区域 中；此外，您需要定义一个*resolver*。
`service=`*name*	启用解析 DNS SRV 记录并设置服务名称。为了使该参数生效，请指定 *resolve* 服务器参数，提供无端口号的主机名。 如果服务名称中没有点，则该名称根据 RFC 标准形成：服务名称以 _ 为前缀，然后在点后添加 `_tcp`。因此，服务名称 `http` 将结果为 `_http._tcp`。 Angie 通过组合归一化的服务名称和主机名解析 SRV 记录，并通过 DNS 获取该组合的服务器列表，以及它们的优先级和权重。 • 优先级最高的 SRV 记录（共享最小优先级值的记录）解析为主服务器，其他记录成为备份服务器。如果 `backup` 与 `server` 一起设置，优先级最高的 SRV 记录解析为备份服务器，其他记录将被忽略。 • 权重类似于 `server` 指令的 `weight` 参数。如果同时通过指令和 SRV 记录设置权重，则使用指令设置的权重。

这个示例将查找 `_http._tcp.backend.example.com` 记录：

```
server backend.example.com service=http resolve;
```

Added in version 1.2.0: Angie

Added in version 1.1.0-P1: Angie PRO

`sid=`*id*	设置组内的服务器 ID。如果参数未设置，ID 将设置为 IP 地址和端口或 UNIX 套接字路径的十六进制 MD5 哈希值。

Added in version 1.4.0.

`slow_start=`*time*	设置*time* 来恢复服务器的 `weight` 当它重新上线时，如果负载均衡使用*round-robin* 或*least_conn* 方法。 如果设置了该值，且服务器被再次视为可用且健康，按照*max_fails* 和*upstream_probe (PRO)* 的定义，服务器将在分配的时间框架内逐渐恢复其指定的权重。 如果未设置该值，类似情况下的服务器将立即恢复其指定的权重。

备注

如果上游中只有一个 `server`，`slow_start` 将无效并被忽略。

state (PRO)

Added in version 1.2.0: PRO

Syntax	`state` *file*;
Default	—
Context	upstream

指定了上游服务器列表持久化的 *file*。从 我们的包安装时，会创建一个指定的 `/var/lib/angie/state/`（在 FreeBSD 上为 `/var/db/angie/state/`）目录，并赋予适当的权限来存储这些文件，因此您只需在配置中添加文件的基本名称：

```
upstream backend {

    zone backend 1m;
    state /var/lib/angie/state/<文件名>;
}
```

该服务器列表的格式类似于 `server`。文件的内容在通过配置 API 对*/config/http/upstreams/* 部分中的服务器进行任何修改时会发生变化。该文件在 Angie 启动或配置重载时读取。

小心

要在 `upstream` 块中使用 `state` 指令，该块不能有 `server` 指令；而是必须有一个共享内存区域 (*zone*)。

sticky

Added in version 1.2.0: Angie

Added in version 1.1.0-P1: Angie PRO

Syntax	`sticky` cookie *name* [*attr=value*]...; `sticky` route *$variable*...; `sticky` learn `zone=`*zone* `create=`*$create_var1*... `lookup=`*$lookup_var1*... [`header`] [`norefresh`] [`timeout=`*time*]; `sticky` learn [`zone=`*zone*] `lookup=`*$lookup_var1*... `remote_action=`*uri* `remote_result=`*$remote_var* [`norefresh`] [`timeout=`*time*];
Default	—
Context	upstream

配置客户端会话与代理服务器的绑定采用第一个参数指定的模式；要从定义了 `sticky` 的服务器中排出请求，请在*server* 块中使用 `drain` 选项 (PRO)。

注意

`sticky` 指令必须在所有设置负载均衡方法的指令之后使用；否则，它将不起作用。如果同时使用*bind_conn (PRO)*，`bind_conn` 应该出现在 `sticky` 之后。

`cookie` 模式

此模式使用 cookie 来维持会话持久性。它更适合于已经使用 cookie 进行会话管理的情况。

在这里，尚未绑定到任何服务器的客户端请求会被发送到根据配置的负载均衡方法选择的服务器。同时，Angie 设置一个 cookie，其中包含标识服务器的唯一值。

cookie 的名称（`name`）由 `sticky` 指令设置，而值（`value`）对应于*server* 指令的*sid* 参数。注意，如果设置了*sticky_secret* 指令，该参数会被额外哈希处理。

包含此 cookie 的后续客户端请求会被转发到由 cookie 值标识的服务器，即具有指定*sid* 的服务器。如果选择服务器失败或所选服务器无法处理请求，则会根据配置的负载均衡方法选择另一个服务器。

该指令允许为 cookie 分配属性；默认情况下唯一设置的属性是 `path=/`。属性值被指定为带变量的字符串。要移除属性，请为其设置空值：`attr=`。因此，`sticky cookie path=` 创建一个没有 `path` 的 cookie。

在这里，Angie 创建一个名为 `srv_id` 的 cookie，生存期为一小时，并使用变量指定的域：

```
upstream backend {
    server backend1.example.com:8080;
    server backend2.example.com:8080;

    sticky cookie srv_id domain=$my_domain max-age=3600;
}
```

`route` 模式

此模式使用预定义的路由标识符，这些标识符可以嵌入到 URL、cookie 或其他请求属性中。它不太灵活，因为它依赖于预定义的值，但如果这些标识符已经存在，则可能更适合。

在这里，当代理服务器接收到请求时，它可以为客户端分配一个路由并返回其标识符，使客户端和服务器都能识别。*server* 指令的*sid* 参数的值必须用作路由标识符。注意，如果设置了*sticky_secret* 指令，该参数会被额外哈希处理。

希望使用此路由的客户端的后续请求必须包含服务器发出的标识符，确保它最终出现在 Angie 变量中，例如，在*cookie* 或请求参数 中。

该指令列出了用于路由的特定变量。要选择转发传入请求的服务器，使用第一个非空变量；然后将其与*server* 指令的*sid* 参数进行比较。如果选择服务器失败或所选服务器无法处理请求，则会根据配置的负载均衡方法选择另一个服务器。

在这里，Angie 在 `route` cookie 中查找路由标识符，然后在 `route` 请求参数中查找：

```
upstream backend {
    server backend1.example.com:8080 "sid=server 1";
    server backend2.example.com:8080 "sid=server 2";

    sticky route $cookie_route $arg_route;
}
```

`learn` 模式 (PRO 1.4.0+)

此模式使用动态生成的密钥将客户端与特定的代理服务器关联；它更灵活，因为它动态分配服务器，在共享内存区域中存储会话，并支持不同的会话标识符传递方式。

在这里，会话是基于代理服务器的响应创建的。`create` 和 `lookup` 参数列出了变量，指示如何创建新会话以及如何查找现有会话。这两个参数都可以多次出现。

会话标识符是用 `create` 指定的第一个非空变量的值；例如，这可能是来自代理服务器的 *cookie*。

会话存储在共享内存区域中；其名称和大小由 `zone` 参数设置。如果会话在由 `timeout` 设置的时间 内处于非活动状态，它将被删除。默认值为 10 分钟。

默认情况下，Angie 会延长会话生命周期，在每次使用时更新最后访问时间戳。`norefresh` 参数禁用此行为：会话将严格按超时过期，即使它继续被使用。当需要在一段时间后强制终止会话时，此模式很有用，例如，在与外部会话管理器集成时。

希望使用会话的客户端的后续请求必须包含其标识符，确保它最终出现在用 `lookup` 指定的非空变量中；然后将其值与共享内存中的会话进行匹配。如果选择服务器失败或所选服务器无法处理请求，则会根据配置的负载均衡方法选择另一个服务器。

`header` 参数允许在接收到代理服务器的响应头后立即创建会话。没有它，会话只有在请求处理完成后才会创建。

在示例中，Angie 创建一个会话，在响应中设置一个名为 `examplecookie` 的 cookie：

```
upstream backend {
    server backend1.example.com:8080;
    server backend2.example.com:8080;

    sticky learn
        create=$upstream_cookie_examplecookie
```

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

带 `remote_action` 的 `learn` 模式 (PRO 1.8.0+)

`remote_action` 和 `remote_result` 参数通过远程会话存储启用动态分配和管理会话 ID。在这里，共享内存区域充当本地缓存，而远程存储是权威来源。因此，`create` 参数与 `remote_action` 不兼容，因为会话 ID 需要远程创建。如果会话在由 `timeout` 设置的时间 内处于非活动状态，它将被删除。`remote_action` 设置不影响超时。默认值为 10 分钟。

初始会话 ID 始终来自 `lookup`；如果它能在本地共享内存中找到，Angie 会继续选择适当的服务器。

如果在本地找不到此会话 ID，Angie 会向远程存储发送同步子请求。`remote_action` 参数设置远程存储的 URI，它应该按如下方式处理会话查找和创建：

- 存储接受来自 `lookup` 的会话 ID 和与此会话关联的本地建议的服务器 ID，通过自定义头或其他方式。

  在 Angie 端，为此目的提供了两个特殊变量：分别是*$sticky_sessid* 和*$sticky_sid*。`sticky_sid` 包含*upstream* 块中 `server` 指令的 `sid=` 参数的值（如果设置），或服务器名称的 MD5 哈希值。

- 来自远程存储的 200 响应表示它已接受会话并保存了建议的值以供将来使用。
- 来自远程存储的 409 响应表示此会话 ID 已存在。在这种情况下，响应应在 `X-Sticky-Sid` 头中包含替代会话 ID。Angie 将此 ID 保存在由 `remote_result` 参数设置的变量中。

在以下示例中，Angie 创建一个会话，使用 `$cookie_bar` 变量作为初始会话 ID，并将远程存储返回的替代会话 ID 存储在 `$upstream_http_x_sticky_sid` 中：

```
http {

    upstream u1 {

        server srv1;
        server srv2;

        sticky learn zone=sz:1m
            lookup=$cookie_bar
            remote_action=/remote_session
            remote_result=$upstream_http_x_sticky_sid;

        zone z 1m;
    }

    server {

        listen localhost;

        location / {
```

```
            proxy_pass http://u1/;
        }

        location /remote_session {

            internal;
            proxy_set_header X-Sticky-Sessid $sticky_sessid;
            proxy_set_header X-Sticky-Sid $sticky_sid;
            proxy_set_header X-Sticky-Last $msec;
            proxy_pass http://remote;
        }
    }
}
```

每次本地记录未命中或超时过期时（考虑 `norefresh`），都会向 `remote_action` 中指定的资源发出子请求。

`sticky` 配置中的 `zone` 参数是可选的。如果未设置，Angie 完全依赖远程存储：它不在本地缓存会话（尽管它允许通过 `proxy_cache` 缓存存储响应）并且每次需要检索或创建会话时都会联系远程存储。

下面是一个简化的配置示例。远程存储在 `X-Sid` 头中返回会话 ID，从而确认或覆盖 Angie 的选择：

```
http {

    proxy_cache_path c1 keys_zone=s1:1m;

    upstream tc_0 {
        server 10.0.0.1 sid=a;
        server 10.0.0.2 sid=b;

        sticky learn
            lookup=$arg_id
            remote_action=@create_session
            remote_result=$upstream_http_x_sid;
    }

    server {
        listen 127.0.0.1:8080;

        location / {
            proxy_pass http://tc_0/;
        }

        # 请求远程会话存储
        location @create_session {
            internal;

            proxy_set_header X-Sticky-Sessid $sticky_sessid;
```

```
            proxy_set_header X-Sticky-Sid    $sticky_sid;
            proxy_set_header X-Sticky-Last   $msec;

            proxy_pass http://session_backend;

            proxy_connect_timeout 1s;
            proxy_read_timeout    1s;

            proxy_cache         s1;
            proxy_cache_valid   200 1d;
            proxy_cache_key     "$scheme$proxy_host$request_uri$sticky_sessid";
        }
    }
}
```

sticky_secret

Added in version 1.2.0: Angie

Added in version 1.1.0-P1: Angie PRO

语法	`sticky_secret` *string*;
默认值	—
上下文	upstream

将 *string* 作为盐值添加到 MD5 哈希函数中用于*sticky* 指令在 `cookie` 和 `route` 模式下。*string* 可以包含变量，例如，*$remote_addr*：

```
upstream backend {
    server backend1.example.com:8080;
    server backend2.example.com:8080;

    sticky cookie cookie_name;
    sticky_secret my_secret.$remote_addr;
}
```

盐值被附加到被哈希的值上；要独立验证哈希机制：

```
$ echo -n "<VALUE><SALT>" | md5sum
```

sticky_strict

Added in version 1.2.0: Angie

Added in version 1.1.0-P1: Angie PRO

语法	`sticky_strict on | off;`
默认值	`sticky_strict off;`
上下文	upstream

启用时，如果所需的服务器不可用，Angie 将返回 HTTP 502 错误给客户端，而不是在没有可用服务器时使用任何其他可用的服务器。

upstream

语法	`upstream` *name* { ... }
默认值	—
上下文	http

定义一组服务器。服务器可以监听不同的端口。此外，可以混合监听 TCP 和 UNIX 域套接字的服务器。

示例：

```
upstream backend {
    server backend1.example.com weight=5;
    server 127.0.0.1:8080       max_fails=3 fail_timeout=30s;
    server unix:/tmp/backend3;

    server backup1.example.com  backup;
}
```

默认情况下，请求在服务器之间使用加权轮询平衡方法分配。在上述示例中，每 7 个请求将按以下方式分配：5 个请求发送到 backend1.example.com，每个第二和第三个服务器各 1 个请求。

如果在与服务器的通信中发生错误，请求将传递给下一个服务器，依此类推，直到所有可用服务器都被尝试。如果无法从任何服务器获得成功响应，客户端将收到与最后一个服务器的通信结果。

zone

语法	`zone` *name* [*size*];
默认值	—
上下文	upstream

定义共享内存区的名称和大小，该内存区保存在工作进程之间共享的组配置和运行时状态。多个组可以共享同一个区域。在这种情况下，只需指定一次大小即可。

内置变量

http_upstream 模块支持以下内置变量：

$sticky_sessid

与*sticky* 中的 remote_action 一起使用；存储从 lookup 中获取的初始会话 ID。

$sticky_sid

与*sticky* 中的 remote_action 一起使用；存储先前与会话相关联的服务器 ID。

$upstream_addr

保存上游服务器的 IP 地址和端口，或 UNIX 域套接字的路径。如果在请求处理期间联系了多个服务器，它们的地址用逗号分隔，例如：

192.168.1.1:80, 192.168.1.2:80, unix:/tmp/sock

如果发生从一个服务器组到另一个服务器组的内部重定向，由"X-Accel-Redirect" 或*error_page* 发起，则来自不同组的服务器地址用冒号分隔，例如：

192.168.1.1:80, 192.168.1.2:80, unix:/tmp/sock : 192.168.10.1:80, 192.168.10.2:80

如果无法选择服务器，该变量将保留*server group* 的 *name*。

$upstream_bytes_received

从上游服务器接收的字节数。来自多个连接的值用逗号和冒号分隔，类似于*$upstream_addr* 变量中的地址。

$upstream_bytes_sent

发送到上游服务器的字节数。来自多个连接的值用逗号和冒号分隔，类似于*$upstream_addr* 变量中的地址。

$upstream_cache_status

保存访问响应缓存的状态。状态可以是 MISS、BYPASS、EXPIRED、STALE、UPDATING、REVALIDATED 或 HIT：

- MISS：缓存中未找到响应，请求被转发到上游服务器。
- BYPASS：绕过缓存，请求直接转发到上游服务器。
- EXPIRED：缓存的响应过期，并向上游服务器发送用于更新内容的新请求。
- STALE：缓存的响应过期，但将提供给客户端直到最终从上游服务器获取更新。
- UPDATING：缓存的响应过期，但将提供给客户端直到当前正在进行的来自上游服务器的更新完成。
- REVALIDATED：缓存的响应过期，但已成功重新验证并且不需要来自上游服务器的更新。

- HIT：响应是从缓存中提供的。

如果完全绕过缓存而未访问它，则该变量未设置。

$upstream_connect_time

保存与上游服务器建立连接所花费的时间；该时间以秒为单位，精确到毫秒。对于 SSL，包括握手所花费的时间。多个连接的时间用逗号和冒号分隔，类似于*$upstream_addr* 变量中的地址。

$upstream_cookie_<name>

由上游服务器在"Set-Cookie" 响应头字段中发送的指定 *name* 的 cookie。仅保存最后一个服务器响应中的 cookie。

$upstream_header_time

保存从上游服务器接收响应头所花费的时间；该时间以秒为单位，精确到毫秒。多个响应的时间用逗号和冒号分隔，类似于*$upstream_addr* 变量中的地址。

$upstream_http_<name>

保存服务器响应头字段。例如，`Server` 响应头字段可以通过 *$upstream_http_server* 变量访问。将头字段名称转换为变量名称的规则与以"$http_" 前缀开头的变量相同。仅保存最后一个服务器响应中的头字段。

$upstream_queue_time

保存请求在选择服务器之前在*queue* 中花费的时间；该时间以秒为单位，精确到毫秒。多个选择尝试的时间用逗号和冒号分隔，类似于*$upstream_addr* 变量中的地址。

$upstream_response_length

保存从上游服务器获取的响应的长度；该长度以字节为单位。多个响应的长度用逗号和冒号分隔，类似于*$upstream_addr* 变量中的地址。

$upstream_response_time

保存接收来自上游服务器的响应所花费的时间；该时间以秒为单位，精确到毫秒。多个响应的时间用逗号和冒号分隔，类似于*$upstream_addr* 变量中的地址。

$upstream_status

保存从上游服务器获取的响应的状态代码。多个响应的状态代码用逗号和冒号分隔，类似于*$upstream_addr* 变量中的地址。如果无法选择服务器，则该变量保留 502（Bad Gateway）状态代码。

`$upstream_sticky_status`

粘性请求的状态。

""	没有启用粘性的请求发送到上游。
NEW	没有粘性信息的请求。
HIT	带有粘性信息的请求路由到所需的后端。
MISS	带有粘性信息的请求路由到通过负载均衡算法选择的后端。

多个连接的值用逗号和冒号分隔，类似于*$upstream_addr* 变量中的地址。

`$upstream_trailer_<name>`

保存从上游服务器获取的响应末尾的字段。

Upstream Probe

该模块实现了针对*Upstream* 的主动健康探测。

配置示例

```
server {
    listen ...;

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

        upstream_probe backend_probe
            uri=/probe
            port=10004
            interval=5s
            test=$good
            essential
            fails=3
            passes=3
            max_body=10m
            mode=idle;
    }
}
```

指令

upstream_probe (PRO)

Added in version 1.2.0: PRO

语法	upstream_probe *name* [uri=*address*] [port=*number*] [interval=*time*] [method=*method*] [test=*condition*] [essential [persistent]] [fails=*number*] [passes=*number*] [max_body=*size*] [mode=always | idle | onfail];
默认	—
上下文	location

为在相同 location 上下文中指定的*proxy_pass*、*uwsgi_pass* 等的*upstream* 组内的服务器定义一个主动健康探测。接下来，Angie 会定期根据指定的参数对上游组中的每个服务器执行请求。

如果请求成功并考虑了 upstream_probe 指令的所有参数设置，以及定义该指令的 location 上下文中控制上游使用的所有参数（包括*proxy_next_upstream* 和*uwsgi_next_upstream* 指令等，以及*proxy_set_header* 等），则服务器通过探测。

要使用探测，上游必须有一个共享内存区 (*zone*)。一个上游可以配置多个探测。

接受以下参数：

name	探测的必需名称。
uri	要添加到*proxy_pass*、*uwsgi_pass* 等参数中的请求 URI。默认值—/。
port	探测请求的替代端口号。
interval	探测之间的间隔。默认值—5s。
method	探测请求的 HTTP 方法。默认值—GET。
test	请求期间要检查的条件；定义为包含变量的字符串。如果变量替换结果为 "" 或 "0"，则探测不通过。
essential	如果设置，服务器的初始状态将被验证，在探测通过前不会转发客户端请求到该服务器。
persistent	设置此参数需要首先启用 essential；在配置重载 前工作的 persistent 服务器开始接收请求，而无需首先通过此探测。
fails	将服务器标记为不健康的连续失败请求次数。默认值—1。
passes	将服务器标记为健康的连续成功请求次数。默认值—1。
max_body	响应体的最大内存量。默认值—256k。
mode	探测模式，取决于服务器的健康状况： • always —无论服务器状态如何都进行探测； • idle —探测不健康的服务器以及自上次客户端请求以来的 interval 已过期的服务器。 • onfail —仅探测不健康的服务器。 默认值—always。

示例：

```
upstream backend {
    zone backend 1m;

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

```
}

map $upstream_status $good {
    200     "1";
}

server {
    listen ...;

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

        upstream_probe backend_probe
            uri=/probe
            port=10004
            interval=5s
            test=$good
            essential
            persistent
            fails=3
            passes=3
            max_body=10m
            mode=idle;
    }
}
```

探测操作的详细信息：

- 最初，服务器不会接收客户端请求，直到它通过为其配置的 所有 `essential` 探测（如果配置被重载且服务器在此之前被认为健康，则跳过 `persistent` 探测）。如果没有这样的探测，服务器被认为是健康的。
- 如果为其配置的 任何探测达到其 `fails` 阈值，或服务器本身达到*max_fails* 阈值，则服务器被认为是不健康的，不会接收客户端请求。
- 要使不健康的服务器再次被认为是健康的，为其配置的 所有探测必须达到各自的 `passes` 阈值；之后，会考虑*max_fails* 阈值。

内置变量

`http_upstream_probe` 模块支持以下内置变量：

`$upstream_probe` (PRO)

当前活动的*upstream_probe* 的名称。

$upstream_probe_body (PRO)

在*upstream_probe* 期间接收到的服务器响应体；其大小由 max_body 限制。

UserID

该模块设置适合客户端识别的 cookies。接收到的和设置的 cookies 可以使用内置变量*$uid_got* 和*$uid_set* 进行记录。该模块与 Apache 的 mod_uid 模块兼容。

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"';
```

指令

userid

语法	userid on | v1 | log | off;
默认值	userid off;
上下文	http, server, location

启用或禁用设置 cookies 和记录接收到的 cookies：

on	启用设置版本 2 的 cookies 和记录接收到的 cookies；
v1	启用设置版本 1 的 cookies 和记录接收到的 cookies；
log	禁用设置 cookies，但启用记录接收到的 cookies；
off	禁用设置 cookies 和记录接收到的 cookies。

userid_domain

语法	userid_domain *name* | none;
默认值	userid_domain none;
上下文	http, server, location

定义设置 cookies 的域。参数 none 禁用设置 cookies 的域。

userid_expires

语法	`userid_expires` *time* | `max` | `off`;
默认值	`userid_expires off;`
上下文	http, server, location

设置浏览器应保持 cookies 的时间。参数 `max` 会导致 cookie 在"2037 年 12 月 31 日 23:55:55 GMT" 过期。参数 `off` 会导致 cookie 在浏览器会话结束时过期。

userid_flags

语法	`userid_flags off` | *flag* ...;
默认值	`userid_flags off;`
上下文	http, server, location

如果参数不是 `off`，则定义一个或多个 cookies 的附加标志：`secure`，`httponly`，`samesite=strict`，`samesite=lax`，`samesite=none`。

userid_mark

语法	`userid_mark` *letter* | *digit* | = | `off`;
默认值	`userid_mark off;`
上下文	http, server, location

如果参数不是 `off`，则启用 cookie 标记机制并设置用作标记的字符。该机制用于在保持客户端标识符的同时添加或更改*userid_p3p* 和/或 cookies 过期时间。标记可以是任何英文字母（区分大小写）、数字或"=" 字符。

如果设置了标记，则将其与传递在 cookie 中的客户端标识符的 base64 表示的第一个填充符号进行比较。如果它们不匹配，则会使用指定的标记、过期时间和"P3P" 头重新发送 cookie。

userid_name

语法	`userid_name` *name*;
默认值	`userid_name uid;`
上下文	http, server, location

设置 cookie 名称。

userid_p3p

语法	`userid_p3p` *string* | `none`;
默认值	`userid_p3p none;`
上下文	http, server, location

设置将与 cookie 一起发送的"P3P" 头字段的值。如果指令设置为特殊值 `none`，则在响应中将不发送"P3P" 头。

userid_path

语法	`userid_path` *path*;
默认值	`userid_path /;`
上下文	http, server, location

定义设置 cookies 的路径。

userid_service

语法	`userid_service` *number*;
默认值	`userid_service` 服务器的 IP 地址;
上下文	http, server, location

如果由多个服务器（服务）发出标识符，则应为每个服务分配其自己的 `number` 以确保客户端标识符的唯一性。对于版本 1 的 cookies，默认值为零。对于版本 2 的 cookies，默认值为由服务器的 IP 地址的最后四个八位字节组成的数字。

内置变量

$uid_got

cookie 名称和接收到的客户端标识符。

$uid_reset

如果变量设置为非空字符串且不是 `0`，则客户端标识符将被重置。特殊值 `log` 还会导致关于重置标识符的消息输出到*error_log*。

$uid_set

cookie 名称和发送的客户端标识符。

uWSGI

允许将请求传递给 uwsgi 服务器。

配置示例

```
location / {
    include    uwsgi_params;
    uwsgi_pass localhost:9000;
}
```

指令

uwsgi_bind

语法	`uwsgi_bind` 地址 [`transparent`] | `off`;
默认	—
上下文	http, server, location

使与 uwsgi 服务器的外部连接从指定的本地 IP 地址发起，并可选指定端口。参数值可以包含变量。特殊值 `off` 取消从上一个配置级别继承的 *uwsgi_bind* 指令的效果，使系统可以自动分配本地 IP 地址和端口。

`transparent` 参数允许与 uwsgi 服务器的外部连接从非本地 IP 地址发起，例如，从客户端的真实 IP 地址：

```
uwsgi_bind $remote_addr transparent;
```

为了使该参数生效，通常需要以超级用户 权限运行 Angie 工作进程。在 Linux 上不需要，因为如果指定了 `transparent` 参数，工作进程将从主进程继承 *CAP_NET_RAW* 能力。

> **重要**
>
> 必须配置内核路由表以拦截来自 uwsgi 服务器的网络流量。

uwsgi_buffer_size

语法	`uwsgi_buffer_size` 大小;
默认	`uwsgi_buffer_size 4k|8k;`
上下文	http, server, location

设置用于读取从 uwsgi 服务器接收到的响应第一部分的缓冲区大小。该部分通常包含一个小的响应头。默认情况下，缓冲区大小等于一个内存页，这取决于平台，可能是 4K 或 8K。不过，也可以将其设置得更小。

uwsgi_buffering

语法	`uwsgi_buffering` 大小;
默认	`uwsgi_buffering on;`
上下文	http, server, location

启用或禁用对来自 uwsgi 服务器的响应的缓冲。

`on`	Angie 尽快从 uwsgi 服务器接收响应，并将其存储到由*uwsgi_buffer_size* 和*uwsgi_buffers* 指令设置的缓冲区中。如果整个响应无法放入内存，则部分响应可以保存到磁盘上的临时文件 中。写入临时文件由*uwsgi_max_temp_file_size* 和*uwsgi_temp_file_write_size* 指令控制。
`off`	响应会同步地立即传递给客户端。Angie 不会尝试从 uwsgi 服务器读取整个响应。Angie 可以一次接收的最大数据量由*uwsgi_buffer_size* 指令设置。

缓冲也可以通过在" X-Accel-Buffering" 响应头字段中传递" yes" 或" no" 来启用或禁用。可以使用*uwsgi_ignore_headers* 指令禁用此功能。

uwsgi_buffers

语法	`uwsgi_buffers` 数量 大小;
默认	`uwsgi_buffers 8 4k	8k;`
上下文	http, server, location

设置用于从 uwsgi 服务器读取响应的缓冲区的数量和大小，针对单个连接。

默认情况下，缓冲区大小等于一个内存页，这取决于平台，可能是 4K 或 8K。

uwsgi_busy_buffers_size

语法	`uwsgi_busy_buffers_size` 大小;
默认	`uwsgi_busy_buffers_size 8k	16k;`
上下文	http, server, location

当启用缓冲 时，限制在响应尚未完全读取时，正在忙于向客户端发送响应的缓冲区的总大小。与此同时，其余的缓冲区可以用于读取响应，如果需要，可以将部分响应缓冲到临时文件中。

默认情况下，大小由*uwsgi_buffer_size* 和*uwsgi_buffers* 指令设置的两个缓冲区的大小限制。

uwsgi_cache

语法	uwsgi_cache 区域 | off;
默认	uwsgi_cache off;
上下文	http, server, location

定义用于缓存的共享内存区域。相同的区域可以在多个地方使用。参数值可以包含变量。

off	禁用从上一个配置级别继承的缓存。

uwsgi_cache_background_update

语法	uwsgi_cache_background_update on | off;
默认	uwsgi_cache_background_update off;
上下文	http, server, location

允许在返回过期缓存项的同时，启动后台子请求以更新过期的缓存项。

注意

请注意，必须允许 使用过期的缓存响应以便进行更新。

uwsgi_cache_bypass

语法	uwsgi_cache_bypass ...;
默认	—
上下文	http, server, location

定义在什么条件下响应不会从缓存中获取。如果字符串参数的至少一个值不为空且不等于”0”，则响应将不会从缓存中获取：

```
uwsgi_cache_bypass $cookie_nocache $arg_nocache$arg_comment;
uwsgi_cache_bypass $http_pragma    $http_authorization;
```

可以与*uwsgi_no_cache* 指令一起使用。

uwsgi_cache_key

语法	uwsgi_cache_key 字符串;
默认	—
上下文	http, server, location

定义缓存的键，例如：

```
uwsgi_cache_key localhost:9000$request_uri;
```

uwsgi_cache_lock

语法	uwsgi_cache_lock on | off;
默认	uwsgi_cache_lock off;
上下文	http, server, location

启用时，仅允许一次一个请求通过将请求传递给 uwsgi 服务器来填充根据*uwsgi_cache_key* 指令识别的新缓存元素。其他相同缓存元素的请求将等待响应出现在缓存中或等待此元素的缓存锁释放，直到*uwsgi_cache_lock_timeout* 指令设置的时间。

uwsgi_cache_lock_age

语法	uwsgi_cache_lock_age 时间;
默认	uwsgi_cache_lock_age 5s;
上下文	http, server, location

如果传递给 uwsgi 服务器以填充新缓存元素的最后请求在指定时间内尚未完成，则可以再向 uwsgi 服务器传递一个请求。

uwsgi_cache_lock_timeout

语法	uwsgi_cache_lock_timeout 时间;
默认	uwsgi_cache_lock_timeout 5s;
上下文	http, server, location

为*uwsgi_cache_lock* 设置超时。当时间到时，请求将被传递给 uwsgi 服务器，但响应不会被缓存。

uwsgi_cache_max_range_offset

语法	uwsgi_cache_max_range_offset 数字;
默认	—
上下文	http, server, location

为字节范围请求设置字节偏移。如果范围超出偏移，将请求传递给 uwsgi 服务器，响应不会被缓存。

uwsgi_cache_methods

语法	uwsgi_cache_methods GET | HEAD | POST ...;
默认	uwsgi_cache_methods GET HEAD;
上下文	http, server, location

如果客户端请求方法在此指令中列出，则响应将被缓存。”GET” 和”HEAD” 方法始终添加到列表中，尽管建议明确指定它们。另见*uwsgi_no_cache* 指令。

uwsgi_cache_min_uses

语法	uwsgi_cache_min_uses 数字;
默认	uwsgi_cache_min_uses 1;
上下文	http, server, location

设置响应被缓存后的请求数量。

uwsgi_cache_path

语法	uwsgi_cache_path 路径 [levels=*levels*] [use_temp_path=on | off] keys_zone=*name:size* [inactive= 时间] [max_size= 大小] [min_free= 大小] [manager_files= 数字] [manager_sleep= 时间] [manager_threshold= 时间] [loader_files= 数字] [loader_sleep= 时间] [loader_threshold= 时间];
默认	—
上下文	http

设置缓存的路径和其他参数。缓存数据存储在文件中。缓存中的文件名是将缓存键 应用 MD5 函数的结果。

The levels 参数定义了缓存的层次级别：从 1 到 3，每个级别接受值 1 或 2。例如，在以下配置中：

```
uwsgi_cache_path /data/angie/cache levels=1:2 keys_zone=one:10m;
```

缓存中的文件名看起来像这样：

```
/data/angie/cache/c/29/b7f54b2df7773722d382f4809d65029c
```

缓存的响应首先写入临时文件，然后该文件被重命名。临时文件和缓存可以放在不同的文件系统上。但是，请注意，在这种情况下，文件会在两个文件系统之间复制，而不是进行简单的重命名操作。因此，建议对于任何给定的位置，缓存和临时文件所在的目录都放在同一个文件系统上。

临时文件的目录基于 `use_temp_path` 参数设置。

`on`	如果此参数被省略或设置为 `on` 的值，将使用为给定 *location* 设置的*uwsgi_temp_path* 指令所指定的目录。
`off`	临时文件将直接放在缓存目录中。

此外，所有活动键和有关数据的信息都存储在共享内存区，其名称和大小由 `keys_zone` 参数配置。一个兆字节的区域可以存储大约 8000 个键。

在 `inactive` 参数指定的时间内未被访问的缓存数据将从缓存中删除，而不管其新鲜程度。

默认情况下，*inactive* 设置为 10 分钟。

一个特殊的 **缓存管理器**进程监控最大缓存大小和文件系统中的最小可用空间，当超出大小或可用空间不足时，它会删除最近最少使用的数据。数据是通过迭代方式删除的。

`max_size`	最大的 uwsgi_cache 大小
`min_free`	文件系统中缓存的最小可用空间
`manager_files`	限制一次迭代中要删除的项目数量 默认值为 100
`manager_threshol`	限制一次迭代的持续时间 默认值为 200 毫秒
`manager_sleep`	配置交互之间的暂停 默认值为 50 毫秒

在 Angie 启动后一分钟，特殊的 **缓存加载器**进程被激活。它从文件系统中加载之前缓存的数据的信息到缓存区。加载也是通过迭代方式进行的。

`loader_files`	限制一次迭代中要加载的项目数量 默认值为 100
`loader_threshold`	限制一次迭代的持续时间 默认值为 200 毫秒
`loader_sleep`	配置交互之间的暂停 默认值为 50 毫秒

uwsgi_cache_revalidate

语法	`uwsgi_cache_revalidate on | off;`
默认值	`uwsgi_cache_revalidate off;`
上下文	http, server, location

启用使用条件请求进行过期缓存项的重新验证，使用” 如果修改过” 和” 如果未匹配” 头字段。

uwsgi_cache_use_stale

语法	`uwsgi_cache_use_stale error | timeout | invalid_header | updating | http_500 | http_503 | http_403 | http_404 | http_429 | off ...;`
默认值	`uwsgi_cache_use_stale off;`
上下文	http, server, location

确定在与 uwsgi 服务器通信时，可以在何种情况下使用过期的缓存响应。指令的参数与*uwsgi_next_upstream* 指令的参数匹配。

`error`	如果无法选择处理请求的 uwsgi 服务器，允许使用过期的缓存响应。
`updating`	额外参数，如果当前正在更新，则允许使用过期的缓存响应。这可以在更新缓存数据时最小化对 uwsgi 服务器的访问次数。

在响应头中直接启用使用过期的缓存响应，可以在响应变为过期后的指定秒数内：

- ”Cache-Control” 头字段的 stale-while-revalidate 扩展允许在当前更新时使用过期的缓存响应。
- ”Cache-Control” 头字段的 stale-if-error 扩展允许在发生错误时使用过期的缓存响应。

备注

这优先级低于使用指令参数。

为了在填充新的缓存元素时最小化对 uwsgi 服务器的访问次数，可以使用*uwsgi_cache_lock* 指令。

uwsgi_cache_valid

语法	`uwsgi_cache_valid` [*code* ...] *time*;
默认值	—
上下文	http, server, location

为不同的响应代码设置缓存时间。例如，以下指令

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

为响应代码为 200 和 302 的响应设置 10 分钟的缓存，为响应代码为 404 的响应设置 1 分钟的缓存。

如果仅指定缓存时间

```
uwsgi_cache_valid 5m;
```

则仅缓存 200、301 和 302 响应。

此外，可以指定 any 参数以缓存任何响应：

```
uwsgi_cache_valid 200 302 10m;
uwsgi_cache_valid 301      1h;
uwsgi_cache_valid any      1m;
```

备注

缓存的参数也可以直接在响应头中设置。这优先级高于使用指令设置的缓存时间。

- ”X-Accel-Expires” 头字段以秒为单位设置响应的缓存时间。零值禁用响应的缓存。如果值以 @ 前缀开头，则设置自 Epoch 以来的绝对时间（以秒为单位），在此时间之前响应可以被缓存。
- 如果头中不包含”X-Accel-Expires” 字段，缓存参数可以在”Expires” 或”Cache-Control” 头字段中设置。
- 如果头中包含”Set-Cookie” 字段，则该响应将不会被缓存。
- 如果头中包含特殊值为”*” 的”Vary” 字段，则该响应将不会被缓存。如果头中包含其他值的”Vary” 字段，则将根据相应的请求头字段缓存该响应。

可以使用*uwsgi_ignore_headers* 指令禁用对一个或多个这些响应头字段的处理。

uwsgi_connect_timeout

语法	`uwsgi_connect_timeout` *time*;
默认值	`uwsgi_connect_timeout 60s;`
上下文	http, server, location

定义与 uwsgi 服务器建立连接的超时。需要注意的是，通常情况下，此超时不能超过 75 秒。

uwsgi_connection_drop

语法	`uwsgi_connection_drop` *time* | `on` | `off`;
默认值	`uwsgi_connection_drop off;`
上下文	http, server, location

启用在代理服务器被从组中移除或被*reresolve* 进程或*API* 命令 DELETE 标记为永久不可用后，终止与代理服务器的所有连接。

当处理客户端或代理服务器的下一个读取或写入事件时，将终止连接。

设置 *time* 启用连接终止的超时；设置为 on 时，连接将立即被断开。

uwsgi_force_ranges

语法	uwsgi_force_ranges off;
默认值	uwsgi_force_ranges off;
上下文	http, server, location

启用对 uwsgi 服务器的缓存和未缓存响应的字节范围支持，无论这些响应中的"Accept-Ranges" 字段如何。

uwsgi_hide_header

语法	uwsgi_hide_header *field*;
默认值	—
上下文	http, server, location

默认情况下，Angie 不将 uwsgi 服务器响应中的:samp:*Status* 和 *:samp:'X-Accel-...'* 头字段传递给客户端。*'uwsgi_hide_header* 指令设置将不传递的附加字段。如果相反需要允许字段的传递，则可以使用*uwsgi_pass_header* 指令。

uwsgi_ignore_client_abort

语法	uwsgi_ignore_client_abort on | off;
默认值	uwsgi_ignore_client_abort off;
上下文	http, server, location

确定当客户端在等待响应时关闭连接时，是否应关闭与 uwsgi 服务器的连接。

uwsgi_ignore_headers

语法	uwsgi_ignore_headers *field* ...;
默认值	—
上下文	http, server, location

禁用处理来自 uwsgi 服务器的某些响应头字段。可以忽略以下字段："X-Accel-Redirect"、"X-Accel-Expires"、"X-Accel-Limit-Rate"、"X-Accel-Buffering"、"X-Accel-Charset"、"Expires"、"Cache-Control"、"Set-Cookie" 和"Vary"。

如果未被禁用，这些头字段的处理将产生以下效果：

- ”X-Accel-Expires”、”Expires”、”Cache-Control”、”Set-Cookie” 和”Vary” 设置响应的缓存 参数；
- ”X-Accel-Redirect” 执行对指定 URI 的内部 重定向；
- ”X-Accel-Limit-Rate” 设置向客户端传输响应的速率限制；
- ”X-Accel-Buffering” 启用或禁用响应的缓冲；
- ”X-Accel-Charset” 设置响应的期望字符集。

uwsgi_intercept_errors

语法	`uwsgi_intercept_errors on | off;`
默认值	`uwsgi_intercept_errors off;`
上下文	http, server, location

决定是否将响应代码大于或等于 300 的 uwsgi 服务器响应传递给客户端，还是拦截并重定向到 Angie 进行处理，使用的是*error_page* 指令。

uwsgi_limit_rate

语法	`uwsgi_limit_rate` *rate*;
默认值	`uwsgi_limit_rate 0;`
上下文	http, server, location

限制从 uwsgi 服务器读取响应的速度。*rate* 以每秒字节数为单位，可以包含变量。

0	禁用速率限制

备注

限制是针对每个请求设定的，因此如果 Angie 同时打开两个与 uwsgi 服务器的连接，则总体速率将是指定限制的两倍。限制仅在启用*buffering* 的情况下有效。

uwsgi_max_temp_file_size

语法	`uwsgi_max_temp_file_size` *size*;
默认值	`uwsgi_max_temp_file_size 1024m;`
上下文	http, server, location

当启用*buffering* 时，如果整个响应不适合由*uwsgi_buffer_size* 和*uwsgi_buffers* 指令设置的缓冲区，可以将部分响应保存到临时文件中。此指令设置临时文件的最大大小。写入临时文件的数据大小由*uwsgi_temp_file_write_size* 指令设置。

0	禁用将响应缓冲到临时文件

备注

此限制不适用于将被缓存或存储到磁盘 的响应。

uwsgi_modifier1

语法	`uwsgi_modifier1` *size*;
默认值	`uwsgi_modifier1 0;`
上下文	http, server, location

设置 uwsgi packet header 中 modifier1 字段的值。

uwsgi_modifier2

语法	`uwsgi_modifier2` *size*;
默认值	`uwsgi_modifier2 0;`
上下文	http, server, location

设置 uwsgi packet header 中 modifier2 字段的值。

uwsgi_next_upstream

语法	`uwsgi_next_upstream error | timeout | invalid_header | http_500 | http_503 | http_403 | http_404 | http_429 | non_idempotent | off` ...;
默认值	`uwsgi_next_upstream error timeout;`
上下文	http, server, location

指定在什么情况下请求应传递给*upstream pool* 中的下一个服务器：

error	在建立与服务器的连接、将请求传递给它或读取响应头时发生错误；
timeout	在建立与服务器的连接、将请求传递给它或读取响应头时发生超时；
invalid_header	服务器返回了空或无效的响应；
http_500	服务器返回了代码为 500 的响应；
http_503	服务器返回了代码为 503 的响应；
http_403	服务器返回了代码为 403 的响应；
http_404	服务器返回了代码为 404 的响应；
http_429	服务器返回了代码为 429 的响应；
non_idempotent	通常，带有 non-idempotent 方法的请求（POST、LOCK、PATCH）如果已经向上游服务器发送请求，则不会传递给下一个服务器；启用此选项显式允许重新尝试此类请求；
off	禁用将请求传递给下一个服务器。

备注

应该记住，只有在没有任何内容发送给客户端的情况下，才可以将请求传递给下一个服务器。也就是说，如果在响应传输的过程中发生错误或超时，无法修复这一点。

该指令还定义了什么被视为与服务器通信的不成功尝试。

error timeout invalid_header	始终被视为不成功的尝试，即使它们未在指令中指定
http_500 http_503 http_429	仅在指令中指定时被视为不成功的尝试
http_403 http_404	从不被视为不成功的尝试

将请求传递给下一个服务器的次数可以受到尝试次数 和时间 的限制。

uwsgi_next_upstream_timeout

语法	uwsgi_next_upstream_timeout *time*;
默认值	uwsgi_next_upstream_timeout 0;
上下文	http, server, location

限制在将请求传递给下一个 服务器时的时间。

0	关闭此限制

uwsgi_next_upstream_tries

语法	`uwsgi_next_upstream_tries` *number*;
默认值	`uwsgi_next_upstream_tries 0;`
上下文	http, server, location

限制将请求传递给下一个 服务器的可能尝试次数。

`0`	关闭此限制

uwsgi_no_cache

语法	`uwsgi_no_cache` *string* ...;
默认值	—
上下文	http, server, location

定义在什么条件下响应不会被保存到缓存中。如果字符串参数中的至少一个值不为空且不等于“0”，则响应将不会被保存：

```
uwsgi_no_cache $cookie_nocache $arg_nocache$arg_comment;
uwsgi_no_cache $http_pragma    $http_authorization;
```

可以与*uwsgi_cache_bypass* 指令一起使用。

uwsgi_param

语法	`uwsgi_param` *parameter value* [`if_not_empty`];
默认值	—
上下文	http, server, location

设置应传递给 uwsgi 服务器的参数。值可以包含文本、变量及其组合。如果当前级别没有定义 `uwsgi_param` 指令，则这些指令将从上一个配置级别继承。

标准 CGI 环境变量 应作为 uwsgi 头提供，请参见分发中提供的 `uwsgi_params` 文件：

```
location / {
    include uwsgi_params;
#    ...
}
```

如果指令使用了 `if_not_empty`，则只有在其值不为空时，该参数才会被传递给服务器：

```
uwsgi_param HTTPS $https if_not_empty;
```

uwsgi_pass

语法	uwsgi_pass [*protocol*://] *address*;
默认值	—
上下文	location, if in location

设置 uwsgi 服务器的协议和地址，以及可选的 URI，以便将位置映射。可以指定 uwsgi 或 suwsgi`（安全的:samp:`uwsgi，通过 SSL 的 uwsgi）作为协议。地址可以指定为域名或 IP 地址，以及端口：

```
uwsgi_pass localhost:9000;
uwsgi_pass uwsgi://localhost:9000;
uwsgi_pass suwsgi://[2001:db8::1]:9090;
```

或作为 UNIX 域套接字路径：

```
uwsgi_pass unix:/tmp/uwsgi.socket;
```

如果域名解析为多个地址，将以轮询方式使用它们。此外，地址可以作为*server group* 指定。如果使用了组，则不能与其指定端口；相反，必须单独为组内的每个服务器指定端口。

参数值可以包含变量。在这种情况下，如果地址指定为域名，将在描述的服务器组中搜索该名称，如果未找到，则使用*resolver* 确定。

uwsgi_pass_header

语法	uwsgi_pass_header *field* ...;
默认值	—
上下文	http, server, location

允许从 uwsgi 服务器向客户端传递否则禁用的 头字段。

uwsgi_pass_request_body

语法	uwsgi_pass_request_body on | off;
默认值	uwsgi_pass_request_body on;
上下文	http, server, location

指示是否将原始请求体传递给 uwsgi 服务器。另请参见*uwsgi_pass_request_headers* 指令。

uwsgi_pass_request_headers

语法	`uwsgi_pass_request_headers on	off;`
默认值	`uwsgi_pass_request_headers on;`
上下文	http, server, location

指示是否将原始请求的头字段传递给 uwsgi 服务器。另请参见*uwsgi_pass_request_body* 指令。

uwsgi_read_timeout

语法	`uwsgi_read_timeout` *time*;
默认值	`uwsgi_read_timeout 60s;`
上下文	http, server, location

定义从 uwsgi 服务器读取响应的超时时间。超时仅在两次连续读取操作之间设置，而不是整个响应的传输。如果 uwsgi 服务器在此时间内没有传输任何内容，则连接将被关闭。

uwsgi_request_buffering

语法	`uwsgi_request_buffering on	off;`
默认值	`uwsgi_request_buffering on;`
上下文	http, server, location

启用或禁用客户端请求体的缓冲。

`on`	在将请求发送到 uwsgi 服务器之前，从客户端读取 整个请求体。
`off`	请求体在接收后立即发送到 uwsgi 服务器。在这种情况下，如果 Angie 已经开始发送请求体，则请求无法传递给下一个服务器。

当使用 HTTP/1.1 分块传输编码发送原始请求体时，请求体将被缓冲，无论指令值如何。

uwsgi_send_timeout

语法	`uwsgi_send_timeout` *time*;
默认值	`uwsgi_send_timeout 60s;`
上下文	http, server, location

设置将请求传输到 uwsgi 服务器的超时时间。超时仅在两次连续写操作之间设置，而不是整个请求的传输。如果 uwsgi 服务器在此时间内没有接收到任何内容，则连接将被关闭。

uwsgi_socket_keepalive

语法	`uwsgi_socket_keepalive on	off;`
默认值	`uwsgi_socket_keepalive off;`
上下文	http, server, location

配置与 uwsgi 服务器的外发连接的"TCP keepalive" 行为。

`""`	默认情况下，操作系统的设置对套接字生效。
`on`	为套接字开启 *SO_KEEPALIVE* 套接字选项。

uwsgi_ssl_certificate

语法	`uwsgi_ssl_certificate` *file*;
默认值	—
上下文	http, server, location

指定用于对安全 uwsgi 服务器进行身份验证的 PEM 格式证书文件。文件名中可以使用变量。

uwsgi_ssl_certificate_cache

语法	`uwsgi_ssl_certificate_cache off;` `uwsgi_ssl_certificate_cache max=`*N* [`inactive=`*time*] [`valid=`*time*];
默认值	`uwsgi_ssl_certificate_cache off;`
上下文	http, server, location

定义一个用于存储通过变量指定的*SSL* 证书 和私钥 的缓存。

该指令支持以下参数：

- `max` —设置缓存中元素的最大数量。缓存溢出时，最少使用（LRU）的元素将被移除。
- `inactive` —指定在多长时间未访问后移除元素。默认值为 10 秒。
- `valid` —指定缓存元素被认为有效并可重复使用的时间。默认值为 60 秒。超过此时间后，证书将重新加载或重新验证。
- `off` —禁用缓存。

示例：

```
uwsgi_ssl_certificate       $uwsgi_ssl_server_name.crt;
uwsgi_ssl_certificate_key   $uwsgi_ssl_server_name.key;
uwsgi_ssl_certificate_cache max=1000 inactive=20s valid=1m;
```

uwsgi_ssl_certificate_key

语法	`uwsgi_ssl_certificate_key` *file*;
默认值	—
上下文	http, server, location

指定用于对安全 uwsgi 服务器进行身份验证的 PEM 格式密钥文件。

可以指定值"engine:*name*:id"，以从 OpenSSL 引擎名称加载具有指定 ID 的密钥。文件名中可以使用变量。

uwsgi_ssl_ciphers

语法	`uwsgi_ssl_ciphers` *ciphers*;
默认值	`uwsgi_ssl_ciphers DEFAULT;`
上下文	http, server, location

指定与安全 uwsgi 服务器请求的启用密码套件。密码套件的格式由 OpenSSL 库理解。

密码列表取决于已安装的 OpenSSL 版本。可以使用 `openssl ciphers` 命令查看完整列表。

注意

当使用 OpenSSL 时，`uwsgi_ssl_ciphers` 指令 * 不会 * 配置 TLS 1.3 的加密套件。如需配置 TLS 1.3 的加密套件，请使用*uwsgi_ssl_conf_command* 指令，它是为支持高级 SSL 配置而添加的。

- 在 LibreSSL 中，可以使用 `uwsgi_ssl_ciphers` 配置 TLS 1.3 加密套件。
- 在 BoringSSL 中，无法配置 TLS 1.3 加密套件。

uwsgi_ssl_conf_command

语法	`uwsgi_ssl_conf_command` *name value*;
默认值	—
上下文	http, server, location

在与安全 uwsgi 服务器建立连接时设置任意 OpenSSL 配置 commands。

重要

当使用 OpenSSL 1.0.2 或更高版本时支持该指令。要在 OpenSSL 中配置 TLS 1.3 密码套件，请使用 `ciphersuites` 命令。

可以在同一级别上指定多个 *uwsgi_ssl_conf_command* 指令。如果当前级别没有定义 *uwsgi_ssl_conf_command* 指令，则这些指令将继承自上一个配置级别。

小心

请注意，直接配置 OpenSSL 可能会导致意外行为。

uwsgi_ssl_crl

语法	`uwsgi_ssl_crl` *file*;
默认值	—
上下文	http, server, location

指定包含被吊销证书 (CRL) 的 PEM 格式文件，用于验证 安全 uwsgi 服务器的证书。

uwsgi_ssl_name

语法	`uwsgi_ssl_name` *name*;
默认值	`uwsgi_ssl_name` `` `host from uwsgi_pass;‘ ``
上下文	http, server, location

允许覆盖用于验证 安全 uwsgi 服务器证书的服务器名称，并在与安全 uwsgi 服务器建立连接时通过*SNI* 传递。

默认情况下，使用*uwsgi_pass* URL 的主机部分。

uwsgi_ssl_password_file

语法	`uwsgi_ssl_password_file` *file*;
默认值	—
上下文	http, server, location

指定包含密钥 的密码文件，每个密码单独一行。加载密钥时逐个尝试密码。

uwsgi_ssl_protocols

语法	`uwsgi_ssl_protocols` [SSLv2] [SSLv3] [TLSv1] [TLSv1.1] [TLSv1.2] [TLSv1.3];
默认值	`uwsgi_ssl_protocols TLSv1.2 TLSv1.3;`
上下文	http, server, location

在 1.2.0 版本发生变更: `TLSv1.3` 参数添加到默认集。

启用与安全 uwsgi 服务器请求的指定协议。

uwsgi_ssl_server_name

语法	`uwsgi_ssl_server_name on | off;`
默认值	`uwsgi_ssl_server_name off;`
上下文	http, server, location

启用或禁用通过*uwsgi_ssl_name* 指令设置的服务器名称在与安全 uwsgi 服务器建立连接时通过 服务器名称指示 TLS 扩展 (SNI, RFC 6066) 进行传递。

uwsgi_ssl_session_reuse

语法	`uwsgi_ssl_session_reuse on | off;`
默认值	`uwsgi_ssl_session_reuse on;`
上下文	http, server, location

确定在与 uwsgi 服务器工作时是否可以重用 SSL 会话。如果日志中出现错误"*SSL3_GET_FINISHED:digest check failed*"，请尝试禁用会话重用。

uwsgi_ssl_trusted_certificate

语法	`uwsgi_ssl_trusted_certificate` *file*;
默认值	—
上下文	http, server, location

指定包含受信任 CA 证书的 PEM 格式文件，用于验证 安全 uwsgi 服务器的证书。

uwsgi_ssl_verify

语法	`uwsgi_ssl_verify on | off;`
默认值	`uwsgi_ssl_verify off;`
上下文	http, server, location

启用或禁用对安全 uwsgi 服务器证书的验证。

uwsgi_ssl_verify_depth

语法	`uwsgi_ssl_verify_depth` *number*;
默认值	`uwsgi_ssl_verify_depth 1;`
上下文	http, server, location

设置安全 uwsgi 服务器证书链中的验证深度。

uwsgi_store

语法	`uwsgi_store on | off | string;`
默认值	`uwsgi_store off;`
上下文	http, server, location

启用保存文件到磁盘。

`on`	保存的文件路径对应于指令*alias* 或*root*
`off`	禁用文件保存

可以使用带有变量的 `string` 明确设置文件名：

```
uwsgi_store /data/www$original_uri;
```

文件的修改时间是根据接收到的"Last-Modified" 响应头字段设置的。响应首先写入一个临时文件，然后该文件被重命名。临时文件和持久存储可以放在不同的文件系统上。然而，请注意，在这种情况下，文件是在两个文件系统之间复制，而不是进行廉价的重命名操作。因此，建议对于任何给定的 *location*，保存的文件和由*uwsgi_temp_path* 指令设置的临时文件目录放在同一个文件系统上。

此指令可用于创建静态不可更改文件的本地副本，例如：

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

location /fetch/ {
    internal;

    uwsgi_pass         backend:9000;
    ...

    uwsgi_store        on;
    uwsgi_store_access user:rw group:rw all:r;
    uwsgi_temp_path    /data/temp;

    alias              /data/www/;
}
```

uwsgi_store_access

语法	`uwsgi_store_access` *users:permissions ...*;
默认	`uwsgi_store_access user:rw;`
上下文	http, server, location

设置新创建的文件和目录的访问权限，例如：

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

如果指定了任何 `group` 或 `all` 访问权限，则可以省略用户权限：

```
uwsgi_store_access group:rw all:r;
```

uwsgi_temp_file_write_size

语法	`uwsgi_temp_file_write_size` *size*;
默认	`uwsgi_temp_file_write_size 8k|16k;`
上下文	http, server, location

限制在启用从 uwsgi 服务器到临时文件的响应缓冲时，一次写入临时文件的数据大小。默认情况下，大小受*uwsgi_buffer_size* 和*uwsgi_buffers* 指令设置的两个缓冲区的限制。临时文件的最大大小由*uwsgi_max_temp_file_size* 指令设置。

uwsgi_temp_path

语法	`uwsgi_temp_path` *path* [*level1* [*level2* [*level3*]]]‘;
默认	`uwsgi_temp_path uwsgi_temp;` (路径取决于 `--http-uwsgi-temp-path` 构建选项)
上下文	http, server, location

定义一个目录用于存储从 uwsgi 服务器接收到的临时文件。可以在指定目录下使用最多三层的子目录层次结构。例如，在以下配置中

```
uwsgi_temp_path /spool/angie/uwsgi_temp 1 2;
```

一个临时文件可能看起来像这样：

```
/spool/angie/uwsgi_temp/7/45/00000123457
```

另见*uwsgi_cache_path* 指令的 *use_temp_path* 参数。

HTTP/2

提供对 HTTP/2 的支持。

当 从源代码构建时，此模块默认不构建；应通过 `--with-http_v2_module` 构建选项启用。

在来自 我们仓库的包和镜像中，该模块包含在构建中。

配置示例

```
server {
    listen 443 ssl;

    http2 on;

    ssl_certificate server.crt;
    ssl_certificate_key server.key;
}
```

重要

请注意，通过 TLS 接受 HTTP/2 连接需要” 应用层协议协商”(ALPN) TLS 扩展支持，该支持自 OpenSSL 1.0.2 版本起可用。

如果*ssl_prefer_server_ciphers* 指令设置为”on”，则*ciphers* 应配置为符合 RFC 9113, 附录 A 黑名单并被客户端支持。

指令

http2

Added in version 1.2.0.

语法	`http2 on	off;`
默认	`http2 off;`
上下文	http, server

启用 HTTP/2 协议。

http2_body_preread_size

语法	`http2_body_preread_size` *size*;
默认	—
上下文	http, server

设置每个请求的缓冲区大小，在请求体开始处理前可以存储请求体。

http2_chunk_size

语法	`http2_chunk_size` *size*;
默认	`http2_chunk_size 8k;`
上下文	http, server, location

设置响应体切片的最大块大小。过低的值会导致更高的开销。过高的值会因 HOL 阻塞 而降低优先级。

http2_max_concurrent_pushes

自 1.2.0 版本弃用.

语法	`http2_max_concurrent_pushes` *number*;
默认	`http2_max_concurrent_pushes 10;`
上下文	http, server

限制连接中并发推送 请求的最大数量。

http2_max_concurrent_streams

语法	`http2_max_concurrent_streams` *number*;
默认	`http2_max_concurrent_streams 128;`
上下文	http, server

设置连接中并发 HTTP/2 流的最大数量。

http2_push

自 1.2.0 版本弃用.

语法	`http2_push` *uri* | `off`;
默认	`http2_push off;`
上下文	http, server, location

主动发送 (推送) 请求到指定的 uri，并与原始请求的响应一起发送。只有具有绝对路径的相对 URI 将被处理，例如：

```
http2_push /static/css/main.css;
```

`uri` 值可以包含变量。

在同一配置级别可以指定多个 *http2_push* 指令。`off` 参数取消从上一级配置继承的 *http2_push* 指令的效果。

http2_push_preload

自 1.2.0 版本弃用.

语法	`http2_push_preload on | off;`
默认	`http2_push_preload off;`
上下文	http, server, location

启用在"Link" 响应头字段中指定的 预加载链接 自动转换为 推送 请求。

http2_recv_buffer_size

语法	`http2_recv_buffer_size` *size*;
默认	`http2_recv_buffer_size 256k;`
上下文	http

设置每个*worker* 输入缓冲区的大小。

内置变量

http_v2 模块支持以下内置变量：

$http2

协商的协议标识符：

h2	用于 TLS 上的 HTTP/2
h2c	用于明文 TCP 上的 HTTP/2
""	否则为空字符串

HTTP/3

提供 HTTP/3 协议对客户端连接的支持，以及对使用以下*Proxy* 模块指令配置的代理服务器连接的支持：

- *proxy_http3_hq*
- *proxy_http3_max_concurrent_streams*
- *proxy_http3_max_table_capacity*
- *proxy_http3_stream_buffer_size*
- *proxy_http_version*
- *proxy_pass*
- *proxy_quic_active_connection_id_limit*

- *proxy_quic_gso*
- *proxy_quic_host_key*

当 从源代码构建时，此模块默认不构建；应通过 `--with-http_v3_module` 构建选项启用。

在来自 我们仓库的软件包和镜像中，该模块已包含在构建中。

配置示例

```
http {
    log_format quic '$remote_addr - $remote_user [$time_local] '
                    '"$request" $status $body_bytes_sent '
                    '"$http_referer" "$http_user_agent" "$http3"';

    access_log logs/access.log quic;

    server {
        # 为了更好的兼容性，建议
        # 对 http/3 和 https 使用相同的端口
        listen 8443 quic reuseport;
        listen 8443 ssl;

        ssl_certificate     certs/example.com.crt;
        ssl_certificate_key certs/example.com.key;

        location / {
            # 用于宣传 HTTP/3 的可用性
            add_header Alt-Svc 'h3=":8443"; ma=86400';
        }
    }
}
```

重要

请注意，接受通过 TLS 的 HTTP/3 连接需要 TLSv1.3 协议支持，该支持自 OpenSSL 版本 1.1.1 起可用。

指令

http3

语法	`http3 on | off;`
默认	`http3 on;`
上下文	http, server

启用 HTTP/3 协议协商。

http3_hq

语法	`http3_hq on | off;`
默认	`http3_hq off;`
上下文	http, server

启用在 QUIC 互操作测试 中使用的 HTTP/0.9 协议协商。

注意

仅在运行明确需要此模式的专用测试时启用此模式。

http3_max_concurrent_streams

语法	`http3_max_concurrent_streams` *number*;
默认	`http3_max_concurrent_streams 128;`
上下文	http, server

初始化 HTTP/3 和 QUIC 设置，并设置连接中并发 HTTP/3 请求流的最大数量。

http3_max_table_capacity

语法	`http3_max_table_capacity` *number*;
默认	`http3_max_table_capacity 4096;`
上下文	http, server

设置服务器连接的 动态表 容量。

备注

类似的*proxy_http3_max_table_capacity* 指令对代理连接执行此操作。为避免错误，当启用带缓存的代理时，禁用动态表的使用。

http3_stream_buffer_size

语法	`http3_stream_buffer_size` *size*;
默认	`http3_stream_buffer_size 64k;`
上下文	http, server

设置用于读取和写入 QUIC 流的缓冲区大小。

quic_active_connection_id_limit

语法	`quic_active_connection_id_limit` *number*;
默认	`quic_active_connection_id_limit 2;`
上下文	http, server

设置 QUIC *active_connection_id_limit* 传输参数的值。这是可以存储在服务器上的连接 ID 的最大数量。

quic_bpf

语法	`quic_bpf on	off;`
默认	`quic_bpf off;`
上下文	main

启用使用 eBPF 路由 QUIC 数据包。当启用时，这允许支持 QUIC 连接迁移。

重要

该指令仅在 Linux 5.7+ 上受支持。

quic_gso

语法	`quic_gso on	off;`
默认	`quic_gso off;`
上下文	http, server

启用使用分段卸载的优化批量模式发送。

重要

优化发送仅在具有 *UDP_SEGMENT* 功能的 Linux 上受支持。

quic_host_key

语法	`quic_host_key` *file*;
默认	—
上下文	http, server

设置用于加密无状态重置和地址验证令牌的秘密密钥的 *file*。默认情况下，每次重载时生成一个随机密钥。使用旧密钥生成的令牌不被接受。

quic_retry

语法	`quic_retry on	off;`
默认	`quic_retry off;`
上下文	http, server

启用 QUIC 地址验证 功能。这包括在 *Retry* 数据包或 *NEW_TOKEN* 帧中发送新令牌，并验证在 *Initial* 数据包中接收到的令牌。

内置变量

http_v3 模块支持以下内置变量：

$http3

协商的协议标识符：

h3	用于 HTTP/3 连接
hq	用于 hq 连接
""	否则为空字符串

$quic_connection

QUIC 连接序列号

XSLT

该模块是一个过滤器，使用一个或多个 XSLT 样式表转换 XML 响应。

当 从源代码构建时，默认不构建此模块；应使用 `--with-http_xslt_module` 构建选项启用它。

在我们的代码库中，该模块是 动态构建的，并作为一个名为 `angie-module-xslt` 或 `angie-pro-module-xslt` 的独立包提供。

重要

该模块需要 libxml2 和 libxslt 库。

配置示例

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

指令

xml_entities

语法	`xml_entities` *path*;
默认值	—
上下文	http, server, location

指定声明字符实体的 DTD 文件。该文件在配置阶段编译。出于技术原因，该模块无法使用在处理的 XML 中声明的外部子集，因此被忽略，并使用专门定义的文件。该文件不应描述 XML 结构。只需声明所需的字符实体，例如：

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

xslt_last_modified

语法	`xslt_last_modified on | off;`
默认值	`xslt_last_modified off;`
上下文	http, server, location

在 XSLT 转换期间允许保留原始响应中的”Last-Modified” 头字段，以便于响应缓存。

默认情况下，头字段被移除，因为响应的内容在转换期间被修改，可能包含动态生成的元素或与原始响应独立更改的部分。

xslt_param

语法	`xslt_param` *parameter value*;
默认值	—
上下文	http, server, location

定义 XSLT 样式表的参数。值被视为 XPath 表达式。值可以包含变量。要将字符串值传递给样式表，可以使用*xslt_string_param* 指令。

可以有多个 *xslt_param* 指令。如果当前级别没有定义 *xslt_param* 和*xslt_string_param* 指令，则这些指令将从上一个配置级别继承。

xslt_string_param

语法	`xslt_string_param` *parameter value*;
默认值	—
上下文	http, server, location

定义 XSLT 样式表的字符串参数。值中的 XPath 表达式不被解释。值可以包含变量。

可以有多个 *xslt_string_param* 指令。如果当前级别没有定义*xslt_param* 和 *xslt_string_param* 指令，则这些指令将从上一个配置级别继承。

xslt_stylesheet

语法	`xslt_stylesheet` *stylesheet* [*parameter=value* ...];
默认值	—
上下文	location

定义 XSLT 样式表及其可选参数。样式表在配置阶段编译。

参数可以单独指定，也可以使用":" 分隔符在一行中分组。如果参数包含":" 字符，应转义为"%3A"。此外，libxslt 要求将包含非字母数字字符的参数用单引号或双引号括起来，例如：

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

参数描述可以包含变量，例如，整个参数行可以从一个变量中取得：

```
location / {
    xslt_stylesheet /site/xslt/one.xslt
                    $arg_xslt_params
                    param1='$value1':param2=value2
                    param3=value3;
}
```

可以指定多个样式表。它们将按指定的顺序依次应用。

xslt_types

语法	`xslt_types` mime-*type* ...;
默认值	`xslt_types text/xml;`
上下文	http, server, location

启用对指定 MIME 类型的响应进行转换，除了 `text/xml` 之外。特殊值"*" 匹配任何 MIME 类型。如果转换结果是 HTML 响应，则其 MIME 类型更改为 `text/html`。

核心 HTTP 模块实现了 HTTP 服务器的基本功能：包括定义服务器块、配置请求路由的位置、提供静态文件和控制访问、配置重定向、支持 keep-alive 连接以及管理请求和响应头。

本节中的其他模块扩展了此功能，使您能够灵活配置和优化 HTTP 服务器以满足各种场景和要求。

指令

absolute_redirect

语法	`absolute_redirect on	off;`
默认	`absolute_redirect on;`
上下文	http, server, location

如果禁用，Angie 发出的重定向将是相对的。

另请参阅*server_name_in_redirect* 和*port_in_redirect* 指令。

aio

语法	`aio on	off	threads` [=*pool*];
默认	`aio off;`
上下文	http, server, location

启用或禁用在 FreeBSD 和 Linux 上使用异步文件 I/O（AIO）：

```
location /video/ {
  aio            on;
  output_buffers 1 64k;
}
```

在 FreeBSD 上，AIO 可以从 FreeBSD 4.3 开始使用。在 FreeBSD 11.0 之前，AIO 可以静态链接到内核：

```
options VFS_AIO
```

或者作为可加载内核模块动态加载：

```
kldload aio
```

在 Linux 上，AIO 可以从内核版本 2.6.22 开始使用。此外，必须启用*directio*，否则读取将是阻塞的：

```
location /video/ {
  aio            on;
  directio       512;
  output_buffers 1 128k;
}
```

在 Linux 上，*directio* 只能用于读取在 512 字节边界（或 XFS 的 4K）对齐的块。文件的未对齐结尾在阻塞模式下读取。对字节范围请求和从文件开头以外的位置的 FLV 请求也是如此：在文件的开头和结尾读取未对齐数据将是阻塞的。

当在 Linux 上启用 AIO 和*sendfile* 时，AIO 用于大于或等于*directio* 指令指定的大小的文件，而*sendfile* 用于较小的文件或当*directio* 被禁用时：

```
location /video/ {
  sendfile       on;
  aio            on;
  directio       8m;
}
```

最后，文件可以使用多线程读取和发送，而不会阻塞工作进程：

```
location /video/ {
  sendfile       on;
  aio            threads;
}
```

读取和发送文件操作被卸载到指定的*pool* 的线程中。如果省略池名称，则使用名为"default" 的池。池名称也可以通过变量设置：

```
aio threads=pool$disk;
```

默认情况下，多线程被禁用，必须通过 *--with-threads* 配置参数启用。目前，多线程仅与*epoll*、*kqueue* 和*eventport* 方法兼容。多线程发送文件仅在 Linux 上支持。

另请参阅*sendfile* 指令。

aio_write

语法	`aio_write on | off;`
默认	`aio_write off;`
上下文	http, server, location

如果启用了*aio*，则指定其是否用于写入文件。目前，这仅在使用 `aio threads` 时有效，并且仅限于写入从代理服务器接收到的临时文件。

alias

语法	`alias` *path*;
默认	—
上下文	location

定义指定位置的替代路径。例如，以下配置：

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

在请求 `/i/top.gif` 时，将发送文件 */data/w3/images/top.gif*。

path 值可以包含变量，除了*$document_root* 和*$realpath_root*。

如果在使用正则表达式定义的 location 内使用 `alias`，则该正则表达式应包含捕获，并且 `alias` 应引用这些捕获，例如：

```
location ~ ^/users/(.+\.(?:gif|jpe?g|png))$ {
  alias /data/w3/images/$1;
}
```

当 location 匹配指令值的最后一部分时：

```
location /images/ {
  alias /data/w3/images/;
}
```

最好使用*root* 指令：

```
location /images/ {
  root /data/w3;
}
```

auth_delay

语法	`auth_delay` *time*;
默认	`auth_delay 0s;`
上下文	http, server, location

延迟处理未经授权的请求，并返回 401 响应代码，以防止在通过密码 或子请求结果 限制访问时发生时间攻击。

auto_redirect

语法	`auto_redirect` [`on` | `off` | `default`];
默认	`auto_redirect default;`
上下文	http, server, location

控制当前缀 location 以斜杠结尾时的重定向 行为：

```
location /prefix/ {
    auto_redirect on;
}
```

在这里，请求 /prefix 将导致重定向到 /prefix/。

值 on 明确启用重定向，而 off 禁用它。当设置为 default 时，仅在 location 处理带有*api*、*proxy_pass*、*fastcgi_pass*、*uwsgi_pass*、*scgi_pass*、*memcached_pass* 或*grpc_pass* 的请求时，才启用重定向。

chunked_transfer_encoding

语法	chunked_transfer_encoding on | off;
默认	chunked_transfer_encoding on;
上下文	http, server, location

允许在 HTTP/1.1 中禁用分块传输编码。当使用不支持分块编码的软件时，这可能会很有用，尽管标准要求如此。

client

语法	client { ... }
默认	—
上下文	http

为命名 location 创建 client 上下文，这些 location 用于处理 Angie 自身创建的 * 出站 *HTTP 请求，无需真实客户端参与。

在此上下文中定义的 location 只能以以下方式使用：

- 通过预定义的 location @acme 向*ACME* 模块中的 ACME 目录发出请求，可以使用*Proxy* 模块指令进行额外配置；
- 通过预定义的 location @docker_events 和 @docker_containers 向*Docker* 模块中的 Docker API 发出事件和容器请求，可以使用*Proxy* 模块指令进行额外配置；
- 在*Upstream* 模块中使用带有 remote_action 的*sticky learn* 模式。

配置中只能存在一个 client 上下文。

client_body_buffer_size

语法	client_body_buffer_size *size*;
默认	client_body_buffer_size 8k|16k;
上下文	http, server, location

设置用于读取客户端请求体的缓冲区大小。如果请求体大于缓冲区，则整个请求体或仅其部分将写入临时文件。默认情况下，缓冲区大小等于两个内存页面。在 x86、其他 32 位平台和 x86-64 上为 8K。在其他 64 位平台上通常为 16K。

client_body_in_file_only

语法	`client_body_in_file_only on | clean | off;`
默认	`client_body_in_file_only off;`
上下文	http, server, location

确定 Angie 是否应将整个客户端请求体保存到文件中。此指令可以在调试期间使用，或在使用*$request_body_file* 变量或*Perl* 模块的*$r->request_body_file* 方法时使用。

`on`	请求处理后不会删除临时文件
`clean`	将导致请求处理后删除剩余的临时文件

client_body_in_single_buffer

语法	`client_body_in_single_buffer on | off;`
默认	`client_body_in_single_buffer off;`
上下文	http, server, location

确定 Angie 是否应将整个客户端请求体保存在单个缓冲区中。此指令在使用*$request_body* 变量时推荐使用，以减少涉及的复制操作数量。

client_body_temp_path

语法	`client_body_temp_path` *path* [*level1* [*level2* [*level3*]]];
默认	`client_body_temp_path client_body_temp;` (路径取决于 `--http-client-body-temp-path` 构建选项)
上下文	http, server, location

定义用于存储客户端请求体的临时文件的目录。在指定目录下最多可以使用三级子目录层次。例如，在以下配置中

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

临时文件的路径可能如下所示：

```
/spool/angie/client_temp/7/45/00000123457
```

client_body_timeout

语法	`client_body_timeout` *time*;
默认	`client_body_timeout 60s;`
上下文	http, server, location

定义读取客户端请求体的超时时间。超时仅针对两次连续读取操作之间的时间段设置，而不是针对整个请求体的传输。如果客户端在此时间内未传输任何内容，请求将以 408（请求超时）错误终止。

client_header_buffer_size

语法	`client_header_buffer_size` *size*;
默认	`client_header_buffer_size 1k;`
上下文	http, server

设置用于读取客户端请求头的缓冲区大小。对于大多数请求，1K 字节的缓冲区就足够了。然而，如果请求包含长 cookie，或者来自 WAP 客户端，则可能无法适应 1K。如果请求行或请求头字段无法放入此缓冲区，则会分配通过*large_client_header_buffers* 指令配置的更大缓冲区。

如果该指令在*server* 级别上指定，则可以使用默认服务器的值。详细信息请参见虚拟服务器选择 部分。

client_header_timeout

语法	`client_header_timeout` *time*;
默认	`client_header_timeout 60s;`
上下文	http, server

定义读取客户端请求头的超时。如果客户端在此时间内未传输整个头，则请求将以 408（请求超时）错误终止。

client_max_body_size

语法	`client_max_body_size` *size*;
默认	`client_max_body_size 1m;`
上下文	http, server, location

设置客户端请求体的最大允许大小。如果请求中的大小超过配置值，则会返回 413（请求实体太大）错误给客户端。请注意，浏览器无法正确显示此错误。

0	禁用对客户端请求体大小的检查

connection_pool_size

语法	`connection_pool_size` *size*;
默认	`connection_pool_size 256	512;`
上下文	http, server, location

允许准确调整每个连接的内存分配。该指令对性能的影响最小，通常不应使用。默认情况下：

256 (字节)	32 位平台
512 (字节)	64 位平台

default_type

语法	`default_type` mime-*type*;
默认	`default_type text/plain;`
上下文	http, server, location

定义响应的默认 MIME 类型。文件名扩展名到 MIME 类型的映射可以通过*types* 指令设置。

directio

语法	`directio` *size*	`off`;
默认	`directio off;`
上下文	http, server, location

启用在读取大于或等于指定大小的文件时使用 `O_DIRECT` 标志（FreeBSD, Linux)、`F_NOCACHE` 标志（macOS）或 `directio()` 函数（Solaris)。该指令会自动禁用对特定请求的*sendfile* 使用。它在提供大文件时非常有用：

```
directio 4m;
```

或在 Linux 上使用*aio* 时。

directio_alignment

语法	`directio_alignment` *size*;
默认	`directio_alignment 512;`
上下文	http, server, location

为*directio* 设置对齐。在大多数情况下，512 字节的对齐就足够了。然而，在 Linux 上使用 XFS 时，需要增加到 4K。

disable_symlinks

语法	`disable_symlinks off;` `disable_symlinks on | if_not_owner` [`from=`*part*];
默认	`disable_symlinks off;`
上下文	http, server, location

确定在打开文件时如何处理符号链接：

`off`	允许并不检查路径名中的符号链接。这是默认行为。
`on`	如果路径名的任何组成部分是符号链接，则拒绝访问文件。
`if_not_owner`	如果路径名的任何组成部分是符号链接，并且链接和链接指向的对象具有不同的所有者，则拒绝访问文件。
`from=`*part*	在检查符号链接时（参数 `on` 和 `if_not_owner`），通常检查路径名的所有组成部分。通过额外指定 `from=part` 参数，可以避免检查路径名初始部分中的符号链接。在这种情况下，仅从指定初始部分之后的路径名组成部分检查符号链接。如果该值不是被检查的路径名的初始部分，则整个路径名将像未指定此参数一样进行检查。如果该值与整个文件名匹配，则不检查符号链接。参数值可以包含变量。

示例：

```
disable_symlinks on from=$document_root;
```

该指令仅在具有 `openat()` 和 `fstatat()` 接口的系统上可用。这些系统包括现代版本的 FreeBSD、Linux 和 Solaris。

警告

参数 `on` 和 `if_not_owner` 增加了处理开销。

在不支持仅为搜索打开目录的系统上，使用这些参数需要工作进程对所有被检查的目录具有读取权限。

备注

AutoIndex、*Random Index* 和*DAV* 模块当前忽略此指令。

error_page

语法	`error_page` *code ...* [`=`[*response*]] *uri*;
默认	—
上下文	http, server, location, if in location

定义在指定错误时将显示的 URI。*uri* 值可以包含变量。

示例：

```
error_page 404             /404.html;
error_page 500 502 503 504 /50x.html;
```

这会导致内部重定向到指定的 *uri*，并将客户端请求方法更改为"GET"（对于除"GET" 和"HEAD" 之外的所有方法）。

另外，可以使用 `=response` 语法将响应代码更改为另一个，例如：

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

如果错误响应由代理服务器或 FastCGI/uwsgi/SCGI/gRPC 服务器处理，并且该服务器可能返回不同的响应代码（例如，200、302、401 或 404），则可以响应其返回的代码：

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

如果在内部重定向期间不需要更改 URI 和方法，可以将错误处理传递到命名位置：

```
location / {
  error_page 404 = @fallback;
}

location @fallback {
  proxy_pass http://backend;
}
```

备注

如果 uri 处理导致错误，则返回给客户端的状态代码为最后发生错误的状态代码。

还可以使用 URL 重定向进行错误处理：

```
error_page 403      http://example.com/forbidden.html;
error_page 404 =301 http://example.com/notfound.html;
```

在这种情况下，默认情况下，响应代码 302 将返回给客户端。它只能更改为重定向状态代码之一（301、302、303、307 和 308）。

etag

语法	`etag on | off;`
默认	`etag on;`
上下文	http, server, location

启用或禁用静态资源的"ETag" 响应头字段的自动生成。

http

语法	`http { ... }`
默认	—
上下文	main

提供 HTTP 服务器指令的配置文件上下文。

if_modified_since

语法	`if_modified_since off | exact | before;`
默认	`if_modified_since exact;`
上下文	http, server, location

指定如何将响应的修改时间与"If-Modified-Since" 请求头字段中的时间进行比较：

`off`	响应始终被视为已修改
`exact`	精确匹配
`before`	响应的修改时间小于或等于"If-Modified-Since" 请求头字段中的时间

ignore_invalid_headers

语法	`ignore_invalid_headers on | off;`
默认	`ignore_invalid_headers on;`
上下文	http, server

控制是否忽略无效名称的头字段。有效名称由英文字母、数字、连字符和可能的下划线（由*underscores_in_headers* 指令控制）组成。

如果该指令在 *server* 级别上指定，则可以使用默认服务器的值。

internal

语法	`internal;`
默认	—
上下文	location

指定给定位置只能用于内部请求。对于外部请求，将返回客户端错误 404（未找到）。内部请求包括：

- 由*error_page*、*index*、*random_index* 和*try_files* 指令重定向的请求；

- 由上游服务器的 *X-Accel-Redirect* 响应头字段重定向的请求；
- 由*http_ssi* 模块的 *include virtual* 命令、*http_addition* 模块指令及*auth_request* 和*mirror* 指令形成的子请求；
- 由*rewrite* 指令更改的请求。

示例：

```
error_page 404 /404.html;

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

备注

每个请求的内部重定向限制为 10 次，以防止在不正确的配置中发生请求处理循环。如果达到该限制，将返回错误 500（内部服务器错误）。在这种情况下，可以在错误日志中看到 *rewrite or internal redirection cycle* 消息。

keepalive_disable

语法	`keepalive_disable none` | *browser* ...;
默认值	`keepalive_disable msie6;`
上下文	http, server, location

禁用与表现不佳的浏览器的长连接。*browser* 参数指定哪些浏览器将受到影响。

`none`	启用与所有浏览器的长连接
`msie6`	禁用与旧版本 MSIE 的长连接，一旦收到 POST 请求
`safari`	禁用与 macOS 及类似操作系统上的 Safari 和类似浏览器的长连接

keepalive_requests

语法	`keepalive_requests` *number*;
默认值	`keepalive_requests 1000;`
上下文	http, server, location

设置通过一个长连接可以服务的最大请求数。在达到最大请求数后，连接将关闭。

定期关闭连接是必要的，以释放每个连接的内存分配。因此，使用过高的最大请求数可能导致过高的内存使用，不建议这样做。

keepalive_time

语法	`keepalive_time` *time*;
默认值	`keepalive_time 1h;`
上下文	http, server, location

限制通过一个长连接处理请求的最长时间。在达到此时间后，连接将在后续请求处理后关闭。

keepalive_timeout

语法	`keepalive_timeout` *timeout* [*header_timeout*];
默认值	`keepalive_timeout 75s;`
上下文	http, server, location

timeout	设置长连接客户端连接在服务器端保持开启的超时时间
0	禁用长连接客户端连接

可选的第二个参数设置”Keep-Alive: timeout=time” 响应头字段中的值。两个参数可能不同。

”Keep-Alive: timeout=time” 头字段被 Mozilla 和 Konqueror 识别。MSIE 大约在 60 秒后自动关闭长连接。

large_client_header_buffers

语法	`large_client_header_buffers` *number size*;
默认值	`large_client_header_buffers 4 8k;`
上下文	http, server

设置用于读取大客户端请求头的最大缓冲区数量和大小。请求行不能超过一个缓冲区的大小，否则将返回 414（请求的 URI 过大）错误给客户端。请求头字段也不能超过一个缓冲区的大小，否则将返回 400（错误的请求）错误给客户端。缓冲区仅在需要时分配。默认情况下，缓冲区大小为 8K 字节。如果在请求处理结束后，连接被转换为长连接状态，则这些缓冲区将被释放。

如果指令在*server* 级别上指定，则可以使用默认服务器的值。

limit_except

语法	`limit_except` *method1* [*method2...*] { ... };
默认值	—
上下文	location

限制在某个位置内允许的 HTTP 方法。*method* 参数可以是以下之一：GET、HEAD、POST、PUT、DELETE、MKCOL、COPY、MOVE、OPTIONS、PROPFIND、PROPPATCH、LOCK、UNLOCK 或 PATCH。允许 GET 方法也会启用 HEAD 方法。可以使用*Access* 和*Auth Basic* 模块中的指令限制对其他方法的访问：

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

备注

此示例中的限制适用于所有方法，**除了** GET 和 HEAD。

limit_rate

语法	limit_rate *rate*;
默认值	limit_rate 0;
上下文	http, server, location, if in location

限制向客户端的响应传输速率。速率以每秒字节为单位指定。零值禁用速率限制。限制是针对每个请求设置的，因此如果客户端同时打开两个连接，则整体速率将是指定限制的两倍。

参数值可以包含变量。在某些情况下，可能需要根据特定条件限制速率：

```
map $slow $rate {
  1     4k;
  2     8k;
}

limit_rate $rate;
```

速率限制也可以在*$limit_rate* 变量中设置，但不推荐使用此方法：

```
server {

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

}
```

速率限制也可以在代理服务器响应的”X-Accel-Limit-Rate”头字段中设置。可以使用*proxy_ignore_headers*、*fastcgi_ignore_headers*、*uwsgi_ignore_headers* 和*scgi_ignore_headers* 指令禁用此功能。

limit_rate_after

语法	`limit_rate_after` *size*;
默认值	`limit_rate_after 0;`
上下文	http, server, location, if in location

设置在继续向客户端传输响应之前的初始量。参数值可以包含变量。

示例：

```
location /flv/ {
 flv;
 limit_rate_after 500k;
 limit_rate       50k;
}
```

lingering_close

语法	`lingering_close on | always | off;`
默认值	`lingering_close on;`
上下文	http, server, location

控制 Angie 如何关闭客户端连接。

`on`	指示 Angie 在完全关闭连接之前：等待 并处理 来自客户端的额外数据，但仅在启发式分析表明客户端可能还在发送更多数据的情况下。
`always`	将导致 Angie 无条件等待并处理额外的客户端数据。
`off`	告诉 Angie 永远不要等待更多数据并立即关闭连接。这种行为破坏了协议，正常情况下不应使用。

要控制 HTTP/2 连接的关闭，指令必须在*server* 级别上指定。

lingering_time

语法	`lingering_time` *time*;
默认值	`lingering_time 30s;`
上下文	http, server, location

当*lingering_close* 生效时，此指令指定 Angie 将处理（读取并忽略）来自客户端的额外数据的最长时间。在此之后，即使还有更多数据，连接也将被关闭。

lingering_timeout

语法	`lingering_timeout` *time*;
默认值	`lingering_timeout 5s;`
上下文	http, server, location

当*lingering_close* 生效时，此指令指定等待更多客户端数据到达的最长等待时间。如果在此时间内未收到数据，则连接将关闭。否则，数据将被读取并忽略，Angie 再次开始等待更多数据。" 等待-读取-忽略" 循环会重复，但不会超过*lingering_time* 指令指定的时间。

在优雅关闭期间，客户端长连接仅在至少处于非活动状态 `lingering_timeout` 指定的时间后才会关闭。

listen

语法	`listen` *address*[:*port*] [`default_server`] [`ssl`] [http2 | `quic`] [`proxy_protocol`] [`setfib=`*number*] [`fastopen=`*number*] [`backlog=`*number*] [`rcvbuf=`*size*] [`sndbuf=`*size*] [`accept_filter=`*filter*] [`deferred`] [`bind`] [`ipv6only=on` | `off`] [`reuseport`] [`so_keepalive=`on|off|[`keepidle`]:[samp:*keepintvl*]:[samp:*keepcnt*]]; `listen` *port* [`default_server`] [`ssl`] [http2 | `quic`] [`proxy_protocol`] [`setfib=`*number*] [`fastopen=`*number*] [`backlog=`*number*] [`rcvbuf=`*size*] [`sndbuf=`*size*] [`accept_filter=`*filter*] [`deferred`] [`bind`] [`ipv6only=on` | `off`] [`reuseport`] [`so_keepalive=`on|off|[`keepidle`]:[samp:*keepintvl*]:[samp:*keepcnt*]]; `listen` unix:*path* [`default_server`] [`ssl`] [http2 | `quic`] [`proxy_protocol`] [`backlog=`*number*] [`rcvbuf=`*size*] [`sndbuf=`*size*] [`accept_filter=`*filter*] [`deferred`] [`bind`] [`so_keepalive=`on|off|[*keepidle*]:[*keepintvl*]:[*keepcnt*]];
默认值	`listen *:80 | *:8000;`
上下文	server

设置服务器将接受请求的监听套接字的 `address` 和 `port`，或 UNIX 域套接字的路径。`address` 也可以是主机名，例如：

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

IPv6 地址在方括号中指定：

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

UNIX 域套接字使用 `unix:` 前缀指定：

```
listen unix:/var/run/angie.sock;
```

可以同时指定 address 和 port，或者只指定 address 或 port。当省略某些部分时，行为会有所不同：

- 如果只给出 address，则使用端口 80。
- 如果只给出 port，Angie 会在所有可用的 IPv4（如果启用，还包括 IPv6）接口上监听。对于该端口，第一个定义的 server 块将成为具有不匹配 Host 头的请求的默认服务器。
- 如果完全省略该指令，Angie 在以超级用户权限运行时使用 *:80，否则使用 *:8000。

default_server	指定此参数的服务器将成为给定 address:port 对的默认服务器（两者共同形成一个 监听套接字）。 如果没有带有 default_server 参数的指令，监听套接字的默认服务器将是配置中为该套接字服务的第一个服务器。
ssl	允许指定在此端口上接受的所有连接应以 SSL 模式工作。这允许处理 HTTP 和 HTTPS 请求的服务器拥有更紧凑的配置。
http2	配置端口以接受 HTTP/2 连接。通常，为了使其工作，还应该指定 ssl 参数，但 Angie 也可以配置为在没有 SSL 的情况下接受 HTTP/2 连接。 自 1.2.0 版本弃用. 使用*http2* 指令代替。
quic	配置端口以接受 QUIC 连接。要使用此选项，Angie 必须启用并配置*HTTP3* 模块。设置 quic 后，您还可以指定 reuseport，以便可以使用多个工作进程。
proxy_protocol	允许指定在此端口上接受的所有连接应使用 PROXY 协议。

listen 指令可以有几个与套接字相关的系统调用特定的附加参数。这些参数可以在任何 listen 指令中指定，但对于给定的监听套接字只能指定一次。

setfib=*number*	此参数设置与监听套接字关联的路由表 FIB（*SO_SETFIB* 选项）。这目前仅在 FreeBSD 上有效。
fastopen=*number*	为监听套接字启用”TCP Fast Open”，并限制尚未完成三次握手的连接队列的最大长度。

小心

除非服务器能够处理多次接收相同的 SYN 数据包，否则请勿启用此功能。

backlog=*number*	设置 listen() 调用中的 backlog 参数，限制待处理连接队列的最大长度。默认情况下，FreeBSD、DragonFly BSD 和 macOS 上的 backlog 设置为 -1，其他平台上设置为 511。
rcvbuf=*size*	设置监听套接字的接收缓冲区大小（SO_RCVBUF 选项）。
sndbuf=*size*	设置监听套接字的发送缓冲区大小（SO_SNDBUF 选项）。
accept_filter=*filt*	设置监听套接字的接受过滤器名称（SO_ACCEPTFILTER 选项），该过滤器在将传入连接传递给 accept() 之前进行过滤。这仅在 FreeBSD 和 NetBSD 5.0+ 上有效。可能的值为 dataready 和 httpready。
deferred	指示在 Linux 上使用延迟的 accept()` （即:samp:`TCP_DEFER_ACCEPT 套接字选项）。
bind	指示为给定的 address:port 对进行单独的 bind() 调用。这很有用，因为如果有多个 listen 指令使用相同的端口但不同的地址，而其中一个 listen 指令监听给定 port 的所有地址（*:port），Angie 将仅 bind() 到 *:port。需要注意的是，在这种情况下，将调用 getsockname() 系统调用来确定接受连接的地址。如果使用了 setfib、fastopen、backlog、rcvbuf、sndbuf、accept_filter、deferred、ipv6only、reuseport 或 so_keepalive 参数，则对于给定的 address:port 对，将始终进行单独的 bind() 调用。
ipv6only=on | off	此参数通过 IPV6_V6ONLY 套接字选项确定监听在通配地址 [::] 上的 IPv6 套接字是否仅接受 IPv6 连接，或同时接受 IPv6 和 IPv4 连接。此参数默认开启。它只能在启动时设置一次。
reuseport	此参数指示为每个工作进程创建一个单独的监听套接字（在 Linux 3.9+ 和 DragonFly BSD 上使用 SO_REUSEPORT 套接字选项，或在 FreeBSD 12+ 上使用 SO_REUSEPORT_LB），允许内核在工作进程之间分配传入连接。此功能目前仅在 Linux 3.9+、DragonFly BSD 和 FreeBSD 12+ 上有效。 **☢ 小心** 不当使用此选项可能会导致安全隐患。
multipath	启用通过 Multipath TCP (MPTCP) 接受连接，Linux 内核从 5.6 版本开始支持此功能。此参数与 quic **不兼容**。

so_keepalive=on | off | [*keepidle*]:[*keepintvl*]:[*keepcnt*]

配置监听套接字的"TCP keepalive" 行为。

''	如果省略此参数，则操作系统的设置将适用于该套接字
on	为套接字开启 SO_KEEPALIVE 选项
off	为套接字关闭 SO_KEEPALIVE 选项

某些操作系统支持使用 TCP_KEEPIDLE、TCP_KEEPINTVL 和 TCP_KEEPCNT 套接字选项按套接字基础设置 TCP keepalive 参数。在此类系统上（目前为 Linux 2.4+、NetBSD 5+ 和 FreeBSD 9.0-STABLE），可以使用 keepidle、keepintvl 和 keepcnt 参数进行配置。可以省略一个或两个参数，在这种情况下，相应套接字选项的系统默认设置将生效。例如，

```
so_keepalive=30m::10
```

将把空闲超时（TCP_KEEPIDLE）设置为 30 分钟，将探测间隔（TCP_KEEPINTVL）保持为其系统默认值，并将探测计数（TCP_KEEPCNT）设置为 10 次探测。

示例：

```
listen 127.0.0.1 default_server accept_filter=dataready backlog=1024;
```

location

语法	`location` ([= | ~ | ~* | ^~] *uri* | `@name`)+ { ... }
默认	—
上下文	server, location

根据请求 URI 是否与任何匹配表达式匹配来设置配置。

匹配是在标准化 URI 上执行的，在解码以"%XX" 形式编码的文本、解析相对路径组件"." 和".." 的引用以及可能的压缩 两个或多个相邻斜杠为单个斜杠后进行。

location 可以通过前缀字符串或正则表达式定义。

正则表达式通过前置修饰符指定：

~*	不区分大小写的匹配
~	区分大小写的匹配

为了找到与请求匹配的位置，Angie 首先检查使用前缀字符串定义的位置（称为前缀位置）。在它们中，选择匹配前缀最长的位置并暂时存储。

> **备注**
>
> 对于诸如 macOS 之类的不区分大小写的操作系统，前缀字符串匹配是不区分大小写的。然而，匹配仅限于单字节区域。

然后，基于正则表达式的位置按其在配置文件中的出现顺序进行评估。它们的评估在第一个匹配处停止，并使用相应的配置。如果未找到匹配的正则表达式位置，Angie 将使用暂时存储的前缀位置的配置。

除了下面提到的一些例外情况外，`location` 块可以嵌套。

正则表达式位置可以定义捕获组，稍后可以与其他指令一起使用。

如果匹配前缀位置使用 `^~` 修饰符，则不检查正则表达式位置。

此外，`=` 修饰符为 `location` 启用精确 URI 匹配模式；如果找到精确匹配，则查找停止。例如，如果 `/` 请求频繁，定义 `location =/` 会加快其处理速度，因为查找在精确匹配处停止。显然，此类位置不能包含嵌套位置。

示例：

```
location =/ {
    #configuration A
}

location / {
    #configuration B
}

location /documents/ {
    #configuration C
}

location ^~/images/ {
    #configuration D
}

location ~*\.(gif|jpg|jpeg)$ {
    #configuration E
}
```

- 一个 / 请求匹配配置 A，
- 一个 /index.html 请求匹配配置 B，
- 一个 /documents/document.html 请求匹配配置 C，
- 一个 /images/1.gif 请求匹配配置 D，
- 一个 /documents/1.jpg 请求匹配配置 E。

备注

如果前缀 location 以斜杠字符结尾，并且*auto_redirect* 已启用，则会发生以下情况：当请求到达具有无尾斜杠但其他方面完全匹配前缀的 URI 时，将返回指向附加斜杠的请求 URI 的永久 301 代码重定向。

对于精确 URI 匹配位置，不应用重定向：

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

location =/user {
  proxy_pass http://login.example.com;
}
```

@ 前缀定义了一个 命名的 location。此类位置不用于常规请求处理，而是仅用于请求重定向。它们不能嵌套，也不能包含嵌套位置。

组合位置

多个定义相同配置块的 location 上下文可以通过将所有匹配表达式列在一个单独的 location 中并使用单个配置块进行压缩。这称为 组合 location。

假设前面的示例中配置 A、D 和 E 定义相同的配置；您可以将它们组合成一个 location：

```
location =/
         ~/images/
         ~*\.(gif|jpg|jpeg)$ {
   # 一般配置
}
```

命名的 location 也可以是组合的一部分：

```
location =/
         @named_combined {
   #...
}
```

小心

组合 location 不能在匹配表达式和其修饰符之间有空格。正确形式：location ~*/match(ing|es|er)$ ...。

备注

目前，组合 location 不能 **立即**包含*proxy_pass* 和类似的带有 URI 设置的指令，也不能包含 api 或 alias。然而，这些指令可以通过嵌套在组合位置中的位置使用。

log_not_found

语法	log_not_found on | off;
默认	log_not_found on;
上下文	http, server, location

启用或禁用将未找到文件的错误记录到*error_log*。

log_subrequest

语法	log_subrequest on | off;
默认	log_subrequest off;
上下文	http, server, location

启用或禁用将子请求记录到*access_log*。

max_headers

语法	`max_headers` *number*;
默认	`max_headers 1000;`
上下文	http, server

设置允许的客户端请求头字段的最大数量。如果超过此限制，将返回 `400 (Bad Request)` 错误。

当该指令在*server* 级别设置时，可能会应用默认服务器的值。有关更多信息，请参阅虚拟服务器选择 部分。

max_ranges

语法	`max_ranges` *number*;
默认	—
上下文	http, server, location

限制字节范围请求中允许的最大范围数量。超出限制的请求将被处理为没有指定字节范围。默认情况下，范围数量没有限制。

`0`	完全禁用字节范围支持

merge_slashes

语法	`merge_slashes on | off;`
默认	`merge_slashes on;`
上下文	http, server

启用或禁用将 URI 中两个或多个相邻的斜杠压缩为一个斜杠。

请注意，压缩对于前缀字符串和正则表达式位置的正确匹配至关重要。如果没有它，`//scripts/one.php` 请求将不会匹配

```
location /scripts/ { }
```

并可能被处理为静态文件。因此，它会被转换为 `/scripts/one.php`。

如果 URI 包含 base64 编码的名称，则可能需要关闭压缩，因为 base64 在内部使用"/" 字符。然而，出于安全考虑，最好避免关闭压缩。

如果在*server* 级别指定该指令，则可以使用默认服务器的值。

msie_padding

语法	`msie_padding on | off;`
默认	`msie_padding on;`
上下文	http, server, location

启用或禁用为状态大于 400 的 MSIE 客户端的响应添加注释，以增加响应大小到 512 字节。

msie_refresh

语法	`msie_refresh on | off;`
默认	`msie_refresh off;`
上下文	http, server, location

启用或禁用为 MSIE 客户端发出刷新而不是重定向。

open_file_cache

语法	`open_file_cache off;` `open_file_cache max=`*N* `[inactive=`*time*`];`
默认	`open_file_cache off;`
上下文	http, server, location

配置一个可以存储的缓存：

- 打开的文件描述符、它们的大小和修改时间；
- 目录存在的信息；
- 文件查找错误，例如” 找不到文件”、” 无读取权限” 等。

错误缓存应通过*open_file_cache_errors* 指令单独启用。

`max`	设置缓存中元素的最大数量；在缓存溢出时，最少使用的元素将被删除；
`inactive`	定义在此时间内未被访问的元素从缓存中移除的时间； 默认设置为 60 秒。
`off`	禁用缓存。

示例：

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

open_file_cache_errors

语法	`open_file_cache_errors on | off;`
默认	`open_file_cache_errors off;`
上下文	http, server, location

启用或禁用通过*open_file_cache* 缓存文件查找错误。

open_file_cache_min_uses

语法	`open_file_cache_min_uses` *number*;
默认	`open_file_cache_min_uses 1;`
上下文	http, server, location

设置在*open_file_cache* 指令的 `inactive` 参数配置的期间内，文件描述符保持在缓存中所需的最小文件访问次数。

open_file_cache_valid

语法	`open_file_cache_valid` *time*;
默认	`open_file_cache_valid 60s;`
上下文	http, server, location

设置*open_file_cache* 元素应被验证的时间。

output_buffers

语法	`output_buffers` *number size*;
默认	`output_buffers 2 32k;`
上下文	http, server, location

设置用于从磁盘读取响应的缓冲区的数量和大小。

port_in_redirect

语法	`port_in_redirect on | off;`
默认	`port_in_redirect on;`
上下文	http, server, location

启用或禁用在 Angie 发出的绝对 重定向中指定端口。

重定向中主服务器名称的使用由*server_name_in_redirect* 指令控制。

postpone_output

语法	`postpone_output` *size*;
默认	`postpone_output 1460;`
上下文	http, server, location

如果可能，将推迟客户端数据的传输，直到 Angie 至少有 *size* 字节的数据可发送。

`0`	禁用推迟数据传输

read_ahead

语法	`read_ahead` *size*;
默认	`read_ahead 0;`
上下文	http, server, location

设置内核在处理文件时的预读量。

在 Linux 上，使用 `posix_fadvise(0, 0, 0, POSIX_FADV_SEQUENTIAL)` 系统调用，因此大小参数被忽略。

在 FreeBSD 上，使用自 FreeBSD 9.0-CURRENT 以来支持的 `fcntl(O_READAHEAD,` size *)* 系统调用。

recursive_error_pages

语法	`recursive_error_pages on` | `off`;
默认	`recursive_error_pages off;`
上下文	http, server, location

启用或禁用使用*error_page* 指令进行多次重定向。这种重定向的数量是*limited*。

request_pool_size

语法	`request_pool_size` *size*;
默认	`request_pool_size 4k;`
上下文	http, server

允许精确调整每个请求的内存分配。此指令对性能影响很小，通常不应使用。

reset_timedout_connection

语法	`reset_timedout_connection on | off;`
默认	`reset_timedout_connection off;`
上下文	http, server, location

启用或禁用重置超时连接和使用非标准代码 444 关闭的连接。重置的执行方式如下：在关闭套接字之前，为其设置 `SO_LINGER` 选项，超时值为 0。

> **备注**
>
> 超时的 keep-alive 连接正常关闭。

resolver

语法	`resolver` 地址... [`valid=` 时间] [`ipv4=on | off`] [`ipv6=on | off`] [`status_zone=`*zone*];
默认	—
上下文	http, server, location, upstream

配置用于将上游服务器名称解析为地址的名称服务器，例如：

```
resolver 127.0.0.53 [::1]:5353;
```

地址可以指定为域名或 IP 地址，并带有可选端口。如果未指定端口，则使用端口 53。名称服务器以轮询方式查询。

默认情况下，Angie 使用响应的 TTL 值缓存答案。

`valid`	可选参数允许覆盖响应缓存有效期

```
resolver 127.0.0.53 [::1]:5353 valid=30s;
```

默认情况下，Angie 在解析时将查找 IPv4 和 IPv6 地址。

`ipv4=off`	禁用查找 IPv4 地址
`ipv6=off`	禁用查找 IPv6 地址

`status_zone`	可选参数；启用在指定区域中收集 DNS 服务器请求和响应指标（*/status/resolvers/<zone>*）。

小技巧

为防止 DNS 欺骗，建议在适当安全的受信任本地网络中配置 DNS 服务器。

小技巧

当在 Docker 中运行时，使用其内部 DNS 服务器地址，例如 `127.0.0.11`。

resolver_timeout

语法	`resolver_timeout` 时间;
默认	`resolver_timeout 30s;`
上下文	http, server, location, upstream

为名称解析设置超时时间，例如：

```
resolver_timeout 5s;
```

root

语法	`root` 路径;
默认	`root html;`
上下文	http, server, location, if in location

为请求设置根目录。例如，使用以下配置

```
location /i/ {
  root /data/w3;
}
```

`/data/w3/i/top.gif` 文件将作为 `/i/top.gif` 请求的响应发送。

路径 ‘值可以包含变量，除了*:ref:‘$document_root <v_document_root>* 和*$realpath_root*。

文件的路径仅通过将 URI 添加到 root 指令的值来构造。如果 URI 需要被修改，则应使用*alias* 指令。

satisfy

语法	`satisfy all` | `any`;
默认	`satisfy all;`
上下文	http, server, location

如果所有（all）或至少一个（any）模块允许访问，将允许访问：*Access*, *Auth Basic* 或*Auth Request*。

```
location / {
  satisfy any;

  allow 192.168.1.0/32;
  deny  all;

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

send_lowat

语法	send_lowat 大小;
默认	send_lowat 0;
上下文	http, server, location

如果指令设置为非零值，Angie 将尝试通过使用 NOTE_LOWAT 标志的*kqueue* 方法或 SO_SNDLOWAT 套接字选项来最小化客户端套接字上的发送操作数。在这两种情况下，使用指定的大小。

send_timeout

语法	send_timeout 时间;
默认	send_timeout 60s;
上下文	http, server, location

设置向客户端传输响应的超时时间。超时仅在两个连续的写操作之间设置，而不是对整个响应的传输。如果客户端在此时间内没有接收到任何内容，连接将被关闭。

sendfile

语法	sendfile on | off;
默认	sendfile off;
上下文	http, server, location, if in location

启用或禁用 sendfile() 的使用。

aio 可用于预加载 sendfile() 的数据：

```
location /video/ {
  sendfile       on;
  tcp_nopush     on;
```

```
  aio           on;
}
```

在此配置中，`sendfile()` 被调用时带有 `SF_NODISKIO` 标志，这使其不会在磁盘 I/O 上阻塞，而是报告数据不在内存中。然后，Angie 通过读取一个字节来启动异步数据加载。在第一次读取时，FreeBSD 内核将文件的前 128K 字节加载到内存中，尽管后续读取将仅以 16K 的块加载数据。这可以通过*read_ahead* 指令进行更改。

sendfile_max_chunk

语法	`sendfile_max_chunk` 大小;
默认	`sendfile_max_chunk 2m;`
上下文	http, server, location

限制可以在单个 `sendfile()` 调用中传输的数据量。没有限制时，一个快速连接可能会完全占用工作进程。

server

语法	`server { ... }`
默认	—
上下文	http

设置虚拟服务器的配置。没有明确区分基于 IP（基于 IP 地址）和基于名称（基于"Host" 请求头字段）的虚拟服务器。相反，*listen* 指令描述了应该接受服务器连接的所有地址和端口，而*server_name* 指令列出了所有服务器名称。

示例配置在*Angie 如何处理请求* 文档中提供。

server_name

语法	`server_name` 名称...;
默认	`server_name "";`
上下文	server

设置虚拟服务器的名称，例如：

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

第一个名称成为主要服务器名称。

服务器名称可以包含一个星号（"*"）以替代名称的第一部分或最后一部分：

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

这样的名称称为通配符名称。

上述名称中的前两个可以合并为一个：

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

也可以在服务器名称中使用正则表达式，在名称前加上波浪号（”~”）：

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

正则表达式可以包含捕获，稍后可以在其他指令中使用：

```
server {
  server_name ~^(www\.)?(.+)$;

  location / {
     root /sites/$2;
  }
}

server {
  server_name _;

  location / {
     root /sites/default;
  }
}
```

正则表达式中的命名捕获创建变量，稍后可以在其他指令中使用：

```
server {
  server_name ~^(www\.)?(?<domain>.+)$;

  location / {
     root /sites/$domain;
  }
}

server {
  server_name _;
```

```
    location / {
        root /sites/default;
    }
}
```

备注

如果指令设置为*$hostname*，将使用 Web 服务器的主机名。

您还可以指定空服务器名称（""）：

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

在按名称搜索虚拟服务器时，如果多个选项匹配（例如，通配符和正则表达式），将按照以下优先级顺序选择第一个匹配的选项：

- 精确名称；
- 以通配符开头的最长名称，例如 *.example.com；
- 以通配符结尾的最长名称，例如 mail.*；
- 第一个匹配的正则表达式（按出现顺序），包括空名称。

注意

要使 server_name 与 TLS 一起使用，您需要终止 TLS 连接。该指令匹配 HTTP 请求中的 Host，因此握手必须完成并且连接被解密。

server_name_in_redirect

语法	server_name_in_redirect on | off;
默认	server_name_in_redirect off;
上下文	http, server, location

启用或禁用在 Angie 发出的绝对重定向 中使用由*server_name* 指令指定的主要服务器名称。

on	使用由*server_name* 指令指定的主要服务器名称
off	使用”Host” 请求头字段中的名称。如果该字段不存在，则使用服务器的 IP 地址。

重定向中使用端口的方式由*port_in_redirect* 指令控制。

server_names_hash_bucket_size

语法	`server_names_hash_bucket_size` 大小;
默认	`server_names_hash_bucket_size 32 | 64 | 128;`
上下文	http

设置服务器名称哈希表的桶大小。默认值取决于处理器缓存行的大小。有关设置哈希表的详细信息，请参见单独的文档。

server_names_hash_max_size

语法	`server_names_hash_max_size` 大小;
默认	`server_names_hash_max_size 512;`
上下文	http

设置服务器名称哈希表的最大大小。有关设置哈希表的详细信息，请参见单独的文档。

server_tokens

语法	`server_tokens on | off | build` | 字符串*;*
默认	`server_tokens` *on;*
上下文	http, server, location

启用或禁用在错误页面和 `Server` 响应头字段中发出 Angie 版本。*build* 参数启用发出构建名称，该名称由相应的 configure 参数设置，以及版本。

Added in version 1.1.0: PRO

在 Angie PRO 中，如果指令设置为 字符串，该字符串也可以包含变量，错误页面和 `Server` 响应头字段将使用该字符串的变量插值值，而不是服务器名称、版本和构建名称。空 字符串禁用发出 `Server` 字段。

status_zone

语法	`status_zone off` | *zone* | *key* `zone=`*zone*[:*count*];
默认	—
上下文	server, location, if in location

分配一个共享内存区域以收集*/status/http/location_zones/<zone>* 和*/status/http/server_zones/<zone>* 的指标。

多个 `server` 上下文可以共享同一区域以进行数据收集；特殊值 `off` 禁用在嵌套 `location` 块中的数据收集。

单值 *zone* 语法在同一共享内存区域中聚合其上下文的所有指标：

```
server {

    listen 80;
    server_name *.example.com;

    status_zone single;
    # ...
}
```

key	一个包含变量的字符串，其值决定了请求在区域内的分组。所有在替换后产生相同值的请求都会被分到同一组。如果替换产生空值，则不会更新指标。
zone	共享内存区域的名称。
count （可选）	收集指标的最大分组数。如果新的 *key* 值超过此限制，则会在 zone 下进行分组。默认值为 1。

在以下示例中，所有共享相同 $host 值的请求被分组到 host_zone 中。每个唯一的 $host 的指标会单独跟踪，直到有 10 个指标组。一旦达到此限制，任何额外的 $host 值将被包含在 host_zone 下：

```
server {

    listen 80;
    server_name *.example.com;

    status_zone $host zone=host_zone:10;

    location / {

        proxy_pass http://example.com;
    }
}
```

因此，生成的指标在 API 输出中在各个主机之间分开。

subrequest_output_buffer_size

语法	subrequest_output_buffer_size *size*;
默认	subrequest_output_buffer_size 4k | 8k;
上下文	http, server, location

设置用于存储子请求响应体的缓冲区大小。默认情况下，缓冲区大小等于一个内存页面。这是 4K 或 8K，具体取决于平台。不过，可以将其设置得更小。

备注

此指令仅适用于响应体存储在内存中的子请求。例如，这种子请求由*SSI* 创建。

tcp_nodelay

语法	`tcp_nodelay on	off;`
默认	`tcp_nodelay on;`
上下文	http, server, location

启用或禁用 `TCP_NODELAY` 选项。当连接进入保持活动状态时，该选项被启用。此外，在 SSL 连接、无缓冲代理和 *WebSocket* 代理 中也会启用该选项。

tcp_nopush

语法	`tcp_nopush on	off;`
默认	`tcp_nopush off;`
上下文	http, server, location

启用或禁用 FreeBSD 上的 `TCP_NOPUSH` 套接字选项或 Linux 上的 `TCP_CORK` 套接字选项。这些选项仅在使用*sendfile* 时启用。启用该选项可以

- 在 Linux 和 FreeBSD 4.* 上将响应头和文件开头一起发送在一个数据包中；
- 以完整数据包发送文件。

try_files

语法	`try_files` file ... *uri*; `try_files` file ... =*code*;
默认	—
上下文	server, location

按指定顺序检查文件的存在性，并使用第一个找到的文件进行请求处理；处理在当前上下文中进行。文件路径根据*root* 和*alias* 指令从文件参数构建。可以通过在名称末尾指定斜杠（例如 `$uri/`）来检查目录的存在性。如果没有找到任何文件，则会进行内部重定向到最后一个参数中指定的 `uri`。例如：

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

location = /images/default.gif {
  expires 30s;
}
```

最后一个参数也可以指向命名 location，如下所示。最后一个参数也可以是代码：

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

在以下示例中，

```
location / {
  try_files $uri $uri/ @drupal;
}
```

try_files 指令等同于

```
location / {
  error_page 404 = @drupal;
  log_not_found off;
}
```

在这里，

```
location ~ \.php$ {
  try_files $uri @drupal;

  fastcgi_pass ...;

  fastcgi_param SCRIPT_FILENAME /path/to$fastcgi_script_name;

#  ...
}
```

try_files 在将请求传递给 FastCGI 服务器之前检查 PHP 文件的存在性。

```
location / {
  try_files /system/maintenance.html
            $uri $uri/index.html $uri.html
            @mongrel;
}

location @mongrel {
  proxy_pass http://mongrel;
}
```

```
location / {
  try_files $uri $uri/ @drupal;
}

location ~ \.php$ {
  try_files $uri @drupal;
```

```
  fastcgi_pass ...;

  fastcgi_param SCRIPT_FILENAME /path/to$fastcgi_script_name;
  fastcgi_param SCRIPT_NAME     $fastcgi_script_name;
  fastcgi_param QUERY_STRING    $args;

#  ... 其他 fastcgi_param
}

location @drupal {
  fastcgi_pass ...;

  fastcgi_param SCRIPT_FILENAME /path/to/index.php;
  fastcgi_param SCRIPT_NAME     /index.php;
  fastcgi_param QUERY_STRING    q=$uri&$args;

#  ... 其他 fastcgi_param
}
```

```
location / {
  try_files $uri $uri/ @wordpress;
}

location ~ \.php$ {
  try_files $uri @wordpress;

  fastcgi_pass ...;

  fastcgi_param SCRIPT_FILENAME /path/to$fastcgi_script_name;
#  ... 其他 fastcgi_param
}

location @wordpress {
  fastcgi_pass ...;

  fastcgi_param SCRIPT_FILENAME /path/to/index.php;
#  ... 其他 fastcgi_param
}
```

types

语法	types { ... }
默认	types *text/html html; image/gif gif; image/jpeg jpg;*
上下文	http, server, location

将文件名扩展名映射到响应的 MIME 类型。扩展名不区分大小写。多个扩展名可以映射到同一类型，例如：

```
types {
  application/octet-stream bin exe dll;
  application/octet-stream deb;
  application/octet-stream dmg;
}
```

一个相对完整的映射表与 Angie 一起分发，存放在 `conf/mime.types` 文件中。

要使特定位置对所有请求发出"application/octet-stream" MIME 类型，可以使用以下配置：

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

types_hash_bucket_size

语法	`types_hash_bucket_size` *size*;
默认	`types_hash_bucket_size 64;`
上下文	http, server, location

设置类型哈希表的桶大小。有关设置哈希表的详细信息，请参阅单独的*document*。

types_hash_max_size

Syntax	`types_hash_max_size` *size*;
Default	`types_hash_max_size 1024;`
Context	http, server, location

设置类型哈希表的最大大小。有关设置哈希表的详细信息，请参阅单独的*document*。

underscores_in_headers

Syntax	`underscores_in_headers on` | `off`;
Default	`underscores_in_headers off;`
Context	http, server

启用或禁用在客户端请求头字段中使用下划线。禁用下划线时，名称中包含下划线的请求头字段被标记为无效，并受到*ignore_invalid_headers* 指令的约束。

如果该指令在*server* 级别指定，则可以使用默认服务器的值。

variables_hash_bucket_size

Syntax	`variables_hash_bucket_size` *size*;
Default	`variables_hash_bucket_size 64;`
Context	http

设置变量哈希表的桶大小。有关设置哈希表的详细信息，请参阅单独的*document*。

variables_hash_max_size

Syntax	`variables_hash_max_size` *size*;
Default	`variables_hash_max_size 1024;`
Context	http

设置变量哈希表的最大大小。有关设置哈希表的详细信息，请参阅单独的*document*。

内置变量

`http_core` 模块支持名称与 Apache 服务器变量匹配的内置变量。首先，这些变量表示客户端请求头字段，例如 `$http_user_agent`、`$http_cookie` 等。此外，还有其他变量：

$angie_version

Angie 版本

$arg_<name>

请求行中指定 *name* 的参数

$args

请求行中的参数

$binary_remote_addr

以二进制形式表示的客户端地址，IPv4 地址的值长度始终为 4 字节，IPv6 地址为 16 字节

$body_bytes_sent

发送到客户端的字节数，不包括响应头；该变量与 *mod_log_config* Apache 模块的”%B” 参数兼容

$bytes_sent

发送到客户端的字节数

`$connection`

连接序列号

`$connection_requests`

通过连接发出的当前请求数量

`$connection_time`

连接时间（以秒为单位，精确到毫秒）

`$content_length`

"Content-Length" 请求头字段

`$content_type`

"Content-Type" 请求头字段

`$cookie_<name>`

指定 *name* 的 cookie

`$document_root`

当前请求的*root* 或*alias* 指令的值

`$document_uri`

与*$uri* 相同

`$host`

按优先顺序：请求行中的主机名、"Host" 请求头字段中的主机名，或与请求匹配的服务器名

`$hostname`

主机名

`$http_<name>`

任意请求头字段；*name* 是转换为小写、用下划线替代破折号的字段名

`$https`

如果连接在 SSL 模式下运行则为 *on*，否则为空字符串

`$is_args`

如果请求行有参数则为 *?*，否则为空字符串

`$limit_rate`

设置该变量以启用响应速率限制；请参见*limit_rate*

`$msec`

当前时间（以秒为单位，精确到毫秒）

`$pid`

工作进程的 PID

`$pipe`

如果请求是管道化的则为 *p*，否则为 .

`$proxy_protocol_addr`

来自 PROXY 协议头的客户端地址。

必须通过在*listen* 指令中设置 *proxy_protocol* 参数来先启用 PROXY 协议。

`$proxy_protocol_port`

来自 PROXY 协议头的客户端端口。

必须通过在*listen* 指令中设置 *proxy_protocol* 参数来先启用 PROXY 协议。

`$proxy_protocol_server_addr`

来自 PROXY 协议头的服务器地址。

必须通过在*listen* 指令中设置 *proxy_protocol* 参数来先启用 PROXY 协议。

`$proxy_protocol_server_port`

来自 PROXY 协议头的服务器端口。

必须通过在*listen* 指令中设置 *proxy_protocol* 参数来先启用 PROXY 协议。

`$proxy_protocol_tlv_<name>`

来自 PROXY 协议头的 TLV。*name* 可以是 TLV 类型或其数字值。在后者情况下，该值为十六进制，并应以 *0x* 为前缀：

```
$proxy_protocol_tlv_alpn
$proxy_protocol_tlv_0x01
```

SSL TLV 也可以通过 TLV 类型名称或其数字值访问，前面都加上 `ssl_`：

```
$proxy_protocol_tlv_ssl_version
$proxy_protocol_tlv_ssl_0x21
```

支持以下 TLV 类型名称：

- *alpn (0x01)* - 在连接中使用的上层协议
- *authority (0x02)* - 客户端传递的主机名值
- *unique_id (0x05)* - 唯一连接 ID
- *netns (0x30)* - 命名空间的名称
- *ssl (0x20)* - 二进制 SSL TLV 结构

支持以下 SSL TLV 类型名称：

- *ssl_version (0x21)* - 客户端连接中使用的 SSL 版本
- *ssl_cn (0x22)* - SSL 证书的通用名称
- *ssl_cipher (0x23)* - 使用的密码名称
- *ssl_sig_alg (0x24)* - 用于签署证书的算法
- *ssl_key_alg (0x25)* - 公钥算法

此外，支持以下特殊 SSL TLV 类型名称：

- *ssl_verify* - 客户端 SSL 证书验证结果，如果客户端提供了证书并成功验证则为 *0*，否则为非零值。

必须通过在*listen* 指令中设置 *proxy_protocol* 参数来先启用 PROXY 协议。

$query_string

与*$args* 相同

$realpath_root

与当前请求的*root* 或*alias* 指令的值对应的绝对路径名，所有符号链接已解析为真实路径

$remote_addr

客户端地址

$remote_port

客户端端口

`$remote_user`

通过基本身份验证提供的用户名

`$request`

完整的原始请求行

`$request_body`

请求体。

该变量的值在通过*proxy_pass*、*fastcgi_pass*、*uwsgi_pass* 和*scgi_pass* 指令处理的位置可用，当请求体被读取到*memory buffer* 时。

`$request_body_file`

请求体的临时文件名称。

处理结束时，需要删除该文件。要始终将请求体写入文件，必须启用*client_body_in_file_only*。当在代理请求或对 FastCGI/uwsgi/SCGI 服务器的请求中传递临时文件名称时，必须通过*proxy_pass_request_body off*、*fastcgi_pass_request_body off*、*uwsgi_pass_request_body off* 或*scgi_pass_request_body off* 指令分别禁用请求体的传递。

`$request_completion`

如果请求已完成则为”OK”，否则为空字符串

`$request_filename`

基于*root* 或*alias* 指令以及请求 URI 的当前请求的文件路径

`$request_id`

由 16 个随机字节生成的唯一请求标识符，采用十六进制格式

`$request_length`

请求长度（包括请求行、头部和请求体）

`$request_method`

请求方法，通常为 `GET` 或 `POST`

`$request_time`

请求处理时间（以秒为单位，精确到毫秒）；自从从客户端读取的第一个字节以来的经过时间

`$request_uri`

完整的原始请求 URI（带参数）

`$scheme`

请求方案，”http” 或”https”

`$sent_http_<name>`

任意响应头字段；*name* 是转换为小写、用下划线替代破折号的字段名

`$sent_trailer_<name>`

在响应末尾发送的任意字段；*name* 是转换为小写、用下划线替代破折号的字段名

`$server_addr`

接受请求的服务器的地址。计算该变量的值通常需要一个系统调用。为避免此情况，*listen* 指令必须指定地址并使用 `bind` 参数。

`$server_name`

接受请求的服务器的名称

`$server_port`

接受请求的服务器的端口

`$server_protocol`

请求协议，通常是”HTTP/1.0”、”HTTP/1.1” 或”HTTP/2.0”

`$status`

响应状态

`$time_iso8601`

本地时间，符合 ISO 8601 标准格式

`$time_local`

本地时间，符合通用日志格式

`$tcpinfo_rtt, $tcpinfo_rttvar, $tcpinfo_snd_cwnd, $tcpinfo_rcv_space`

有关客户端 TCP 连接的信息；在支持 `TCP_INFO` 套接字选项的系统上可用

`$uri`

请求中的当前 URI，*normalized*。在请求处理期间，`$uri` 的值可能会发生变化，例如在进行内部重定向或使用索引文件时。

流模块

Access

该模块允许限制某些客户端地址的访问。

配置示例

```
server {
    ...
    deny  192.168.1.1;
    allow 192.168.1.0/24;
    allow 10.1.1.0/16;
    allow 2001:0db8::/32;
    deny  all;
}
```

规则按顺序检查，直到找到第一个匹配项。在此示例中，只有 IPv4 网络 `10.1.1.0/16` 和 `192.168.1.0/24`（不包括地址 `192.168.1.1`）以及 IPv6 网络 `2001:0db8::/32` 允许访问。

指令

allow

语法	`allow` *address* | *CIDR* | `unix:` | `all`;
默认值	—
上下文	stream, server

允许指定网络或地址的访问。如果指定了特殊值 `unix:`，则允许所有 UNIX 域套接字的访问。

deny

语法	`deny` *address* | *CIDR* | `unix:` | `all`;
默认值	—
上下文	stream, server

拒绝指定网络或地址的访问。如果指定了特殊值 `unix:`，则拒绝所有 UNIX 域套接字的访问。

ACME

允许使用 ACME 协议为:samp:*stream* 上下文中定义的服务器自动获取证书。

在:ref: 从源代码构建 <*sourcebuild*> 时，该模块默认不会被构建；必须使用 构建参数 `--with-stream_acme_module` （同时需要:option:*!--with-http_acme_module* ）来启用它。在来自:ref: 我们的仓库 <*install-packages*> 的软件包和镜像中，该模块已包含在构建中。

重要

为了正确运行，`stream` 块必须位于:samp:*http* 块之后。这是因为 stream 模块使用在 HTTP 配置解析期间创建的客户端定义。

配置示例

有关配置示例和设置说明，请参阅流模块中的 *ACME* 部分。

指令

acme

语法	`acme` *name*;
默认值	—
上下文	server

对于在所有引用 HTTP 模块中给定 'name'的:ref:*ACME* 客户端 <*acme_client*> 的所有:ref:*s_server* 块中通过:ref:*s_server_name* 指令指定的所有域名，将获取单个证书；如果:samp:*server_name* 配置发生变化，证书将被更新以适应这些变化。

在每次 Angie 启动时，会为所有缺少有效证书的域名请求新证书。可能的原因包括证书过期、文件丢失或无法读取，以及证书设置的变更。

备注

目前，通过正则表达式指定的域名不受支持，将被跳过。

通配符域名仅在:samp:*acme_client* 中的 `challenge=dns` 模式下支持。

可以多次指定此指令以加载不同类型的证书，例如 RSA 和 ECDSA：

```
server {

    listen 12345 ssl;
    server_name example.com www.example.com;

    ssl_certificate $acme_cert_rsa;
```

```
    ssl_certificate_key $acme_cert_key_rsa;

    ssl_certificate $acme_cert_ecdsa;
    ssl_certificate_key $acme_cert_key_ecdsa;

    acme rsa;
    acme ecdsa;
}
```

内嵌变量

`$acme_cert_<name>`

具有此 'name'的客户端获取的最后一个证书文件（如果有）的内容。

`$acme_cert_key_<name>`

具有此 'name'的客户端使用的证书密钥文件的内容。

> **重要**
>
> 证书文件仅在 ACME 客户端至少获取了一个证书后才可用，而密钥文件在启动后立即可用。

Geo

该模块根据客户端 IP 地址创建具有不同值的变量。

配置示例

```
geo $geo {
    default        0;

    127.0.0.1      2;
    192.168.1.0/24 1;
    10.1.0.0/16    1;

    ::1            2;
    2001:0db8::/32 1;
}
```

指令

geo

语法	geo [*$address*] *$variable* { ... }
默认	—
上下文	stream

描述指定变量的值如何依赖于客户端 IP 地址。默认情况下，地址取自*$remote_addr* 变量，但也可以取自其他变量，例如：

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

备注

因为变量仅在使用时才被评估，所以即使存在大量声明的 geo 变量，也不会对连接处理造成额外开销。

如果变量的值不是一个有效的 IP 地址，则使用”255.255.255.255” 地址。

地址可以指定为 CIDR 表示法中的前缀（包括单个地址）或作为范围。

还支持以下特殊参数：

delete	删除指定的网络
default	当客户端地址不匹配任何指定地址时，赋给变量的值。当地址以 CIDR 表示法指定时，可以用”0.0.0.0/0” 和”::/0” 代替 default。当未指定 default 时，默认值为空字符串
include	包含一个包含地址和值的文件。可以有多个包含。
ranges	指示地址以范围形式指定。此参数应放在首位。为了加快 geo 基础的加载，地址应按升序排列。

示例：

```
geo $country {
    default         ZZ;
    include         conf/geo.conf;
    delete          127.0.0.0/16;

    127.0.0.0/24    US;
    127.0.0.1/32    RU;
    10.1.0.0/16     RU;
```

```
    192.168.1.0/24 UK;
}
```

conf/geo.conf 文件可能包含以下行：

```
10.2.0.0/16    RU;
192.168.2.0/24 RU;
```

使用最具体匹配的值。例如，对于 127.0.0.1 地址，将选择值 RU，而不是 US。

使用范围的示例：

```
geo $country {
    ranges;
    default                   ZZ;
    127.0.0.0-127.0.0.0       US;
    127.0.0.1-127.0.0.1       RU;
    127.0.0.2-127.0.0.255     US;
    10.1.0.0-10.1.255.255     RU;
    192.168.1.0-192.168.1.255 UK;
}
```

GeoIP

使用预编译的 MaxMind 数据库，根据客户端 IP 地址创建变量。

当使用支持 IPv6 的数据库时，IPv4 地址会被查找为 IPv4 映射的 IPv6 地址。

在 从源代码构建时，此模块默认不构建；需要使用 --with-stream_geoip_module 构建选项启用。

重要

此模块需要 MaxMind GeoIP 库。

配置示例

```
stream {
    geoip_country         GeoIP.dat;
    geoip_city            GeoLiteCity.dat;

    map $geoip_city_continent_code $nearest_server {
        default        example.com;
        EU             eu.example.com;
        NA             na.example.com;
        AS             as.example.com;
    }
#   ...
}
```

指令

geoip_country

语法	`geoip_country` *file*;
默认值	—
上下文	stream

指定用于根据客户端 IP 地址确定国家的数据库。使用此数据库时可以使用以下变量：

`$geoip_country_c`	两字母国家代码，例如，”RU”，”US”。
`$geoip_country_c`	三字母国家代码，例如，”RUS”，”USA”。
`$geoip_country_n`	国家名称，例如，”Russian Federation”，”United States”。

geoip_city

语法	`geoip_city` *file*;
默认值	—
上下文	stream

指定用于根据客户端 IP 地址确定国家、地区和城市的数据库。使用此数据库时可以使用以下变量：

`$geoip_city_cont`	两字母洲代码，例如，”EU”，”NA”。
`$geoip_city_coun`	两字母国家代码，例如，”RU”，”US”。
`$geoip_city_coun`	三字母国家代码，例如，”RUS”，”USA”。
`$geoip_city_coun`	国家名称，例如，”Russian Federation”，”United States”。
`$geoip_dma_code`	美国的 DMA 区域代码（也称为” 地铁代码”），根据 Google AdWords API 中的地理定位。
`$geoip_latitude`	纬度。
`$geoip_longitude`	经度。
`$geoip_region`	两个字符的国家地区代码（地区、领土、州、省、联邦土地等），例如，”48”，”DC”。
`$geoip_region_na`	国家地区名称（地区、领土、州、省、联邦土地等），例如，”Moscow City”，”District of Columbia”。
`$geoip_city`	城市名称，例如，”Moscow”，”Washington”。
`$geoip_postal_co`	邮政编码。

geoip_org

语法	geoip_org *file*;
默认值	—
上下文	stream

指定用于根据客户端 IP 地址确定组织的数据库。使用此数据库时可以使用以下变量：

$geoip_org	组织名称，例如，”The University of Melbourne”。

JS

该模块用于在 njs 中实现处理程序——njs 是 JavaScript 语言的一个子集。

在我们的代码库中，该模块是 动态构建的并作为一个名为 angie-module-njs 或 angie-pro-module-njs 的独立包提供。

> **备注**
>
> 此软件包还提供一个轻量级版本，名为 ...-njs-light；但它无法与常规版本同时使用。

配置示例

```
stream {
    js_import stream.js;

    js_set $bar stream.bar;
    js_set $req_line stream.req_line;

    server {
        listen 12345;

        js_preread stream.preread;
        return     $req_line;
    }

    server {
        listen 12346;

        js_access  stream.access;
        proxy_pass 127.0.0.1:8000;
        js_filter  stream.header_inject;
    }
}
```

```
http {
    server {
        listen 8000;
        location / {
            return 200 $http_foo\n;
        }
    }
}
```

stream.js 文件内容：

```
var line = '';

function bar(s) {
    var v = s.variables;
    s.log("hello from bar() handler!");
    return "bar-var" + v.remote_port + "; pid=" + v.pid;
}

function preread(s) {
    s.on('upload', function (data, flags) {
        var n = data.indexOf('\n');
        if (n != -1) {
            line = data.substr(0, n);
            s.done();
        }
    });
}

function req_line(s) {
    return line;
}

// 读取 HTTP 请求行。
// 收集 'req' 中的字节，直到
// 请求行被读取。
// 将 HTTP 头注入客户端请求

var my_header =  'Foo: foo';
function header_inject(s) {
    var req = '';
    s.on('upload', function(data, flags) {
        req += data;
        var n = req.search('\n');
        if (n != -1) {
            var rest = req.substr(n + 1);
            req = req.substr(0, n + 1);
            s.send(req + my_header + '\r\n' + rest, flags);
```

```
            s.off('upload');
        }
    });
}

function access(s) {
    if (s.remoteAddress.match('^192.*')) {
        s.deny();
        return;
    }

    s.allow();
}

export default {bar, preread, req_line, header_inject, access};
```

指令

js_access

语法	`js_access` *function* | *module.function*;
默认值	—
上下文	stream, server

设置将在访问阶段 调用的 njs 函数。可以引用模块函数。

当流会话首次达到访问阶段 时，该函数会被调用一次。该函数使用以下参数调用：

s	stream session 对象

在此阶段，可以使用 s.on() 方法为每个传入的数据块执行初始化或注册回调，直到调用以下方法之一：s.done()，s.decline()，s.allow()。一旦调用了这些方法之一，流会话处理将切换到下一个阶段，并且所有当前的 s.on() 回调将被丢弃。

js_fetch_buffer_size

语法	`js_fetch_buffer_size` *size*;
默认值	`js_fetch_buffer_size 16k;`
上下文	stream, server

设置用于读取和写入 Fetch API 的缓冲区大小。

js_fetch_ciphers

语法	`js_fetch_ciphers` *ciphers*;
默认值	`js_fetch_ciphers HIGH:!aNULL:!MD5;`
上下文	stream, server

指定 Fetch API 的 HTTPS 连接中启用的加密套件。这些加密套件以 OpenSSL 库理解的格式指定。

密码列表取决于已安装的 OpenSSL 版本。可以使用 `openssl ciphers` 命令查看完整列表。

js_fetch_max_response_buffer_size

语法	`js_fetch_max_response_buffer_size` *size*;
默认值	`js_fetch_max_response_buffer_size 1m;`
上下文	stream, server

设置使用 Fetch API 接收的响应的最大大小。

js_fetch_protocols

语法	`js_fetch_protocols` [`TLSv1`] [`TLSv1.1`] [`TLSv1.2`] [`TLSv1.3`];
默认值	`js_fetch_protocols TLSv1 TLSv1.1 TLSv1.2;`
上下文	stream, server

启用 Fetch API 的 HTTPS 连接的指定协议。

js_fetch_timeout

语法	`js_fetch_timeout` *time*;
默认值	`js_fetch_timeout 60s;`
上下文	stream, server

定义 Fetch API 的读取和写入超时时间。该超时时间仅在两次连续的读/写操作之间设置，而不是针对整个响应。如果在此时间内没有传输数据，连接将被关闭。

js_fetch_trusted_certificate

语法	`js_fetch_trusted_certificate` *file*;
默认值	—
上下文	stream, server

指定一个包含 PEM 格式的受信 CA 证书的文件，用于使用 Fetch API 验证 HTTPS 证书。

js_fetch_verify

语法	`js_fetch_verify on | off;`
默认值	`js_fetch_verify on;`
上下文	stream, server

启用或禁用使用 Fetch API 验证 HTTPS 服务器证书。

js_fetch_verify_depth

语法	`js_fetch_verify_depth` *number*;
默认值	`js_fetch_verify_depth 100;`
上下文	stream, server

设置 Fetch API 的 HTTPS 服务器证书链的验证深度。

js_filter

语法	`js_filter` *function* | *module.function*;
默认值	—
上下文	stream, server

设置数据过滤器。可以引用模块函数。

当流会话达到内容阶段 时，过滤器函数会被调用一次。过滤器函数使用以下参数调用：

`s`	stream session 对象

在此阶段，可以使用 s.on() 方法为每个传入的数据块执行初始化或注册回调。可以使用 s.off() 方法注销回调并停止过滤。

js_import

语法	`js_import` *module.js* | *export_name from module.js*;
默认值	—
上下文	stream, server

导入在 njs 中实现位置和变量处理程序的模块。*export_name* 用作访问模块函数的命名空间。如果未指定 *export_name*，则使用模块名称作为命名空间。

```
js_import stream.js;
```

在此，模块名称 *stream* 被用作命名空间。

可以指定多个 *js_import* 指令。

js_path

语法	`js_path` *path*;
默认值	—
上下文	stream, server

设置 njs 模块的附加路径。

js_preload_object

语法	`js_preload_object` *name.json* | *name* from *file.json*;
默认值	—
上下文	stream, server

在配置时预加载一个不可变对象。*name* 用作在 njs 代码中通过该名称访问对象的全局变量名。如果未指定 *name*，则使用文件名。

```
js_preload_object map.json;
```

在此，*map* 作为访问预加载对象时使用的名称。

可以指定多个 *js_preload_object* 指令。

js_preread

语法	`js_preread` *function* | *module.function*;
默认值	—
上下文	stream, server

设置将在预读取阶段 调用的 njs 函数。可以引用模块函数。

当流会话首次达到预读取阶段 时，该函数会被调用一次。该函数使用以下参数调用：

`s`	stream session 对象

在这个阶段，可以为每个接收到的数据块执行初始化或使用 s.on() 方法注册回调，直到调用以下方法之一：s.done()、s.decline()、s.allow()。当调用这些方法之一时，流会话将切换到下一个阶段，并且当前所有的 s.on() 回调将被丢弃。

> **备注**
>
> 因为 *js_preread* 处理器会立即返回其结果，所以它仅支持同步操作。因此，不支持像 ngx.fetch() 或 setTimeout() 这样的异步操作。不过，在预读阶段 的 s.on() 回调中支持异步操作。

js_set

语法	`js_set` *$variable function* | *module.function*;
默认值	—
上下文	stream, server

为指定变量设置 njs 函数。可以引用模块函数。

当第一次引用给定请求的变量时调用该函数。具体时机取决于引用变量的阶段。这可以用于执行一些与变量评估无关的逻辑。例如，如果变量仅在*log_format* 指令中引用，其处理器将在日志阶段之前不会被执行。这个处理器可以用于在请求释放之前进行一些清理。

> **备注**
>
> 因为 *js_set* 处理器会立即返回其结果，所以它仅支持同步操作。因此，不支持像 ngx.fetch() 或 setTimeout() 这样的异步操作。

js_shared_dict_zone

语法	`js_shared_dict_zone zone=`*name:size* [`timeout=`*time*] [`type=`*string* | *number*] [`evict`];
默认值	—
上下文	stream

设置共享内存区域的名称和大小，该区域在工作进程之间共享键值字典。

`type`	可选参数，允许将值类型重新定义为 `number`，默认情况下，共享字典使用 `string` 作为键和值
`timeout`	可选参数，设置从区域中移除所有共享字典条目的时间
`evict`	可选参数，当区域存储耗尽时，移除最旧的键值对

示例：

```
example.conf:
    # 创建一个1Mb的字典，值为字符串，
    # 在60秒不活动后移除键值对：
```

```
    js_shared_dict_zone zone=foo:1M timeout=60s;

    # 创建一个512Kb的字典，值为字符串，
    # 当区域耗尽时强制移除最旧的键值对：
    js_shared_dict_zone zone=bar:512K timeout=30s evict;

    # 创建一个32Kb的永久字典，值为数字：
    js_shared_dict_zone zone=num:32k type=number;
```

```
example.js:
    function get(r) {
        r.return(200, ngx.shared.foo.get(r.args.key));
    }

    function set(r) {
        r.return(200, ngx.shared.foo.set(r.args.key, r.args.value));
    }

    function delete(r) {
        r.return(200, ngx.shared.bar.delete(r.args.key));
    }

    function increment(r) {
        r.return(200, ngx.shared.num.incr(r.args.key, 2));
    }
```

js_var

语法	`js_var` *$variable* [*value*];
默认值	—
上下文	stream, server

声明一个 可写 变量。值可以包含文本、变量及其组合。

会话对象属性

每个流 njs 处理器接收一个参数，即 流会话 对象。

Limit Conn

该模块用于限制每个定义的键的连接数量，特别是来自单个 IP 地址的连接数量。

配置示例

```
stream {
    limit_conn_zone $binary_remote_addr zone=addr:10m;

    ...

    server {

        ...

        limit_conn           addr 1;
        limit_conn_log_level error;
    }
}
```

指令

limit_conn

语法	`limit_conn` *zone number*;
默认	—
上下文	stream, server

设置共享内存区域和给定键值的最大允许连接数。当超过此限制时，服务器将关闭连接。例如，指令

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

server {
    ...
    limit_conn addr 1;
}
```

仅允许每个 IP 地址同时有一个连接。

当指定多个 `limit_conn` 指令时，任何配置的限制将适用。

如果当前级别没有定义 `limit_conn` 指令，则这些指令将从前一个配置级别继承。

limit_conn_dry_run

语法	limit_conn_dry_run on | off;
默认	limit_conn_dry_run off;
上下文	stream, server

启用干运行模式。在此模式下，连接数量没有限制，但是在共享内存区域 中，过量连接的数量仍然会正常计算。

limit_conn_log_level

语法	limit_conn_log_level info | notice | warn | error;
默认	limit_conn_log_level error;
上下文	stream, server

设置服务器限制连接数量时所需的日志记录级别。

limit_conn_zone

语法	limit_conn_zone *key* zone = *name:size*;
默认	—
上下文	stream

设置共享内存区域的参数，该区域将保存各种键的状态。特别是，状态包括当前连接数。键可以包含文本、变量及其组合。具有空键值的连接不被计算。

使用示例：

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

这里，客户端 IP 地址由 $binary_remote_addr 变量设置。

对于 IPv4 地址，$binary_remote_addr 的大小为 4 字节，对于 IPv6 地址则为 16 字节。在 32 位平台上，存储的状态始终占用 32 或 64 字节，在 64 位平台上占用 64 字节。

一个兆字节的区域可以保存大约 32000 个 32 字节的状态或大约 16000 个 64 字节的状态。如果区域存储耗尽，服务器将关闭连接。

内置变量

$limit_conn_status

保存限制连接数量的结果：PASSED, REJECTED 或 REJECTED_DRY_RUN

Log

该模块以指定格式写入请求日志。

配置示例

```
log_format basic '$remote_addr [$time_local] '
                 '$protocol $status $bytes_sent $bytes_received '
                 '$session_time';

access_log /spool/logs/angie-access.log basic buffer=32k;
```

指令

access_log

语法	`access_log` *path* [*format* [`buffer=`*size*] [gzip[=*level*]] [`flush=`*time*] [`if=`*condition*]]; `access_log off;`
默认	`access_log off;`
上下文	stream, server

设置缓冲日志写入的 `path`、`format` 和配置。可以在同一配置级别上指定多个日志。通过在第一个参数中指定 `"syslog:"` 前缀，可以配置记录到*syslog*。特殊值 `off` 取消当前级别上的所有 `access_log` 指令。

如果使用了 `buffer` 或 `gzip` 参数，则日志写入将被缓冲。

小心

缓冲区大小不得超过对磁盘文件的原子写入大小。对于 FreeBSD，该大小是无限制的。

启用缓冲时，数据将写入文件：

- 如果下一行日志不适合缓冲区；
- 如果缓冲的数据比 flush 参数指定的时间更旧；
- 当工作进程重新打开日志文件 或关闭时。

如果使用了 *gzip* 参数，则缓冲的数据将在写入文件之前进行压缩。压缩级别可以设置在 *1*（最快，压缩较少）到 *9*（最慢，压缩效果最好）之间。默认情况下，缓冲区大小等于 *64K* 字节，压缩级别设置为 *1*。由于数据以原子块压缩，因此日志文件可以在任何时候通过 *"zcat"* 解压或读取。

示例：

```
access_log /path/to/log.gz basic gzip flush=5m;
```

重要

为了使 gzip 压缩生效，Angie 必须使用 zlib 库构建。

文件路径可以包含变量，但这类日志有一些限制：

- 使用工作进程凭据的*user* 应该具有在此类日志目录中创建文件的权限；
- 不支持缓冲写入；
- 每次日志写入时都会打开和关闭文件。然而，由于可以将频繁使用的文件的描述符存储在缓存中，因此在*open_log_file_cache* 指令的 *valid* 参数指定的时间内，可以继续写入旧文件。

if 参数启用条件日志记录。如果条件评估为 *"0"* 或空字符串，则不会记录该会话。

log_format

语法	`log_format` *name* [`escape=`default | `json` | `none`] *string* ...;
默认	—
上下文	stream

指定日志格式。

`escape` 参数允许在变量中设置 `json` 或 `default` 字符转义，默认使用 `default` 转义。`none` 值禁用转义。

对于 `default` 转义，字符""、"\" 以及值小于 32 或大于 126 的其他字符将被转义为"\xXX"。如果未找到变量值，则将记录一个连字符"-"。

对于 `json` 转义，所有在 JSON 字符串中不允许的字符都将被转义：字符""" 和"\" 被转义为"\"" 和"\\"，值小于 32 的字符被转义为"\n"、"\r"、"\t"、"\b"、"\f" 或"\u00XX"。

open_log_file_cache

语法	`open_log_file_cache max=`*N* [`inactive=`*time*] [`min_uses=`*N*] [`valid=`*time*]; `open_log_file_cache off;`
默认	`open_log_file_cache off;`
上下文	stream, server

定义一个缓存，用于存储包含变量名称的频繁使用日志的文件描述符。该指令具有以下参数：

`max`	设置缓存中描述符的最大数量；如果缓存已满，则关闭最少使用（LRU）描述符
`inactive`	设置缓存描述符在此时间内未被访问后关闭的时间；默认为 10 秒
`min_uses`	设置在 *inactive* 参数定义的时间内，文件使用的最小次数，以便让描述符保持在缓存中打开；默认为 1
`valid`	设置检查文件是否仍然以相同名称存在的时间；默认为 60 秒
`off`	禁用缓存

用法示例：

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

Map

创建的变量，其值依赖于其他变量的值。

配置示例

```
map $remote_addr $limit {
    127.0.0.1    "";
    default      $binary_remote_addr;
}

limit_conn_zone $limit zone=addr:10m;
limit_conn addr 1;
```

指令

map

语法	map *string $variable* { ... };
默认值	—
上下文	stream

创建一个新变量。其值依赖于第一个参数，该参数以包含变量的字符串形式指定，例如：

```
set $var1 "foo";
set $var2 "bar";

map $var1$var2 $new_variable {
    default "foobar_value";
}
```

在这里，变量 `$new_variable` 的值将由两个变量 `$var1` 和 `$var2` 组成，或者如果这些变量未定义，则使用默认值。

备注

由于变量仅在使用时才会被求值，即使声明了大量的"map" 变量也不会对请求处理增加额外的成本。

`map` 块内的参数指定源值与结果值之间的映射关系。

源值以字符串或正则表达式的形式指定。

字符串匹配时不区分大小写。

正则表达式应以 ~ 符号开头以进行区分大小写的匹配，或以 ~* 符号开头以进行不区分大小写的匹配。正则表达式可以包含命名和位置捕获，后者可以与结果变量一起在其他指令中使用。

如果源值与下面描述的特殊参数之一匹配，则应以 \ 符号作为前缀。

结果值可以包含文本、变量及其组合。

还支持以下特殊参数：

`default` *value*	如果源值与指定的变体都不匹配，则设置结果值。当未指定 *default* 时，默认结果值将为空字符串。
`hostnames`	指示源值可以是带有前缀或后缀掩码的主机名。此参数应在值列表之前指定。

例如，

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

以下两条记录

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

可以合并为：

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

`include` *file*	包含一个文件，其中包含值。可以有多个包含。
`volatile`	指示变量不可缓存。

如果源值与多个指定的变体匹配，例如掩码和正则表达式都匹配，则将选择第一个匹配的变体，优先级顺序如下：

1. 没有掩码的字符串值
2. 带前缀掩码的最长字符串值，例如 `*.example.com`
3. 带后缀掩码的最长字符串值，例如 `mail.*`
4. 第一个匹配的正则表达式（按在配置文件中的出现顺序）
5. 默认值（`default`）

map_hash_bucket_size

语法	`map_hash_bucket_size` *size*;
默认值	`map_hash_bucket_size 32|64|128;`
上下文	stream

设置*map* 变量哈希表的桶大小。默认值取决于处理器的缓存行大小。有关设置哈希表的详细信息，请参见单独。

map_hash_max_size

语法	`map_hash_max_size` *size*;
默认值	`map_hash_max_size 2048;`
上下文	stream

设置*map* 变量哈希表的最大大小。有关设置哈希表的详细信息，请参见单独。

MQTT Preread

启用从消息队列遥测传输（MQTT）版本的 CONNECT 数据包中提取客户端 ID 和用户名 3.1.1 和 5.0。

当从源代码 构建时，该模块必须通过 `--with-stream_mqtt_preread_module` 构建参数启用。在来自 我们的仓库的软件包和镜像中，该模块已包含在构建中。

配置示例

通过客户端 ID 选择组中的服务器：

```
stream {

    mqtt_preread on;

    upstream mqtt {
        hash $mqtt_preread_clientid;
        # ...
    }
}
```

指令

mqtt_preread

语法	`mqtt_preread on | off;`
默认	`mqtt_preread off;`
上下文	stream, server

控制在预读阶段 从 CONNECT 数据包中提取信息。如果参数启用（on），将在指定的上下文中填充下面列出的变量。

内置变量

有关值语义的详细描述，请参阅 MQTT 协议规范版本 3.1.1 和 5.0。

$mqtt_preread_clientid

唯一客户端标识符。

$mqtt_preread_username

可选用户名。

Pass

允许将已接受的连接直接传递到在*HTTP*、*Stream* 或*Mail* 模块中配置的任何监听套接字。

配置示例

在 stream 模块处理 SSL/TLS 终止后，连接被转发到 http 模块：

```
http {
    server {
        listen 8000;

        location / {
            root html;
        }
    }
}

stream {
    server {
        listen 12345 ssl;

        ssl_certificate     domain.crt;
        ssl_certificate_key domain.key;

        pass 127.0.0.1:8000;
    }
}
```

指令

pass

Syntax	pass *address*;
默认	—
Context	server

该指令设置客户端连接应传递到的服务器地址。*address* 可以作为 IP 地址和端口给出：

```
pass 127.0.0.1:12345;
```

或者作为 UNIX 域套接字的路径：

```
pass unix:/tmp/stream.socket;
```

此外，*address* 也可以用变量设置：

```
pass $upstream;
```

Proxy

允许通过 TCP、UDP 和 UNIX 域套接字代理数据流。

配置示例

```
server {
    listen 127.0.0.1:12345;
    proxy_pass 127.0.0.1:8080;
}

server {
    listen 12345;
    proxy_connect_timeout 1s;
    proxy_timeout 1m;
    proxy_pass example.com:12345;
}

server {
    listen 53 udp reuseport;
    proxy_timeout 20s;
    proxy_pass dns.example.com:53;
}

server {
    listen [::1]:12345;
```

```
    proxy_pass unix:/tmp/stream.socket;
}
```

指令

proxy_bind

语法	`proxy_bind` *address* [`transparent`] | `off`;
默认值	—
上下文	stream, server

使发往被代理服务器的出站连接从指定的本地 IP 地址发起。参数值可以包含变量。特殊值 `off` 取消从先前配置级别继承的 *proxy_bind* 指令的效果，从而允许系统自动分配本地 IP 地址。

`transparent` 参数允许从非本地 IP 地址发起出站连接，例如，从客户端的真实 IP 地址：

```
proxy_bind $remote_addr transparent;
```

为了使此参数生效，Angie 工作进程通常需要以超级用户 权限运行。在 Linux 上，这不是必需的：如果指定了 `transparent` 参数，工作进程将从主进程继承 *CAP_NET_RAW* 权限。

重要

内核路由表也应该配置为拦截来自被代理服务器的网络流量。

proxy_buffer_size

语法	`proxy_buffer_size` *size*;
默认值	`proxy_buffer_size 16k;`
上下文	stream, server

设置用于从被代理服务器读取数据的缓冲区大小。还设置用于从客户端读取数据的缓冲区大小。

proxy_connect_timeout

语法	`proxy_connect_timeout` *time*;
默认值	`proxy_connect_timeout 60s;`
上下文	stream, server

定义与被代理服务器建立连接的超时时间。

proxy_connection_drop

语法	proxy_connection_drop *time* | on | off;
默认值	proxy_connection_drop off;
上下文	stream, server

在被代理服务器从组中移除或被*reresolve* 进程或*API* 命令 DELETE 标记为永久不可用后，启用终止与该服务器的所有会话。

当为客户端或被代理服务器处理下一个读或写事件时，会话被终止。

设置 *time* 启用会话终止超时；如果设置为 on，会话将立即被丢弃。

proxy_download_rate

语法	proxy_download_rate *rate*;
默认值	proxy_download_rate 0;
上下文	stream, server

限制从被代理服务器读取数据的速度。rate 以每秒字节为单位指定。

0	禁用速率限制

备注

限制是针对每个连接设置的，因此如果 Angie 同时打开两个连接到被代理服务器，则整体速率将是指定限制的两倍。

参数值可以包含变量。在需要根据某种条件限制速率的情况下，这可能会很有用：

```
map $slow $rate {
    1     4k;
    2     8k;
}

proxy_download_rate $rate;
```

proxy_half_close

语法	proxy_half_close on | off;
默认值	proxy_half_close off;
上下文	stream, server

启用或禁用独立关闭 TCP 连接的每个方向（"TCP 半关闭"）。如果启用，TCP 的代理将保持，直到双方都关闭连接。

proxy_next_upstream

语法	`proxy_next_upstream on | off;`
默认值	`proxy_next_upstream on;`
上下文	stream, server

当无法建立与被代理服务器的连接时，决定是否将客户端连接传递给上游池 中的下一个服务器。

将连接传递给下一个服务器可以受到尝试次数 和时间 的限制。

proxy_next_upstream_timeout

语法	`proxy_next_upstream_timeout` *time*;
默认值	`proxy_next_upstream_timeout 0;`
上下文	stream, server

限制传递连接到下一个 服务器的时间。

`0`	关闭此限制

proxy_next_upstream_tries

语法	`proxy_next_upstream_tries` *number*;
默认值	`proxy_next_upstream_tries 0;`
上下文	stream, server

限制将连接传递给下一个 服务器的可能尝试次数。

`0`	关闭此限制

proxy_pass

语法	`proxy_pass` *address*;
默认值	—
上下文	server

设置被代理服务器的地址。`address` 可以指定为域名或 IP 地址，以及端口：

```
proxy_pass localhost:12345;
```

或作为 UNIX 域套接字路径：

```
proxy_pass unix:/tmp/stream.socket;
```

如果域名解析为多个地址，则将以轮询的方式使用所有这些地址。此外，地址可以指定为服务器组。

地址也可以使用变量指定：

```
proxy_pass $upstream;
```

在这种情况下，服务器名称将在描述的服务器组 中进行搜索，如果未找到，则通过*resolver* 确定。

proxy_protocol

语法	`proxy_protocol on | off;`
默认值	`proxy_protocol off;`
上下文	stream, server

启用与被代理服务器连接的 PROXY 协议。

proxy_requests

语法	`proxy_requests` *number*`;`
默认值	`proxy_requests 0;`
上下文	stream, server

设置在丢弃客户端与现有 UDP 流会话之间的绑定时所需的客户端数据报数量。在接收到指定数量的数据报后，来自同一客户端的下一个数据报将启动一个新会话。当所有客户端数据报传输到被代理服务器并收到预期的响应数量 时，会话终止，或者当达到超时 时。

proxy_responses

语法	`proxy_responses` *number*`;`
默认值	—
上下文	stream, server

设置在使用*UDP* 协议时，预期从被代理服务器收到的客户端数据报数量。该数量作为会话终止的提示。默认情况下，数据报的数量没有限制。

如果指定零值，则不期望任何响应。然而，如果收到响应且会话仍未结束，则将处理该响应。

proxy_socket_keepalive

语法	`proxy_socket_keepalive on	off;`
默认值	`proxy_socket_keepalive off;`
上下文	stream, server

配置与被代理服务器的出站连接的"TCP keepalive" 行为。

`off`	默认情况下，套接字使用操作系统的设置。
`on`	为套接字开启 SO_KEEPALIVE 套接字选项。

proxy_ssl

语法	`proxy_ssl on	off;`
默认值	`proxy_ssl off;`
上下文	stream, server

启用与被代理服务器的 SSL/TLS 协议。

proxy_ssl_certificate

语法	`proxy_ssl_certificate` *file* [*file*];
默认值	—
上下文	stream, server

指定用于与被代理服务器进行身份验证的 PEM 格式证书文件。文件名中可以使用变量。

Added in version 1.2.0.

当启用*proxy_ssl_ntls* 时，该指令接受两个参数，而不是一个：

```
server {
    proxy_ssl_ntls  on;

    proxy_ssl_certificate      sign.crt enc.crt;
    proxy_ssl_certificate_key  sign.key enc.key;

    proxy_ssl_ciphers "ECC-SM2-WITH-SM4-SM3:ECDHE-SM2-WITH-SM4-SM3:RSA";

    proxy_pass backend:12345;
}
```

proxy_ssl_certificate_key

语法	`proxy_ssl_certificate_key` *file* [*file*];
默认值	—
上下文	stream, server

指定用于与被代理服务器进行身份验证的 PEM 格式私钥文件。文件名中可以使用变量。

Added in version 1.2.0.

当启用*proxy_ssl_ntls* 时，该指令接受两个参数，而不是一个：

```
server {
    proxy_ssl_ntls  on;

    proxy_ssl_certificate      sign.crt enc.crt;
    proxy_ssl_certificate_key  sign.key enc.key;

    proxy_ssl_ciphers "ECC-SM2-WITH-SM4-SM3:ECDHE-SM2-WITH-SM4-SM3:RSA";

    proxy_pass backend:12345;
}
```

proxy_ssl_ciphers

语法	`proxy_ssl_ciphers` *ciphers*;
默认值	`proxy_ssl_ciphers DEFAULT;`
上下文	stream, server

指定对代理服务器请求启用的密码。密码以 OpenSSL 库可理解的格式指定。

密码列表取决于已安装的 OpenSSL 版本。可以使用 `openssl ciphers` 命令查看完整列表。

注意

当使用 OpenSSL 时，`proxy_ssl_ciphers` 指令 * 不会 * 配置 TLS 1.3 的加密套件。要配置 OpenSSL 的 TLS 1.3 加密套件，请使用*proxy_ssl_conf_command* 指令，该指令是为高级 SSL 配置而添加的。

- 在 LibreSSL 中，可以使用 `proxy_ssl_ciphers` 配置 TLS 1.3 加密套件。
- 在 BoringSSL 中，无法配置 TLS 1.3 加密套件。

proxy_ssl_conf_command

语法	`proxy_ssl_conf_command` *name value*;
默认值	—
上下文	stream, server

在与代理服务器建立连接时设置任意 OpenSSL 配置 命令。

重要

该指令在使用 OpenSSL 1.0.2 或更高版本时受支持。要在 OpenSSL 中配置 TLS 1.3 密码套件，请使用 `ciphersuites` 命令。

可以在同一级别上指定多个 *proxy_ssl_conf_command* 指令。这些指令仅在当前级别没有定义 *proxy_ssl_conf_command* 指令时才从上一个配置级别继承。

小心

请注意，直接配置 OpenSSL 可能会导致意外行为。

proxy_ssl_crl

语法	`proxy_ssl_crl` *file*;
默认值	—
上下文	stream, server

指定 PEM 格式的已撤销证书文件 (CRL)，用于验证 代理服务器的证书。

proxy_ssl_name

语法	`proxy_ssl_name` *name*;
默认值	`proxy_ssl_name` *host from* `proxy_pass`;
上下文	stream, server

允许覆盖用于验证 代理服务器证书的服务器名称，并在与代理服务器建立连接时通过*SNI* 传递。服务器名称也可以使用变量指定。

默认情况下，使用*proxy_pass* 指令指定的地址中的主机名。

proxy_ssl_ntls

Added in version 1.2.0.

语法	proxy_ssl_ntls on | off;
默认值	proxy_ssl_ntls off;
上下文	stream, server

在使用 TongSuo TLS 库时启用客户端对 NTLS 的支持。

```
server {
    proxy_ssl_ntls  on;

    proxy_ssl_certificate      sign.crt enc.crt;
    proxy_ssl_certificate_key  sign.key enc.key;

    proxy_ssl_ciphers "ECC-SM2-WITH-SM4-SM3:ECDHE-SM2-WITH-SM4-SM3:RSA";

    proxy_pass backend:12345;
}
```

重要

Angie 必须使用 *--with-ntls* 配置参数构建，并链接支持 NTLS 的相应 SSL 库

```
./configure --with-openssl=../Tongsuo-8.3.0 \
            --with-openssl-opt=enable-ntls  \
            --with-ntls
```

proxy_ssl_password_file

语法	proxy_ssl_password_file *file*;
默认值	—
上下文	stream, server

指定包含密钥 密码的文件，每个密码单独一行。加载密钥时会依次尝试这些密码。

proxy_ssl_protocols

语法	proxy_ssl_protocols [SSLv2] [SSLv3] [TLSv1] [TLSv1.1] [TLSv1.2] [TLSv1.3];
默认值	proxy_ssl_protocols TLSv1.2 TLSv1.3;
上下文	stream, server

在 1.2.0 版本发生变更: 默认设置中添加了 `TLSv1.3` 参数。

为代理服务器请求启用指定的协议。

proxy_ssl_server_name

语法	`proxy_ssl_server_name on	off;`
默认值	`proxy_ssl_server_name off;`
上下文	stream, server

启用或禁用在与代理服务器建立连接时通过 服务器名称指示 TLS 扩展 (SNI, RFC 6066) 传递由*proxy_ssl_name* 指令指定的服务器名称。

proxy_ssl_session_reuse

语法	`proxy_ssl_session_reuse on	off;`
默认值	`proxy_ssl_session_reuse on;`
上下文	stream, server

确定在与代理服务器通信时是否可以重用 SSL 会话。如果日志中出现"*SSL3_GET_FINISHED:digest check failed*" 错误，请尝试禁用会话重用。

proxy_ssl_trusted_certificate

语法	`proxy_ssl_trusted_certificate` *file*;
默认值	—
上下文	stream, server

指定 PEM 格式的受信任 CA 证书文件，用于验证 代理服务器的证书。

proxy_ssl_verify

语法	`proxy_ssl_verify on	off;`
默认值	`proxy_ssl_verify off;`
上下文	stream, server

启用或禁用对代理服务器证书的验证。

proxy_ssl_verify_depth

语法	`proxy_ssl_verify_depth` *number*;
默认值	`proxy_ssl_verify_depth 1;`
上下文	stream, server

设置代理服务器证书链中的验证深度。

proxy_timeout

语法	`proxy_timeout` *time*;
默认值	`proxy_timeout 10m;`
上下文	stream, server

设置客户端或代理服务器连接上两个连续读或写操作之间的超时时间。如果在此时间内没有传输数据，连接将被关闭。

upstream_probe_timeout (PRO)

Added in version 1.4.0: PRO

语法	`upstream_probe_timeout` *time*;
默认值	`upstream_probe_timeout 50s;`
上下文	server

为使用*upstream_probe (PRO)*指令配置的探测设置已建立服务器连接的最大不活动时间；如果超过此限制，连接将被关闭。

proxy_upload_rate

语法	`proxy_upload_rate` *rate*;
默认值	`proxy_upload_rate 0;`
上下文	stream, server

限制从客户端读取数据的速度。速率以每秒字节数指定。

0	禁用速率限制

备注

限制是针对每个连接设置的，所以如果客户端同时打开两个连接，总体速率将是指定限制的两倍。

参数值可以包含变量。这在需要根据特定条件限制速率的情况下很有用：

```
map $slow $rate {
    1     4k;
    2     8k;
}

proxy_upload_rate $rate;
```

RDP Preread

在使用 RDP 协议时，此模块允许在做出负载均衡决策之前提取用于会话识别和管理的 Cookie。

当从源代码 构建时，需要通过 `--with-stream_rdp_preread_module` 构建选项来启用此模块。在 我们的仓库中的包和镜像中，该模块已包含在构建中。

配置示例

绑定到发放 Cookie 的服务器

这使用了*sticky* 指令的 `learn` 模式：

```
stream {

    rdp_preread on;

    upstream rdp {

        server 127.0.0.1:3390 sid=a;
        server 127.0.0.1:3391 sid=b;

        sticky learn lookup=$rdp_cookie create=$rdp_cookie zone=sessions:1m;
    }
}
```

指令

rdp_preread

语法	`rdp_preread on | off;`
默认值	`rdp_preread off;`
上下文	stream, server

控制在预读阶段 从 RDP 协议 Cookie 中提取信息。如果设置为 `on`，将在指定的上下文中填充下面列出的变量。

内置变量

Cookie 值的语义取决于 RDP 协议版本。

`$rdp_cookie`

整个 Cookie 值。

`$rdp_cookie_<name>`

具有指定名称的 Cookie 字段的值。

RealIP

允许将客户端地址和端口更改为在 PROXY 协议头中传递的地址和端口。必须通过在*listen* 指令中设置 `proxy_protocol` 参数来预先启用 PROXY 协议。

当从源代码 构建时，默认情况下不会构建此模块；应使用 `--with-stream_realip_module` 构建选项来启用它。

在 我们的仓库中的包和镜像中，该模块已包含在构建中。

配置示例

```
listen 12345 proxy_protocol;

set_real_ip_from  192.168.1.0/24;
set_real_ip_from  192.168.2.1;
set_real_ip_from  2001:0db8::/32;
```

指令

set_real_ip_from

语法	`set_real_ip_from` *address* | *CIDR* | `unix:`;
默认值	—
上下文	stream, server

定义已知发送正确替换地址的可信地址。如果指定了特殊值 `unix:`，则所有 UNIX 域套接字将被信任。

内置变量

`$realip_remote_addr`

保留原始客户端地址

`$realip_remote_port`

保留原始客户端端口

Return

该模块允许向客户端发送指定的值，然后关闭连接。

配置示例

```
server {
    listen 12345;
    return $time_iso8601;
}
```

指令

return

语法	`return` *value*;
默认值	—
上下文	server

指定要发送给客户端的值。该值可以包含文本、变量及其组合。

Set

该模块允许为变量设置一个值。

配置示例

```
server {
    listen 12345;
    set    $true 1;
}
```

指令

set

语法	`set` *$variable value*;
默认值	—
上下文	server

为指定的变量设置一个值。该值可以包含文本、变量及其组合。

Split Clients

该模块生成用于 A/B 测试、金丝雀发布或其他需要将一定比例的客户端路由到一个服务器或配置的场景，同时将剩余部分引导到其他地方。

配置示例

```
stream {
    # ...
    split_clients "${remote_addr}AAA" $upstream {
                  0.5%                feature_test1;
                  2.0%                feature_test2;
                  *                   production;
    }

    server {
        # ...
        proxy_pass $upstream;
    }
}
```

指令

split_clients

Syntax	split_clients *string* *$variable* { ... }
默认值	—
Context	stream

通过对 *string* 进行哈希来创建一个 *$variable*；*string* 中的变量被替换，结果被哈希，然后哈希值用于选择 *$variable* 的字符串值。

哈希函数使用 MurmurHash2 （32 位），其整个值范围（0 到 4294967295）按出现顺序映射到桶；百分比决定了桶的大小。通配符 (*) 可以出现在最后；不属于其他桶的哈希值会被映射到其分配的值。

示例：

```
split_clients "${remote_addr}AAA" $variant {
              0.5%                .one;
              2.0%                .two;
              *                   "";
}
```

这里，在 $*remote_addr*AAA 字符串中替换变量后，哈希值分布如下：

- 值为 0 到 21474835（0.5%）产生 .one
- 值为 21474836 到 107374180（2%）产生 .two

- 值为 107374181 到 4294967295（所有其他值）产生 "" （一个空字符串）

SSL

提供了流代理服务器与 SSL/TLS 协议配合工作的必要支持。

当 从源代码构建时，此模块默认不构建；应使用 `--with-stream_ssl_module` 构建选项启用它。

在 我们的仓库中的包和镜像中，该模块包含在构建中。

重要

此模块需要 OpenSSL 库。

配置示例

为了减少处理器负载，建议

- 将工作进程 的数量设置为处理器的数量，
- 启用共享 会话缓存，
- 禁用内置 会话缓存，
- 并可能增加会话的生命周期 （默认值为 5 分钟）：

```
worker_processes auto;

stream {

    #...

    server {
        listen              12345 ssl;

        ssl_protocols       TLSv1.2 TLSv1.3;
        ssl_ciphers         AES128-SHA:AES256-SHA:RC4-SHA:DES-CBC3-SHA:RC4-MD5;
        ssl_certificate     /usr/local/angie/conf/cert.pem;
        ssl_certificate_key /usr/local/angie/conf/cert.key;
        ssl_session_cache   shared:SSL:10m;
        ssl_session_timeout 10m;

    #    ...
    }
```

指令

ssl_alpn

语法	ssl_alpn *protocol* ...;
默认	—
上下文	stream, server

指定支持的 ALPN 协议列表。如果客户端使用 ALPN，必须协商其中一个协议：

```
map $ssl_alpn_protocol $proxy {
    h2                  127.0.0.1:8001;
    http/1.1            127.0.0.1:8002;
}

server {
    listen      12346;
    proxy_pass  $proxy;
    ssl_alpn    h2 http/1.1;
}
```

ssl_certificate

语法	ssl_certificate *file* [*file*];
默认	—
上下文	stream, server

指定给定服务器的 PEM 格式的证书文件。如果除了主证书外还需要指定中间证书，应按以下顺序在同一文件中指定：主证书在前，然后是中间证书。PEM 格式的私钥可以放在同一文件中。

此指令可以多次指定，以加载不同类型的证书，例如 RSA 和 ECDSA：

```
server {
    listen              12345 ssl;

    ssl_certificate     example.com.rsa.crt;
    ssl_certificate_key example.com.rsa.key;

    ssl_certificate     example.com.ecdsa.crt;
    ssl_certificate_key example.com.ecdsa.key;

#    ...
}
```

只有 OpenSSL 1.0.2 或更高版本支持不同证书的独立证书链。使用较旧版本时，只能使用一个证书链。

重要

在使用 OpenSSL 1.0.2 或更高版本时，文件名中可以使用变量：

```
ssl_certificate     $ssl_server_name.crt;
ssl_certificate_key $ssl_server_name.key;
```

请注意，使用变量意味着在每次 SSL 握手时都会加载证书，这可能会对性能产生负面影响。

可以指定值"data:*$variable*" 代替 `file`，这样可以从变量中加载证书，而不使用中间文件。

请注意，不当使用此语法可能会有安全隐患，例如将私钥数据写入错误日志。

Added in version 1.2.0.

当*ssl_ntls* 启用时，该指令接受两个参数而不是一个：证书的签名部分和加密部分：

```
listen ... ssl;

ssl_ntls  on;

# dual NTLS certificate
ssl_certificate      sign.crt enc.crt;
ssl_certificate_key  sign.key enc.key;

# 可以与常规RSA证书结合使用：
ssl_certificate  rsa.crt;
ssl_certificate  rsa.key;
```

ssl_certificate_key

语法	`ssl_certificate_key` *file* [*file*];
默认	—
上下文	stream, server

指定给定服务器的 PEM 格式的私钥文件。

重要

在使用 OpenSSL 1.0.2 或更高版本时，文件名中可以使用变量。

可以指定值 `engine:`name`:id` 代替 `file`，这将从 OpenSSL 引擎 *name* 加载指定 *id* 的私钥。

可以指定值"data:*$variable*" 代替 `file`，这将从变量中加载私钥，而不使用中间文件。请注意，不当使用此语法可能会有安全隐患，例如将私钥数据写入错误日志。

Added in version 1.2.0.

当*ssl_ntls* 启用时，该指令接受两个参数而不是一个：密钥的签名部分和加密部分：

```
listen ... ssl;

ssl_ntls  on;

# dual NTLS certificate
ssl_certificate      sign.crt enc.crt;
ssl_certificate_key  sign.key enc.key;

# 可以与常规RSA证书结合使用：
ssl_certificate  rsa.crt;
ssl_certificate  rsa.key;
```

ssl_ciphers

语法	`ssl_ciphers` *ciphers*;
默认	`ssl_ciphers HIGH:!aNULL:!MD5;`
上下文	stream, server

指定启用的密码。密码以 OpenSSL 库理解的格式指定，例如：

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

密码列表取决于已安装的 OpenSSL 版本。可以使用 `openssl ciphers` 命令查看完整列表。

注意

当使用 OpenSSL 时，`ssl_ciphers` 指令 * 不会 * 配置 TLS 1.3 的加密套件。如需配置 TLS 1.3 的加密套件，请使用*ssl_conf_command* 指令，它是为支持高级 SSL 配置而添加的。

- 在 LibreSSL 中，可以使用 `ssl_ciphers` 配置 TLS 1.3 加密套件。
- 在 BoringSSL 中，无法配置 TLS 1.3 加密套件。

ssl_client_certificate

语法	`ssl_client_certificate` *file*;
默认	—
上下文	stream, server

指定用于验证 客户端证书和 OCSP 响应的 PEM 格式的受信任 CA 证书文件，如果*ssl_stapling* 被启用。

证书列表将发送给客户端。如果不希望这样，可以使用*ssl_trusted_certificate* 指令。

ssl_conf_command

语法	`ssl_conf_command` *name value*;
默认	—
上下文	stream, server

设置任意 OpenSSL 配置 命令。

重要

当使用 OpenSSL 1.0.2 或更高版本时支持该指令。要在 OpenSSL 中配置 TLS 1.3 密码套件，请使用 `ciphersuites` 命令。

可以在同一层级上指定多个 *ssl_conf_command* 指令：

```
ssl_conf_command Options PrioritizeChaCha;
ssl_conf_command Ciphersuites TLS_CHACHA20_POLY1305_SHA256;
```

如果当前级别上没有定义 *ssl_conf_command* 指令，则这些指令会从上一个配置级别继承。

小心

请注意，直接配置 OpenSSL 可能会导致意外行为。

ssl_crl

语法	`ssl_crl` *file*;
默认	—
上下文	stream, server

指定 PEM 格式的被吊销证书文件（CRL），用于验证 客户端证书。

ssl_dhparam

语法	`ssl_dhparam` *file*;
默认	—
上下文	stream, server

指定用于 DHE 密码的 DH 参数文件。

小心

默认情况下未设置任何参数，因此不会使用 DHE 密码。

ssl_early_data

语法	`ssl_early_data on	off;`
默认	`ssl_early_data off;`
上下文	stream, server

启用或禁用 TLS 1.3 早期数据。

重要

在使用 OpenSSL 1.1.1 或更高版本以及 BoringSSL 时支持该指令。

ssl_ecdh_curve

语法	`ssl_ecdh_curve` *curve*;
默认	`ssl_ecdh_curve auto;`
上下文	stream, server

指定 ECDHE 密码的曲线。

重要

在使用 OpenSSL 1.0.2 或更高版本时，可以指定多个曲线，例如：

```
ssl_ecdh_curve prime256v1:secp384r1;
```

特殊值 `auto` 指示 Angie 在使用 OpenSSL 1.0.2 或更高版本时使用 OpenSSL 库内置的曲线列表，或在较旧版本中使用 prime256v1。

重要

在使用 OpenSSL 1.0.2 或更高版本时，此指令设置服务器支持的曲线列表。因此，为了使 ECDSA 证书正常工作，必须包含证书中使用的曲线。

ssl_handshake_timeout

语法	`ssl_handshake_timeout` *time*;
默认	`ssl_handshake_timeout 60s;`
上下文	stream, server

指定 SSL 握手完成的超时时间。

ssl_ocsp

语法	`ssl_ocsp on` | `off` | `leaf`;
默认	`ssl_ocsp off;`
上下文	http, server

启用客户端证书链的 OCSP 验证。`leaf` 参数仅启用客户端证书的验证。

为了使 OCSP 验证工作，*ssl_verify_client* 指令应设置为 `on` 或 `optional`。

为了解析 OCSP 响应者主机名，还应指定*resolver* 指令。

示例：

```
ssl_verify_client on;
ssl_ocsp          on;
resolver          127.0.0.53;
```

ssl_ocsp_cache

语法	`ssl_ocsp_cache off` | [shared:*name*:*size*];
默认	`ssl_ocsp_cache off;`
上下文	http, server

设置存储客户端证书状态以进行 OCSP 验证的缓存的名称和大小。该缓存在所有工作进程之间共享。可以在多个虚拟服务器中使用具有相同名称的缓存。

`off` 参数禁止使用缓存。

ssl_ocsp_responder

语法	`ssl_ocsp_responder` *uri*;
默认	—
上下文	http, server

覆盖在*validation* 客户端证书验证中指定的 "Authority Information Access" 证书扩展的 OCSP 响应者 URI。

仅支持 `http://` OCSP 响应者：

```
ssl_ocsp_responder http://ocsp.example.com/;
```

ssl_ntls

Added in version 1.2.0.

Syntax	`ssl_ntls on	off;`
Default	`ssl_ntls off;`
Context	stream, server

启用使用 TongSuo 库的服务器端 NTLS 支持。

```
listen ... ssl;
ssl_ntls  on;
```

重要

使用 *--with-ntls* 构建选项构建 Angie，并与启用 NTLS 的 SSL 库链接。

```
./configure --with-openssl=../Tongsuo-8.3.0 \
            --with-openssl-opt=enable-ntls  \
            --with-ntls
```

ssl_password_file

Syntax	`ssl_password_file` *file*;
Default	—
Context	stream, server

指定一个包含*secret keys* 密码的文件，每个密码在单独的一行上。加载密钥时，依次尝试这些密码。

示例：

```
stream {
    ssl_password_file /etc/keys/global.pass;
    ...

    server {
        listen 127.0.0.1:12345;
        ssl_certificate_key /etc/keys/first.key;
```

```
    }

    server {
        listen 127.0.0.1:12346;

        # 可以使用命名管道代替文件
        ssl_password_file /etc/keys/fifo;
        ssl_certificate_key /etc/keys/second.key;
    }
}
```

ssl_prefer_server_ciphers

Syntax	`ssl_prefer_server_ciphers on	off;`
Default	`ssl_prefer_server_ciphers off;`
Context	stream, server

指定在使用 SSLv3 和 TLS 协议时应优先使用服务器密码而不是客户端密码。

ssl_protocols

Syntax	`ssl_protocols` [SSLv2] [SSLv3] [TLSv1] [TLSv1.1] [TLSv1.2] [TLSv1.3];
默认值	`ssl_protocols TLSv1.2 TLSv1.3;`
上下文	stream, server

在 1.2.0 版本发生变更: `TLSv1.3` 参数已添加到默认集合。

启用指定的协议。

重要

TLSv1.1 和 TLSv1.2 参数仅在使用 OpenSSL 1.0.1 或更高版本时有效。

TLSv1.3 参数仅在使用 OpenSSL 1.1.1 或更高版本时有效。

ssl_session_cache

Syntax	`ssl_session_cache off	none |` [builtin[:*size*]] [shared:*name*:*size*];
Default	`ssl_session_cache none;`
Context	stream, server

设置存储会话参数的缓存类型和大小。缓存可以是以下类型之一：

`off`	严格禁止使用会话缓存：Angie 明确告诉客户端会话不能被重用。
`none`	温和地不允许使用会话缓存：Angie 告诉客户端会话可以重用，但实际上不在缓存中存储会话参数。
`builtin`	OpenSSL 内置的缓存；仅由一个工作进程使用。缓存大小以会话为单位指定。如果未给定大小，则等于 20480 会话。使用内置缓存可能会导致内存碎片。
`shared`	在所有工作进程之间共享的缓存。缓存大小以字节为单位指定；一兆字节大约可以存储 4000 会话。每个共享缓存应具有任意名称。具有相同名称的缓存可以在多个服务器中使用。它还用于自动生成、存储和定期轮换 TLS 会话票据密钥，除非使用*ssl_session_ticket_key* 指令明确配置。

可以同时使用两种缓存类型，例如：

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

但仅使用共享缓存而不使用内置缓存应更有效。

ssl_session_ticket_key

Syntax	`ssl_session_ticket_key` *file*;
Default	—
Context	stream, server

设置用于加密和解密 TLS 会话票据的秘密密钥文件。如果必须在多个服务器之间共享相同的密钥，则该指令是必要的。默认情况下，使用随机生成的密钥。

如果指定多个密钥，则仅使用第一个密钥加密 TLS 会话票据。这允许配置密钥轮换，例如：

```
ssl_session_ticket_key current.key;
ssl_session_ticket_key previous.key;
```

该文件必须包含 80 或 48 字节的随机数据，可以使用以下命令创建：

```
openssl rand 80 > ticket.key
```

根据文件大小，使用 AES256（对于 80 字节密钥）或 AES128（对于 48 字节密钥）进行加密。

ssl_session_tickets

Syntax	`ssl_session_tickets on` | `off`;
Default	`ssl_session_tickets on`;
Context	stream, server

启用或禁用通过 TLS 会话票据 的会话恢复。

ssl_session_timeout

Syntax	`ssl_session_timeout` *time*;
Default	`ssl_session_timeout 5m;`
Context	stream, server

指定客户端可以重用会话参数的时间。

ssl_stapling

Syntax	`ssl_stapling on | off;`
Default	`ssl_stapling off;`
Context	http, server

启用或禁用服务器的 OCSP 响应 stapling。示例：

```
ssl_stapling on;
resolver 127.0.0.53;
```

为了使 OCSP stapling 工作，服务器证书颁发者的证书应是已知的。如果*ssl_certificate* 文件不包含中间证书，则服务器证书颁发者的证书应存在于*ssl_trusted_certificate* 文件中。

> **注意**
>
> 对于 OCSP 响应者主机名的解析，还应指定*resolver* 指令。

ssl_stapling_file

Syntax	`ssl_stapling_file` *file*;
Default	—
Context	http, server

设置时，粘贴的 OCSP 响应将来自指定文件，而不是查询服务器证书中指定的 OCSP 响应者。

该文件应为 DER 格式，由 `openssl ocsp` 命令生成。

ssl_stapling_responder

Syntax	`ssl_stapling_responder uri;`
Default	—
Context	http, server

覆盖在 "Authority Information Access" 证书扩展中指定的 OCSP 响应者 URI。

仅支持 `http://` OCSP 响应者：

```
ssl_stapling_responder http://ocsp.example.com/;
```

ssl_stapling_verify

Syntax	`ssl_stapling_verify on	off;`
Default	`ssl_stapling_verify off;`
Context	http, server

启用或禁用服务器对 OCSP 响应的验证。

为了使验证工作，服务器证书颁发者的证书、根证书和所有中间证书应使用*ssl_trusted_certificate* 指令配置为信任。

ssl_trusted_certificate

Syntax	`ssl_trusted_certificate` *file*;
Default	—
Context	stream, server

指定一个包含 PEM 格式的受信任 CA 证书的文件，用于*verify* 客户端证书。

与*ssl_client_certificate* 指定的证书相比，这些证书的列表不会发送给客户端。

ssl_verify_client

Syntax	`ssl_verify_client on	off	optional	optional_no_ca;`
Default	`ssl_verify_client off;`
Context	stream, server

启用客户端证书的验证。验证结果存储在*$ssl_client_verify* 变量中。如果在客户端证书验证过程中发生错误，或者客户端未提供所需证书，则连接将关闭。

`optional`	请求客户端证书，并在证书存在时进行验证。
`optional_no_ca`	请求客户端证书，但不要求其由受信任的 CA 证书签名。这适用于 Angie 外部的服务执行实际证书验证的情况。

ssl_verify_depth

语法	`ssl_verify_depth` 数字;
默认	`ssl_verify_depth 1;`
上下文	stream, server

设置客户端证书链的验证深度。

内置变量

`stream_ssl` 模块支持以下变量：

$ssl_alpn_protocol

返回在 SSL 握手期间由 ALPN 选择的协议，或者在其他情况下返回空字符串。

$ssl_cipher

返回已建立 SSL 连接所使用的密码名称。

$ssl_ciphers

返回客户端支持的密码列表。已知的密码以名称列出，未知的以十六进制显示，例如：

AES128-SHA:AES256-SHA:0x00ff

重要

该变量仅在使用 OpenSSL 版本 1.0.2 或更高版本时完全支持。对于旧版本，该变量仅在新会话中可用，并且仅列出已知的密码。

$ssl_client_cert

返回已建立 SSL 连接的客户端证书，采用 PEM 格式，除第一行外，每行前面都加上制表符。

$ssl_client_fingerprint

返回已建立 SSL 连接的客户端证书的 SHA1 指纹。

$ssl_client_i_dn

返回已建立 SSL 连接的客户端证书的“颁发者 DN”字符串，符合 RFC 2253。

`$ssl_client_raw_cert`

返回已建立 SSL 连接的客户端证书，采用 PEM 格式。

`$ssl_client_s_dn`

返回已建立 SSL 连接的客户端证书的“主题 DN”字符串，符合 RFC 2253。

`$ssl_client_serial`

返回已建立 SSL 连接的客户端证书的序列号。

`$ssl_client_v_end`

返回客户端证书的到期日期。

`$ssl_client_v_remain`

返回客户端证书到期的天数。

`$ssl_client_v_start`

返回客户端证书的开始日期。

`$ssl_client_verify`

返回客户端证书验证的结果：`SUCCESS`、`FAILED:reason` 和 `NONE`（如果证书不存在）。

`$ssl_curve`

返回用于 SSL 握手密钥交换过程的协商曲线。已知的曲线以名称列出，未知的以十六进制显示，例如：

```
prime256v1
```

重要

该变量仅在使用 OpenSSL 版本 3.0 或更高版本时支持。对于旧版本，变量值将为空字符串。

`$ssl_curves`

返回客户端支持的曲线列表。已知的曲线以名称列出，未知的以十六进制显示，例如：

```
0x001d:prime256v1:secp521r1:secp384r1
```

重要

该变量仅在使用 OpenSSL 版本 1.0.2 或更高版本时支持。对于旧版本，变量值将为空字符串。

该变量仅在新会话中可用。

`$ssl_early_data`

如果使用 TLS 1.3 *early* 数据且握手尚未完成，则返回"1"，否则返回""。

`$ssl_protocol`

返回已建立 SSL 连接的协议。

`$ssl_server_cert_type`

根据服务器证书和密钥的类型，取值为 RSA、DSA、ECDSA、ED448、ED25519、SM2、RSA-PSS 或 unknown。

`$ssl_server_name`

返回通过 SNI 请求的服务器名称。

`$ssl_session_id`

返回已建立 SSL 连接的会话标识符。

`$ssl_session_reused`

如果 SSL 会话被重用，则返回 r，否则返回"."。

SSL Preread

启用从 ClientHello 消息中提取信息而不终止 TLS，例如通过 SNI 请求的服务器名称或在 ALPN 中广告的协议。

当 从源代码构建时，默认情况下不会构建此模块；它应该通过 `--with-stream_ssl_preread_module` 构建选项启用。

在来自 我们的仓库的包和镜像中，该模块已包含在构建中。

配置示例

通过服务器名称选择上游

```
map $ssl_preread_server_name $name {
    backend.example.com      backend;
    default                  backend2;
}

upstream backend {
    server 192.168.0.1:12345;
    server 192.168.0.2:12345;
}

upstream backend2 {
    server 192.168.0.3:12345;
```

```
    server 192.168.0.4:12345;
}

server {
    listen      12346;
    proxy_pass  $name;
    ssl_preread on;
}
```

通过协议选择服务器

```
map $ssl_preread_alpn_protocols $proxy {
    ~\bh2\b           127.0.0.1:8001;
    ~\bhttp/1.1\b     127.0.0.1:8002;
    ~\bxmpp-client\b  127.0.0.1:8003;
}

server {
    listen      9000;
    proxy_pass  $proxy;
    ssl_preread on;
}
```

通过 SSL 协议版本选择服务器

```
map $ssl_preread_protocol $upstream {
    ""        ssh.example.com:22;
    "TLSv1.2" new.example.com:443;
    default   tls.example.com:443;
}

# ssh 和 https 在同一端口
server {
    listen      192.168.0.1:443;
    proxy_pass  $upstream;
    ssl_preread on;
}
```

指令

ssl_preread

语法	ssl_preread on | off;
默认	ssl_preread off;
上下文	stream, server

在预读取 阶段启用从 ClientHello 消息中提取信息。

内置变量

`$ssl_preread_protocol`

客户端支持的最高 SSL 协议版本。

`$ssl_preread_server_name`

通过 SNI 请求的服务器名称。

`$ssl_preread_alpn_protocols`

客户端通过 ALPN 广告的协议列表。这些值用逗号分隔。

Upstream

该模块用于定义可以通过*proxy_pass* 指令引用的服务器组。

配置示例

```
upstream backend {
    hash $remote_addr consistent;
    zone backend 1m;

    server backend1.example.com:1935  weight=5;
    server unix:/tmp/backend3;
    server backend3.example.com       service=_example._tcp resolve;

    server backup1.example.com:1935   backup;
    server backup2.example.com:1935   backup;
}

resolver 127.0.0.53 status_zone=resolver;

server {
    listen 1936;
    proxy_pass backend;
}
```

指令

upstream

语法	upstream *name* { ... }
默认值	—
上下文	stream

定义一组服务器。服务器可以监听不同的端口。此外，监听 TCP 和 UNIX 域套接字的服务器可以混合使用。

示例：

```
upstream backend {
    server backend1.example.com:1935 weight=5;
    server 127.0.0.1:1935            max_fails=3 fail_timeout=30s;
    server unix:/tmp/backend2;
    server backend3.example.com:1935 resolve;

    server backup1.example.com:1935  backup;
}
```

默认情况下，连接在服务器之间使用加权轮询负载均衡方法进行分配。在上面的示例中，每 7 个连接将按如下方式分配：5 个连接发送到 backend1.example.com:1935，剩下的连接分别发送到第二和第三个服务器。

如果在与服务器通信时发生错误，连接将被传递到下一个服务器，依此类推，直到尝试所有可用的服务器为止。如果与所有服务器的通信都失败，连接将被关闭。

server

语法	`server` *address* [*parameters*];
默认值	—
上下文	upstream

定义服务器的地址和其他参数。地址可以指定为带有必需端口的域名或 IP 地址，或在 `unix:` 前缀后指定为 UNIX 域套接字路径。解析为多个 IP 地址的域名一次定义多个服务器。

可以定义以下参数：

`weight=`*number*	设置服务器的权重；默认值为 1。
`max_conns=`*number*	限制到代理服务器的最大同时活动连接数。默认值为 `0`，表示没有限制。如果服务器组不在共享内存 中，则限制适用于每个工作进程。

`max_fails=`*number* —设置在*fail_timeout* 指定的时间段内与服务器通信失败的尝试次数，以便认为服务器不可用；然后将在同一时间段后重试。

此处，不成功的尝试是指在与服务器建立连接时发生的错误或超时。

备注

如果组中的 `server` 指令解析为多个服务器，其 `max_fails` 设置将分别应用于每个服务器。

如果在解析完所有 `server` 指令后，上游中只有一个服务器，`max_fails` 设置将无效并被忽略。

`max_fails=1`	默认尝试次数。
`max_fails=0`	禁用尝试的计数。

`fail_timeout=`*time* —设置在指定时间段内应发生的与服务器通信失败次数 (*max_fails*) 以便认为服务器不可用。然后服务器在同一时间段内变得不可用，之后再重试。

默认设置为 10 秒。

备注

如果组中的 `server` 指令解析为多个服务器，其 `fail_timeout` 设置将分别应用于每个服务器。

如果在解析完所有 `server` 指令后，上游中只有一个服务器，`fail_timeout` 设置将无效并被忽略。

`backup`	将服务器标记为备份服务器。当主服务器不可用时，将传递请求给它。
`down`	将服务器标记为永久不可用。
`drain` (PRO)	将服务器设为排出状态；这意味着它仅接收早先通过*sticky* 绑定的会话请求。否则它的行为类似于 `down`。

小心

`backup` 参数不能与*hash* 和*random* 负载均衡方法一起使用。

`down` 和 `drain` 参数是互斥的。

Added in version 1.3.0.

`resolve`	启用对与域名对应的 IP 地址列表的更改进行监控，更新它而无需重新加载配置。该组必须存储在共享内存区 中；还需要定义*resolver*。
`service=`*name*	启用解析 DNS SRV 记录并设置服务名称。要使此参数生效，必须同时指定 *resolve* 参数，且不在主机名中指定服务器端口。 如果服务名称中没有点，名称将根据 RFC 标准形成：服务名称以 `_` 为前缀，然后在点后加上 `_tcp`。因此，服务名称 `http` 将变成 `_http._tcp`。 Angie 通过组合规范化的服务名称和主机名解析 SRV 记录，并通过 DNS 获取该组合的服务器列表及其优先级和权重。 • 最高优先级的 SRV 记录（具有最小优先级值的记录）解析为主服务器，其他记录变为备份服务器。如果在 `server` 中设置了 `backup`，最高优先级的 SRV 记录解析为备份服务器，其他记录将被忽略。 • 权重类似于 `server` 指令的 `weight` 参数。如果由指令和 SRV 记录同时设置权重，则使用指令设置的权重。

此示例将查找 _http._tcp.backend.example.com 记录：

```
server backend.example.com service=http resolve;
```

Added in version 1.2.0: Angie

Added in version 1.1.0-P1: Angie PRO

sid=*id*	设置组中的服务器 ID。如果未指定参数，ID 将设置为 IP 地址和端口或 UNIX 域套接字路径的十六进制 MD5 哈希值。

Added in version 1.4.0.

slow_start=*time*	设置服务器在重新投入服务时恢复其权重的时间，适用于使用*round-robin* 或*least_conn* 负载均衡方法的情况。 如果设置了该参数，且服务器根据*max_fails* 和*upstream_probe (PRO)* 的判断再次被视为健康，则服务器将在指定的时间段内逐渐恢复其指定权重。 如果未设置该参数，在类似情况下，服务器将立即以其指定权重开始工作。

备注

如果在上游中只指定了一个 server，则 slow_start 不会生效并将被忽略。

state (PRO)

Added in version 1.4.0: PRO

语法	state *file*;
默认值	—
上下文	upstream

指定持久存储上游服务器列表的 *file*。当从 我们的软件包安装时，会创建一个专用目录 /var/lib/angie/state/ (/var/db/angie/state/ 在 FreeBSD 上) 并设置适当的权限来存储这些文件，因此您只需在配置中添加文件名：

```
upstream backend {

    zone backend 1m;
    state /var/lib/angie/state/<FILE NAME>;
}
```

这里的服务器列表格式类似于 s_server。当通过配置 API 在*/config/stream/upstreams/* 部分修改服务器时，文件内容会发生变化。文件在 Angie 启动或配置重新加载时读取。

小心

要在 upstream 块中使用 state 指令，该块中不应有 server 指令，但需要一个共享内存区域 (*zone*)。

zone

语法	zone *name* [*size*];
默认值	—
上下文	upstream

定义共享内存区域的名称和大小，该区域存储组的配置和运行时状态，在工作进程之间共享。多个组可以使用同一个区域。在这种情况下，只需指定一次大小即可。

backup_switch (PRO)

Added in version 1.10.0: PRO

语法	backup_switch permanent[=*time*];
默认值	—
上下文	upstream

该指令启用一种能力，使服务器选择不从主要组开始，而是从 * 活动 * 组开始，即之前成功找到服务器的组。如果在活动组中找不到服务器处理下一个请求，搜索会转移到备份组，这个备份组成为活动组，后续请求首先定向到该组中的服务器。

如果定义了 permanent 参数但没有*time* 值，该组在选择后保持活动状态，不会自动重新检查优先级较低的组。如果指定了 *time*，组的活动状态在指定的时间间隔后过期，负载均衡器会再次检查优先级较低的组，如果服务器正常工作，则返回到这些组。

示例：

```
upstream media_backend {
    server primary1.example.com:1935;
    server primary2.example.com:1935;

    server reserve1.example.com:1935 backup;
    server reserve2.example.com:1935 backup;

    backup_switch permanent=2m;
}
```

如果负载均衡器从主服务器切换到备份组，所有后续请求将由该备份组处理 2 分钟。2 分钟过后，负载均衡器会重新检查主服务器，如果它们正常工作，则再次将它们设为活动状态。

feedback (PRO)

Added in version 1.7.0: PRO

语法	`feedback` *variable* [`inverse`] [`factor=`*number*] [`account=`*condition_variable*];
默认值	—
上下文	upstream

为 `upstream` 启用基于反馈的负载均衡机制。它通过将每个代理服务器的权重乘以平均反馈值来动态调整负载均衡决策，该值会随时间根据 *variable* 的值变化，并受可选条件的约束。

可以指定以下参数：

`variable`	从中获取反馈值的变量。它应代表性能或健康指标；假定由服务器提供。 每次从服务器收到响应时都会评估该值，并根据 `inverse` 和 `factor` 设置计入移动平均值。
`inverse`	如果设置了该参数，反馈值将被反向解释：较低的值表示更好的性能。
`factor`	计算平均值时反馈值的权重因子。有效值为 0 到 99 之间的整数。默认值为 90。 平均值使用 指数平滑 公式计算。 因子越大，新值对平均值的影响越小；如果指定 90，将取 90% 的前值和仅 10% 的新值。
`account`	指定一个条件变量，控制如何将连接计入计算。只有当条件变量不等于 "" 或 "0" 时，平均值才会用反馈值更新。 **备注** 默认情况下，来自探测 的流量不包括在计算中；结合*$upstream_probe* 变量与 `account` 可以将其包括在内，甚至可以排除所有其他流量。

示例：

```
upstream backend {

    zone backend 1m;

    feedback $feedback_value factor=80 account=$condition_value;

    server backend1.example.com:1935  weight=1;
    server backend2.example.com:1935  weight=2;
}

map $protocol $feedback_value {
    "TCP"                       100;
    "UDP"                       75;
    default                     10;
}
```

```
map $upstream_probe $condition_value {
    "high_priority" "1";
    "low_priority"  "0";
    default         "1";
}
```

此配置根据个别会话中使用的协议对服务器进行反馈级别分类，并且还添加了一个基于*$upstream_probe*的条件，只计算 high_priority 探测或常规客户端会话。

hash

语法	hash *key* [consistent];
默认值	—
上下文	upstream

指定一种负载均衡方法，其中客户端-服务器映射由哈希键值确定。键可以包含文本、变量及其组合。使用示例：

```
hash $remote_addr;
```

该方法与 Perl Cache::Memcached 库兼容。

如果指定了 consistent 参数，将使用 ketama 一致性哈希方法代替上述方法。该方法确保当服务器添加到组中或从组中移除时，只有最少数量的键会重新映射到其他服务器。对缓存服务器使用此方法可提供更高的缓存命中率。该方法与 Perl Cache::Memcached::Fast 库兼容，其中 ketama_points 参数设置为 160。

least_conn

语法	least_conn;
默认值	—
上下文	upstream

指定一种负载均衡方法，其中连接传递给活动连接数最少的服务器，同时考虑服务器权重。如果有多个合适的服务器，则以循环方式（round-robin）选择它们，并考虑其权重。

least_time (PRO)

语法	least_time connect | first_byte | last_byte [factor=*number*] [account=*condition_variable*];
默认值	—
上下文	upstream

指定一种负载均衡方法，其中将连接传递给活动服务器的概率与其平均响应时间成反比；响应时间越小，服务器接收的连接越多。

`connect`	该指令考虑平均连接建立时间。
`first_byte`	该指令使用接收响应第一个字节的平均时间。
`last_byte`	该指令使用接收完整响应的平均时间。

Added in version 1.7.0: PRO

`factor`	与*response_time_factor (PRO)* 具有相同功能，如果设置了该参数，则会覆盖它。
`account`	指定一个条件变量，控制哪些连接计入计算。只有当连接的条件变量不等于 `""` 或 `"0"` 时，才会更新平均值。 **备注** 默认情况下，探测 不包括在计算中；结合*$upstream_probe* 变量与 `account` 可以将其包括在内，甚至可以排除所有其他内容。

当前值在服务器的 `health` 对象中显示为 `connect_time` 、`first_byte_time` 和 `last_byte_time` ，作为 API 中*upstream metrics* 的一部分。

random

语法	`random [two];`
默认值	—
上下文	upstream

指定一种负载均衡方法，其中连接传递给随机选择的服务器，同时考虑服务器权重。

可选的 `two` 参数指示 Angie 随机选择两个服务器，然后使用指定的方法选择一个服务器。默认方法是*least_conn*，将请求传递到活动连接数最少的服务器。

response_time_factor (PRO)

Syntax	`response_time_factor` *number*;
默认值	`response_time_factor 90;`
Context	upstream

设置*least_time (PRO)* 负载均衡方法的平滑因子，在计算平均响应时间时使用 ** 先前 ** 的值，根据 '指数加权移动平均 <https://en.wikipedia.org/wiki/Moving_average#Exponential_moving_average>'___ 公式。

指定的 ‘number‘越大，新值对平均值的影响越小；如果指定:samp:*90* ，则将取先前值的 90%，而新值仅占 10%。有效值范围为 0 到 99（含）。

相应的移动平均值在服务器的:samp:*health* 对象中显示为 `connect_time` 连接建立时间） 、`first_byte_time` 接收响应第一个字节的时间）和:samp:*last_byte_time* （接收完整响应的时间），这些都可以在 API 的:ref:*stream upstream metrics <api_status_stream_upstreams>* 中找到。

备注

计算中只考虑成功的响应；什么构成不成功的响应由:ref:*s_proxy_next_upstream* 指令决定。

sticky

Added in version 1.6.0: Angie

Added in version 1.6.0: Angie PRO

Syntax	`sticky route` *$variable...*; `sticky learn zone=`*zone* `create=`*$create_var1...* `lookup=`*$lookup_var1...* [`connect`] [`norefresh`] [`timeout=`*time*]; `sticky learn lookup=`*$lookup_var1...* `remote_action=`*uri* `remote_result=`*$remote_var*;
默认值	—
Context	upstream

配置客户端会话与代理服务器的绑定，使用第一个参数指定的模式；要从配置了:samp:*sticky* 指令的服务器中排空请求，请在:ref:*server <s_u_server>* 块中使用:samp:*drain* 选项 (PRO)。

注意

`sticky` 指令必须在所有设置负载均衡方法的指令之后使用；否则将不起作用。

`route` 模式

此模式使用预定义的路由标识符，可以嵌入 Angie 可访问的连接属性中。它灵活性较低，因为依赖预定义值，但如果这些标识符已在使用中，则更适合。

在此，建立连接时，代理服务器可以为客户端分配路由并以双方都知道的方式返回其标识符。*server* 指令的:ref:*sid <s_reresolve>* 参数值必须用作路由标识符。注意，如果设置了:ref:*sticky_secret <s_u_sticky_secret>* 指令，该参数会被额外哈希。

希望使用此路由的客户端的后续连接必须包含服务器发出的标识符，确保其最终进入 Angie 变量。

指令参数指定用于路由的变量。选择传入连接路由到的服务器时，使用第一个非空变量；然后将其与:ref:*server <s_u_server>* 指令的:ref:*sid <s_reresolve>* 参数进行比较。如果选择服务器失败或所选服务器无法接受连接，则根据配置的负载均衡方法选择另一个服务器。

在此，Angie 在:samp:*$route* 变量中查找路由标识符，该变量基于:ref:*v_ssl_preread_server_name* 获取值（注意必须启用:ref:*s_ssl_preread* ）：

```
stream {

    map $ssl_preread_server_name $route {

        a.example.com            a;
        b.example.com            b;
        default                  "";
    }

    upstream backend {

        server 127.0.0.1:8081 sid=a;
        server 127.0.0.1:8082 sid=b;

        sticky route $route;
    }

    server {

        listen 127.0.0.1:8080;

        ssl_preread on;

        proxy_pass backend;
    }
}
```

`learn` 模式 (PRO)

此模式使用动态生成的密钥将客户端绑定到特定代理服务器；它更灵活，因为它动态分配服务器，在共享内存区域存储会话，并支持各种传递会话标识符的方式。

在此，基于来自代理服务器的连接属性创建会话。`create` 和:samp:*lookup* 参数列出表示如何创建新会话和查找现有会话的变量。两个参数都可以多次使用。

会话标识符是:samp:*create* 指定的第一个非空变量的值；例如，这可能是:ref: 代理服务器的名称 *<v_server_name>*。

会话存储在共享内存区域中；其名称和大小由:samp:*zone* 参数设置。如果会话在:samp:*timeout* 指定的:ref: 时间 *<syntax>* 内处于非活动状态，它将被删除。默认为 1 小时。

默认情况下，Angie 延长会话生命周期，每次使用时更新最后访问时间戳。`norefresh` 参数禁用此行为：会话将严格按超时过期，即使它继续被使用。当需要在一段时间后强制终止会话时，此模式很有用，例如，与外部会话管理器集成时。

希望使用会话的客户端的后续连接必须包含其标识符，确保其最终进入:samp:*lookup* 指定的非空变量；其值将与共享内存中的会话匹配。如果选择服务器失败或所选服务器无法处理连接，则根据配置的负载均

衡方法选择另一个服务器。

`connect` 参数允许在从代理服务器接收响应头后立即创建会话。没有它，只有在连接处理完成后才会创建会话。

在示例中，Angie 使用:ref:*$rdp_cookie <v_rdp_cookie>* 变量创建和查找会话：

```
stream {

    upstream backend {

        server 127.0.0.1:3390 sid=a;
        server 127.0.0.1:3391 sid=b;

        sticky learn lookup=$rdp_cookie create=$rdp_cookie zone=sessions:1m;
    }

    server {

        listen 127.0.0.1:3389;

        ssl_preread on;

        proxy_pass backend;
    }
}
```

带:samp:*remote_action* 的:samp:*learn* 模式 (PRO 1.10.0+)

`remote_action` 和:samp:*remote_result* 参数允许使用远程会话存储动态分配会话标识符并管理它们。

Angie 完全依赖远程存储：它不在本地缓存会话（尽管它允许通过:samp:*proxy_cache* 缓存存储响应），每次需要检索或创建会话时都会向远程存储发送单独的请求。

`remote_action` 参数指定远程存储的 URI，它应该按如下方式处理会话查找和创建：

- 存储从:samp:*lookup* 接收会话标识符和与此会话关联的本地建议的服务器标识符，通过自定义头或其他方法。

  在 Angie 端，为此提供了两个特殊变量：*$sticky_sessid* 和:ref:*v_s_sticky_sid*。`sticky_sid` 包含:ref:*s_u_upstream* 块中:samp:*server* 指令的:samp:*sid=* 参数的值（如果设置），或服务器名称的 MD5 哈希。

  > **备注**
  >
  > 如果:samp:*remote_action* 指向:ref:*client* 上下文中的:samp:*location* ，变量会自动导出到 HTTP 上下文，带有:samp:*stream_* 前缀（例如，`$stream_sticky_sessid` ，`$stream_sticky_sid`）。这允许在 HTTP 指令中直接使用，无需额外配置。
  >
  > 此外，在这种情况下，`remote_uri` 参数适用，指定客户端 HTTP 请求到指定:samp:*location* 的

URI。默认为:samp:*/create*。

- 来自远程存储的 200 响应表示它已接受会话并存储它，以便将来使用所提议的值。
- 来自远程存储的 409 响应表示给定的会话标识符已存在。在这种情况下，响应必须在:samp:*X-Sticky-Sid* 头中包含替代会话标识符。Angie 将此标识符存储在:samp:*remote_result* 参数指定的变量中。

以下是简化的配置示例。远程存储在:samp:*X-Sid* 头中返回会话标识符，从而确认或覆盖 Angie 的选择：

```
http {

    client {

        location @sticky_client1 {

            # 使用来自stream upstream的变量;
            # 它将这些变量添加到HTTP上下文，带有stream_*前缀
            proxy_set_header X-Sticky-Sessid $stream_sticky_sessid;
            proxy_set_header X-Sticky-Sid $stream_sticky_sid;
            proxy_set_header X-Sticky-Last $msec;
            proxy_pass http://127.0.0.1:8080;

            proxy_cache remote;
            proxy_cache_valid 200 1d;
            proxy_cache_key $scheme$proxy_host$request_uri$stream_sticky_sessid;
        }
    }
}

stream {

    upstream u {

        server 127.0.0.1:8081;
        server 127.0.0.1:8082;

        sticky learn lookup=$remote_addr            # stream变量
        remote_action=@sticky_client1               # 来自client块的location
        remote_result=$upstream_http_x_sid          # HTTP变量
        remote_uri=/foo;                            # 默认为/create
    }

    server {

        listen 127.0.0.1:8080;
        proxy_pass u;
    }
}
```

sticky_strict

Added in version 1.6.0: Angie

Added in version 1.6.0: Angie PRO

Syntax	`sticky_strict on	off;`
默认值	`sticky_strict off;`
Context	upstream

启用时，如果所需服务器不可用，Angie 会向客户端返回连接错误，而不是使用任何其他可用服务器，就像当组中没有服务器可用时那样。

sticky_secret

Added in version 1.6.0: Angie

Added in version 1.6.0: Angie PRO

Syntax	`sticky_secret` *string*;
默认值	—
Context	upstream

将 ‘string‘作为盐值添加到:samp:*route* 模式下:ref:*sticky <s_u_sticky>* 指令的 MD5 哈希函数中。*string* 可以包含变量，例如 ‘$remote_addr‘：

```
upstream backend {
    server 127.0.0.1:8081 sid=a;
    server 127.0.0.1:8082 sid=b;

    sticky route $route;
    sticky_secret my_secret.$remote_addr;
}
```

盐值附加在哈希值之后；要独立验证哈希机制：

```
$ echo -n "<VALUE><SALT>" | md5sum
```

内置变量

`stream_upstream` 模块支持以下内置变量：

$sticky_sessid

与*sticky* 中的 `remote_action` 一起使用；存储从 `lookup` 获取的初始会话标识符。

$sticky_sid

与*sticky* 中的 `remote_action` 一起使用；存储先前与会话关联的服务器标识符。

`sticky_sid` 包含*upstream* 块中 `server` 指令的 `sid=` 参数的值（如果指定），或服务器名称的 MD5 哈希值。

$upstream_addr

保持上游服务器的 IP 地址和端口，或 UNIX 域套接字的路径。如果在请求处理期间联系了多个服务器，其地址以逗号分隔，例如：

```
192.168.1.1:1935, 192.168.1.2:1935, unix:/tmp/sock
```

如果无法选择服务器，变量保存*server group* 的 *name*。

$upstream_bytes_received

从上游服务器接收的字节数。多个连接的值以逗号和冒号分隔，类似于*$upstream_addr* 变量中的地址。

$upstream_bytes_sent

发送到上游服务器的字节数。多个连接的值以逗号和冒号分隔，类似于*$upstream_addr* 变量中的地址。

$upstream_connect_time

连接上游服务器的时间；时间以秒为单位，精确到毫秒。多个连接的时间以逗号和冒号分隔，类似于*$upstream_addr* 变量中的地址。

$upstream_first_byte_time

接收第一个字节数据的时间；时间以秒为单位，精确到毫秒。多个连接的时间以逗号分隔，类似于*$upstream_addr* 变量中的地址。

$upstream_session_time

会话持续时间，以秒为单位，精确到毫秒。多个连接的时间以逗号分隔，类似于*$upstream_addr* 变量中的地址。

$upstream_sticky_status

粘性连接的状态。

""	连接路由到上游而未启用粘性。
NEW	无粘性信息的连接。
HIT	含粘性信息的连接路由到所需的后端。
MISS	含粘性信息的连接路由到由负载均衡算法选择的后端。

多个连接的值以逗号和冒号分隔，类似于*$upstream_addr* 变量中的地址。

Upstream Probe

该模块为*stream_upstream* 实现主动健康探针。

配置示例

```
server {
    listen ...;

    # ...
    proxy_pass backend;
    upstream_probe_timeout 1s;

    upstream_probe backend_probe
        port=12345
        interval=5s
        test=$good
        essential
        fails=3
        passes=3
        max_response=512k
        mode=onfail
        "send=data:GET / HTTP/1.0\r\n\r\n";
}
```

备注

根据 RFC 2616（HTTP/1.1）和 RFC 9110（HTTP 语义）规范，HTTP 头部必须使用 CRLF 序列（rn）进行分隔，而不能仅使用 n。

指令

upstream_probe (PRO)

Added in version 1.4.0: PRO

语法	upstream_probe *name* [port=*number*] [interval=*time*] [test=*condition*] [essential [persistent]] [fails=*number*] [passes=*number*] [max_response=*size*] [mode=always | idle | onfail] [udp] [send=*string*];
默认	—
上下文	server

定义一个针对*upstream* 组内服务器的主动健康探针，该组在与 upstream_probe 指令相同的 server 上下文中通过*proxy_pass* 指令指定。

如果向服务器的请求成功，并考虑到 upstream_probe 指令的所有参数设置以及影响 server 上下文如何使用上游的所有参数，包括*proxy_next_upstream* 指令，则服务器通过探针。

为了使用探针，上游必须具有共享内存区域 (*zone*)。一个上游可以配置多个探针。

接受以下参数：

name	探针的必需名称。
port	探针请求的备用端口号。
interval	探针之间的间隔。默认值—5s。
test	探针的条件，定义为变量字符串。如果变量替换结果为 "" 或 "0"，则探针不通过。
essential	如果设置，则检查服务器的初始状态，因此在探针通过之前，服务器不会接收客户端请求。
persistent	设置此参数需要先启用 essential；在配置重载 之前被认为健康的 persistent 服务器开始接收请求，而无需先通过此探针。
fails	使服务器不健康的连续失败探针次数。默认值—1。
passes	使服务器健康的连续通过探针次数。默认值—1。
max_response	响应的最大内存大小。如果指定零值，则禁用响应等待。默认值—256k。
mode	探针模式，取决于服务器的健康状态： • always —无论状态如何都探测服务器； • idle —探测不健康的服务器和自上次客户端请求以来经过 interval 的服务器。 • onfail —仅探测不健康的服务器。 默认值—always。
udp	如果指定，则使用 UDP 协议进行探测。默认情况下，使用 TCP 进行探测。
send	用于检查的数据；这可以是带有前缀 data: 的字符串或带有数据的文件名（绝对指定或相对于 /usr/local/angie/ 目录）。

示例：

```
upstream backend {
    zone backend 1m;

    server a.example.com;
    server b.example.com;
}
```

```
map $upstream_probe_response $good {
    ~200    "1";
    default  "";
}

server {
    listen ...;

    # ...
    proxy_pass backend;
    upstream_probe_timeout 1s;

    upstream_probe backend_probe
        port=12345
        interval=5s
        test=$good
        essential
        persistent
        fails=3
        passes=3
        max_response=512k
        mode=onfail
        "send=data:GET / HTTP/1.0\r\n\r\n";
}
```

探针操作细节：

- 最初，服务器在通过为其配置的 所有 `essential` 探针之前不会接收客户端请求，如果配置被重载并且服务器在此之前被认为是健康的，则跳过 `persistent` 探针。如果没有这样的探针，则认为服务器是健康的。
- 如果为服务器配置的 任何探针达到 `fails` 或服务器达到*max_fails*，则该服务器被认为是不健康的，不会接收客户端请求。
- 为了使不健康的服务器再次被认为是健康的，为其配置的 所有探针必须达到各自的 `passes`；之后，还会考虑*max_fails*。

内置变量

`stream_upstream` 模块支持以下内置变量：

`$upstream_probe` (PRO)

当前活动的*upstream_probe* 的名称。

$upstream_probe_response (PRO)

在由*upstream_probe* 配置的主动探针期间接收到的响应内容。

核心流模块实现了处理 TCP 和 UDP 连接的基本功能：这包括定义服务器块、流量路由、配置代理、SSL/TLS 支持，以及管理用于流式服务（如数据库、DNS 和其他通过 TCP 和 UDP 操作的协议）的连接。

本节中的其他模块扩展了此功能，使您能够灵活地配置和优化流服务器以满足各种场景和需求。

当 从源代码构建时，此模块默认不构建；它应通过 `--with-stream` 构建选项启用。在来自 我们的仓库的软件包和镜像中，该模块已包含在构建中。

配置示例

```
worker_processes auto;

error_log /var/log/angie/error.log info;

events {
    worker_connections  1024;
}

stream {
    upstream backend {
        hash $remote_addr consistent;

        server backend1.example.com:12345 weight=5;
        server 127.0.0.1:12345            max_fails=3 fail_timeout=30s;
        server unix:/tmp/backend3;
    }

    upstream dns {
       server 192.168.0.1:53535;
       server dns.example.com:53;
    }

    server {
        listen 12345;
        proxy_connect_timeout 1s;
        proxy_timeout 3s;
        proxy_pass backend;
    }

    server {
        listen 127.0.0.1:53 udp reuseport;
        proxy_timeout 20s;
        proxy_pass dns;
    }

    server {
```

```
        listen [::1]:12345;
        proxy_pass unix:/tmp/stream.socket;
    }
}
```

指令

listen

语法	`listen` *address*[:*port*] [`ssl`] [`udp`] [`proxy_protocol`] [`setfib=`*number*] [`fastopen=`*number*] [`backlog=`*number*] [`rcvbuf=`*size*] [`sndbuf=`*size*] [`accept_filter=`*filter*] [`deferred`] [`bind`] [`ipv6only=on` | `off`] [`reuseport`] [`so_keepalive=`on|off|[*keepidle*]:[*keepintvl*]:[*keepcnt*]] [`multipath`];
默认	—
上下文	server

设置服务器将接受连接的套接字的 *address* 和 *port*。可以仅指定 *port*，此时 Angie 将监听所有可用的 IPv4（以及启用时的 IPv6）接口。地址也可以是主机名，例如：

```
listen 127.0.0.1:12345;
listen *:12345;
listen 12345;      # 同 *:12345
listen localhost:12345;
```

IPv6 地址用方括号指定：

```
listen [::1]:12345;
listen [::]:12345;
```

UNIX 域套接字用 `unix:` 前缀指定：

```
listen unix:/var/run/angie.sock;
```

端口范围用第一个和最后一个端口用连字符分隔：

```
listen 127.0.0.1:12345-12399;
listen 12345-12399;
```

重要

不同的服务器必须侦听不同的 *address:port* 组合。

`ssl`	允许指定所有在此端口上接受的连接应在 SSL 模式下工作。
`udp`	配置用于处理数据报的监听套接字。为了在同一会话中处理来自同一地址和端口的数据包，还应指定 *reuseport* 参数。
`proxy_protocol`	允许指定所有在此端口上接受的连接应使用 PROXY 协议。
`multipath`	启用通过 多路径 TCP (MPTCP) 协议接受连接，该协议从 Linux 内核 5.6 版本开始支持。此参数与 `udp` **不兼容**。

listen 指令可以有几个与套接字相关的系统调用的附加参数。

`setfib=`*number*	为正在监听的套接字设置关联的路由表，FIB (即 `SO_SETFIB` 选项)。此功能目前仅在 FreeBSD 上有效。
`fastopen=`*number*	为监听套接字启用"TCP 快速打开"，并限制尚未完成三次握手的连接队列的最大长度 <https://datatracker.ietf.org/doc/html/rfc7413#section-5.1>。

小心

除非服务器能够处理接收相同的 SYN 数据包 <https://datatracker.ietf.org/doc/html/rfc7413#section-6.1> 多次，否则请不要启用此功能。

`backlog=`*number*	设置 `listen()` 调用中的 `backlog` 参数，限制待处理连接队列的最大长度。默认情况下，在 FreeBSD、DragonFly BSD 和 macOS 上，`backlog` 设置为 `-1`，在其他平台上设置为 511。
`rcvbuf=`*size*	设置监听套接字的接收缓冲区大小 (即 `SO_RCVBUF` 选项)。
`sndbuf=`*size*	设置监听套接字的发送缓冲区大小 (即 `SO_SNDBUF` 选项)。
`accept_filter=`*filt*	设置监听套接字的接收过滤器名称 (即 `SO_ACCEPTFILTER` 选项)，在将传入连接传递给 `accept()` 之前过滤它们。此功能仅在 FreeBSD 和 NetBSD 5.0+ 上有效。可接受的值为 `dataready` 和 `httpready`。
`deferred`	指示在 Linux 上使用延迟 `accept()` (即 `TCP_DEFER_ACCEPT` 套接字选项)。
`bind`	此参数指示对给定的 *address:port* 组合进行单独的 `bind()` 调用。实际上，如果有多个*listen* 指令具有相同的 *port* 但不同的地址，并且其中一个 `listen` 指令为给定端口监听所有地址 (**:port*)，则 Angie 仅会对 **:port* 进行 `bind()`。需要注意的是，在这种情况下，将会调用 `getsockname()` 系统调用以确定接受连接的地址。如果使用了 `setfib`、`fastopen`、`backlog`、`rcvbuf`、`sndbuf`、`accept_filter`、`deferred`、`ipv6only`、`reuseport` 或 `so_keepalive` 参数，则对于给定的 *address:port* 组合将始终进行单独的 `bind()` 调用。
`ipv6only=on` | `off`	此参数确定 (通过 `IPV6_V6ONLY` 套接字选项) IPv6 套接字在通配地址 *[::]* 上是否仅接受 IPv6 连接或同时接受 IPv6 和 IPv4 连接。此参数默认开启。它只能在启动时设置一次。
`reuseport`	此参数指示为每个工作进程创建一个独立的监听套接字 (在 Linux 3.9+ 和 DragonFly BSD 上使用 `SO_REUSEPORT` 套接字选项，或在 FreeBSD 12+ 上使用 `SO_REUSEPORT_LB`)，允许内核在工作进程之间分配传入连接。此功能目前仅在 Linux 3.9+、DragonFly BSD 和 FreeBSD 12+ 上有效。 **☢ 小心** 不当使用此选项可能会带来安全隐患。

`so_keepalive=on` | `off` | [*keepidle*]:[*keepintvl*]:[*keepcnt*]

配置监听套接字的"TCP 保持活动" 行为。

`''`	如果省略此参数，则将对套接字生效操作系统的设置。
`on`	为套接字开启 *SO_KEEPALIVE* 选项。
`off`	为套接字关闭 *SO_KEEPALIVE* 选项。

某些操作系统支持使用 `TCP_KEEPIDLE`、`TCP_KEEPINTVL` 和 `TCP_KEEPCNT` 套接字选项在每个套接字的基础上设置 TCP 保持活动参数。在此类系统上 (目前，Linux 2.4+、NetBSD 5+ 和 FreeBSD 9.0-STABLE)，可以使用 keepidle、keepintvl 和 keepcnt 参数进行配置。可以省略一个或两个参数，在这种情况下，将对相应的套接字选项使用系统默认设置。

例如，

```
so_keepalive=30m::10
```

将把空闲超时 (*TCP_KEEPIDLE*) 设置为 30 分钟，将探测间隔 (*TCP_KEEPINTVL*) 保持为系统默认值，并将探测计数 (*TCP_KEEPCNT*) 设置为 10 次探测。

preread_buffer_size

语法	`preread_buffer_size` *size*;
默认	`preread_buffer_size 16k;`
上下文	stream, server

指定*preread* 缓冲区的大小。

preread_timeout

语法	`preread_timeout` *timeout*;
默认	`preread_timeout 30s;`
上下文	stream, server

指定*preread* 阶段的超时。

proxy_protocol_timeout

语法	`proxy_protocol_timeout` *timeout*;
默认	`proxy_protocol_timeout 30s;`
上下文	stream, server

指定读取 PROXY 协议头以完成的 *timeout*。如果在此时间内未传输整个头，则连接将关闭。

resolver

语法	`resolver` *address* ... [`valid=`*time*] [`ipv4=on` | `off`] [`ipv6=on` | `off`] [`status_zone=`*zone*];
默认	—
上下文	stream, server, upstream

配置用于将上游服务器的名称解析为地址的名称服务器，例如：

```
resolver 127.0.0.53 [::1]:5353;
```

地址可以指定为域名或 IP 地址，并可选指定端口。如果未指定端口，则使用端口 53。名称服务器以循环的方式查询。

默认情况下，Angie 使用响应的 TTL 值缓存答案。可选的 valid 参数允许覆盖它：

`valid`	可选 valid 参数允许覆盖缓存条目的有效性

```
resolver 127.0.0.53 [::1]:5353 valid=30s;
```

默认情况下，Angie 在解析时会查找 IPv4 和 IPv6 地址。

`ipv4=off`	禁用 IPv4 地址的查找
`ipv6=off`	禁用 IPv6 地址的查找

`status_zone`	可选参数；允许在指定区域中收集 DNS 服务器请求和响应指标（*/status/resolvers/<zone>*）。

小技巧

为防止 DNS 欺骗，建议在适当安全的受信任本地网络中配置 DNS 服务器。

小技巧

当在 Docker 中运行时，使用其内部 DNS 服务器地址，例如 `127.0.0.11`。

resolver_timeout

语法	`resolver_timeout` 时间;
默认	`resolver_timeout 30s;`
上下文	stream, server, upstream

设置名称解析的超时时间，例如：

```
resolver_timeout 5s;
```

server

语法	`server { ... }`
默认	—
上下文	stream

设置服务器的配置。

server_name

语法	server_name *name* ...;
默认	server_name "";
上下文	server

设置虚拟服务器的名称。

> **⚠ 注意**
>
> 在 stream 模块中，server_name 指令基于服务器名称指示 (*SNI*) 并且仅适用于 TLS 连接。要使用它，必须在相应的 server 块中配置 *TLS* 终止 或启用 *TLS* 预读取。

示例配置：

```
server {
    listen 443 ssl;
    server_name example.com www.example.com;
    ssl_certificate /etc/angie/cert.pem;
    ssl_certificate_key /etc/angie/key.pem;
}
```

第一个名称将成为主服务器名称。

服务器名称可以包含星号 (*)，用于替换名称的首部或尾部：

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

这些名称称为通配符名称。

还可以在服务器名称中使用正则表达式，方法是在名称前加上波浪号 (~)：

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

正则表达式可以包含捕获组，这些捕获组可以用于其他指令：

```
server {
    server_name ~^(www\.)?(.+)$;

    proxy_pass www.$2:12345;
}
```

正则表达式中的命名捕获会创建变量，可以在其他指令中使用：

```
server {
    server_name ~^(www\.)?(?<domain>.+)$;

    proxy_pass www.$domain:12345;
}
```

如果指令的参数设置为 `$hostname`，则插入机器的主机名。

在通过名称查找虚拟服务器时，如果名称与多个指定变体匹配（例如同时是通配符名称和正则表达式匹配），将按照以下优先级选择第一个匹配变体：

- 精确名称
- 以星号开头的最长通配符名称，例如，`*.example.com`
- 以星号结尾的最长通配符名称，例如 `mail.*`
- 第一个匹配的正则表达式（按出现在配置文件中的顺序）

server_names_hash_bucket_size

语法	`server_names_hash_bucket_size` *size*;
默认	`server_names_hash_bucket_size 32|64|128;`
上下文	stream

设置服务器名称哈希表的桶大小。默认值取决于处理器缓存行的大小。

server_names_hash_max_size

语法	`server_names_hash_max_size` *size*;
默认	`server_names_hash_max_size 512;`
上下文	stream

设置服务器名称哈希表的最大大小。

status_zone

语法	`status_zone` *zone* | *key* `zone=`*zone*[:*count*];
默认	—
上下文	*server*

分配一个共享内存区域以收集 */status/stream/server_zones/<zone>* 的指标。

多个 `server` 上下文可以共享同一区域进行数据收集。

单值 *zone* 语法会将当前上下文的所有指标聚合到一个共享内存区域：

```
server {

    listen 80;
    server_name *.example.com;

    status_zone single;
    # ...
}
```

替代语法允许指定以下参数：

key	包含变量的字符串，其值决定了连接在区域中的分组。所有在替换后产生相同值的连接将被分为一组。如果替换结果为空，则不更新指标。
zone	共享内存区域的名称。
count （可选）	收集指标的最大分组数。如果新的 *key* 值超过此限制，则将它们归入 *zone* 下。默认值为 1。

在以下示例中，所有共享相同 `$server_addr` 值的连接被分组到 `host_zone` 中。对于每个唯一的 `$server_addr` 单独收集指标，直到指标组数量达到 10 个。之后，任何新的 `$server_addr` 值将被添加到 `server_zone` 组：

```
stream {

    upstream backend {
        server 192.168.0.1:3306;
        server 192.168.0.2:3306;
        # ...
    }

    server {

        listen 3306;
        proxy_pass backend;

        status_zone $server_addr zone=server_zone:10;
    }
}
```

最终的指标在 API 输出中在各个服务器之间分配。

stream

语法	`stream { ... }`
默认	—
上下文	main

提供配置文件上下文，在其中指定流服务器指令。

tcp_nodelay

语法	`tcp_nodelay on | off;`
默认	`tcp_nodelay on;`
上下文	stream, server

启用或禁用 *TCP_NODELAY* 选项。该选项对客户端连接和代理服务器连接均启用。

variables_hash_bucket_size

语法	`variables_hash_bucket_size` *size*;
默认	`variables_hash_bucket_size 64;`
上下文	stream

设置变量哈希表的桶大小。设置哈希表的详细信息在单独的文档 中提供。

variables_hash_max_size

语法	`variables_hash_max_size` *size*;
默认	`variables_hash_max_size 1024;`
上下文	stream

设置变量哈希表的最大大小。设置哈希表的详细信息在单独的文档 中提供。

内置变量

stream core 模块支持以下变量：

$angie_version

Angie 版本

`$binary_remote_addr`

以二进制形式表示的客户端地址，IPv4 地址的值长度始终为 4 字节，IPv6 地址为 16 字节

`$bytes_received`

从客户端接收的字节数

`$bytes_sent`

发送到客户端的字节数

`$connection`

连接序列号

`$hostname`

主机名

`$msec`

当前时间（以秒为单位，精确到毫秒）

`$pid`

工作进程的 PID

`$protocol`

与客户端通信使用的协议：`TCP` 或 `UDP`

`$proxy_protocol_addr`

来自 PROXY 协议头的客户端地址。必须通过在*listen* 指令中设置 *proxy_protocol* 参数来启用 PROXY 协议。

`$proxy_protocol_port`

来自 PROXY 协议头的客户端端口。必须通过在*listen* 指令中设置 *proxy_protocol* 参数来启用 PROXY 协议。

`$proxy_protocol_server_addr`

来自 PROXY 协议头的服务器地址。必须通过在*listen* 指令中设置 *proxy_protocol* 参数来启用 PROXY 协议。

`$proxy_protocol_server_port`

来自 PROXY 协议头的服务器端口。必须通过在*listen* 指令中设置 *proxy_protocol* 参数来启用 PROXY 协议。

`$proxy_protocol_tlv_<name>`

从 PROXY 协议头获取的 TLV。*name* 可以是 TLV 类型名称或其数值。在后一种情况下，值以十六进制指定，必须以 *0x* 开头：

```
$proxy_protocol_tlv_alpn
$proxy_protocol_tlv_0x01
```

SSL TLV 也可以通过 TLV 类型名称和其数值访问，两者都必须以 `ssl_` 开头：

```
$proxy_protocol_tlv_ssl_version
$proxy_protocol_tlv_ssl_0x21
```

支持以下 TLV 类型名称：

- `alpn` (0x01) - 连接上使用的上层协议
- `authority` (0x02) - 客户端传递的主机名值
- `unique_id` (0x05) - 唯一连接标识符
- `netns` (0x30) - 命名空间名称
- `ssl` (0x20) - 二进制格式的 SSL TLV 结构

支持以下 SSL TLV 类型名称：

- `ssl_version` (0x21) - 客户端连接中使用的 SSL 版本
- `ssl_cn` (0x22) - 证书通用名称
- `ssl_cipher` (0x23) - 使用的密码名称
- `ssl_sig_alg` (0x24) - 用于签署证书的算法
- `ssl_key_alg` (0x25) - 公钥算法

还支持以下特殊 SSL TLV 类型名称：

- `ssl_verify` - 客户端证书验证结果：如果客户端提供了证书并成功验证则为 0，否则为非零值

必须通过在*listen* 指令中设置 *proxy_protocol* 参数来启用 PROXY 协议。

`$remote_addr`

客户端地址

`$remote_port`

客户端端口

`$server_addr`

接受连接的服务器地址。计算此变量的值通常需要一次系统调用。为避免系统调用，*listen* 指令必须指定地址并使用 `bind` 参数。

`$server_port`

接受连接的服务器端口

`$session_time`

会话持续时间，以秒为单位，精确到毫秒

`$status`

会话状态，可以是以下之一：

200	会话成功完成
400	客户端数据无法解析，例如，PROXY 协议头
403	访问被禁止，例如，当访问受到特定客户端地址 限制时
500	内部服务器错误
502	错误的网关，例如，如果无法选择或到达上游服务器
503	服务不可用，例如，当访问受到连接数量 限制时

`$time_iso8601`

ISO 8601 标准格式的本地时间

`$time_local`

常见日志格式的本地时间

邮件模块

Auth HTTP

该模块通过在处理主请求之前发送额外的 HTTP 请求来实现基于子请求的认证。如果子请求返回 2xx 状态，主请求将继续处理；如果返回 401 或 403，相应的错误将发送给用户，而任何其他响应都会触发 500 错误。这种方法通常用于将认证委托给外部服务，统一跨应用程序的认证，或与 OAuth 或 LDAP 等第三方系统集成。

指令

auth_http

语法	`auth_http` *uri*;
默认值	—
上下文	mail, server

设置 HTTP 认证服务器的 URL。协议描述见下文。

auth_http_header

语法	`auth_http_header` *header value*;
默认值	—
上下文	mail, server

将指定的头部附加到发送给认证服务器的请求中。此头部可以用作共享密钥，以验证请求是否来自 Angie。例如：

```
auth_http_header X-Auth-Key "secret_string";
```

auth_http_pass_client_cert

语法	`auth_http_pass_client_cert on | off;`
默认值	`auth_http_pass_client_cert off;`
上下文	mail, server

将"Auth-SSL-Cert" 头部与 PEM 格式（urlencoded）的客户端证书 附加到发送给认证服务器的请求中。

auth_http_timeout

语法	`auth_http_timeout` *time*;
默认值	`auth_http_timeout 60s;`
上下文	mail, server

设置与认证服务器通信的超时时间。

协议

HTTP 协议用于与认证服务器通信。响应体中的数据被忽略，仅在头部传递信息。

请求和响应示例：

请求：

```
GET /auth HTTP/1.0
Host: localhost
Auth-Method: plain # plain/apop/cram-md5/external
Auth-User: user
Auth-Pass: password
Auth-Protocol: imap # imap/pop3/smtp
Auth-Login-Attempt: 1
Client-IP: 192.0.2.42
Client-Host: client.example.org
```

成功响应：

```
HTTP/1.0 200 OK
Auth-Status: OK
Auth-Server: 198.51.100.1
Auth-Port: 143
```

失败响应：

```
HTTP/1.0 200 OK
Auth-Status: Invalid login or password
Auth-Wait: 3
```

如果没有”Auth-Wait” 头部，将返回错误并关闭连接。当前实现为每次认证尝试分配内存。内存仅在会话结束时释放。因此，单个会话中的无效认证尝试次数必须有限制——在 10-20 次尝试后，服务器必须响应不带”Auth-Wait” 头部（尝试次数在”Auth-Login-Attempt” 头部中传递）。

当使用 APOP 或 CRAM-MD5 时，请求响应如下所示：

```
GET /auth HTTP/1.0
Host: localhost
Auth-Method: apop
Auth-User: user
Auth-Salt: <238188073.1163692009@mail.example.com>
Auth-Pass: auth_response
Auth-Protocol: imap
Auth-Login-Attempt: 1
Client-IP: 192.0.2.42
Client-Host: client.example.org
```

成功响应：

```
HTTP/1.0 200 OK
Auth-Status: OK
Auth-Server: 198.51.100.1
```

```
Auth-Port: 143
Auth-Pass: plain-text-pass
```

如果响应中存在"Auth-User" 头部，它将覆盖用于与后端认证的用户名。

对于 SMTP，响应还会考虑"Auth-Error-Code" 头部——如果存在，它将在错误情况下用作响应代码。否则，*535 5.7.0* 代码将被添加到"Auth-Status" 头部。

例如，如果从认证服务器收到以下响应：

```
HTTP/1.0 200 OK
Auth-Status: Temporary server problem, try again later
Auth-Error-Code: 451 4.3.0
Auth-Wait: 3
```

则 SMTP 客户端将收到错误

```
451 4.3.0 Temporary server problem, try again later
```

如果代理 SMTP 不需要认证，请求将如下所示：

```
GET /auth HTTP/1.0
Host: localhost
Auth-Method: none
Auth-User:
Auth-Pass:
Auth-Protocol: smtp
Auth-Login-Attempt: 1
Client-IP: 192.0.2.42
Client-Host: client.example.org
Auth-SMTP-Helo: client.example.org
Auth-SMTP-From: MAIL FROM: <>
Auth-SMTP-To: RCPT TO: <postmaster@mail.example.com>
```

对于 SSL/TLS 客户端连接，添加"Auth-SSL" 头部，并且"Auth-SSL-Verify" 将包含客户端证书验证的结果，如果启用：`SUCCESS`，`FAILED:reason`，以及如果没有证书则为 `NONE`。

当存在客户端证书时，其详细信息将通过以下请求头部传递："Auth-SSL-Subject"，"Auth-SSL-Issuer"，"Auth-SSL-Serial"，和"Auth-SSL-Fingerprint"。如果启用了*auth_http_pass_client_cert*，证书本身将通过"Auth-SSL-Cert" 头部传递。已建立连接的协议和密码将通过"Auth-SSL-Protocol" 和"Auth-SSL-Cipher" 头部传递。请求将如下所示：

```
GET /auth HTTP/1.0
Host: localhost
Auth-Method: plain
Auth-User: user
Auth-Pass: password
Auth-Protocol: imap
Auth-Login-Attempt: 1
```

```
Client-IP: 192.0.2.42
Auth-SSL: on
Auth-SSL-Protocol: TLSv1.3
Auth-SSL-Cipher: TLS_AES_256_GCM_SHA384
Auth-SSL-Verify: SUCCESS
Auth-SSL-Subject: /CN=example.com
Auth-SSL-Issuer: /CN=example.com
Auth-SSL-Serial: C07AD56B846B5BFF
Auth-SSL-Fingerprint: 29d6a80a123d13355ed16b4b04605e29cb55a5ad
```

当使用*PROXY 协议* 时，其详细信息将通过以下请求头部传递：”Proxy-Protocol-Addr”, ”Proxy-Protocol-Port”, ”Proxy-Protocol-Server-Addr”, 和”Proxy-Protocol-Server-Port”。

IMAP

该模块启用 IMAP 邮件协议支持，允许服务器与邮件存储系统交互。它建立与 IMAP 服务器的连接，处理常见命令如列出邮箱和检索消息，并提供安全的身份验证和消息状态管理。

指令

imap_auth

语法	`imap_auth` *method* ...;
默认值	`imap_auth plain;`
上下文	mail, server

设置 IMAP 客户端允许的身份验证方法。支持的方法有：

`plain`	LOGIN, AUTH=PLAIN
`login`	AUTH=LOGIN
`cram-md5`	AUTH=CRAM-MD5。为了使此方法正常工作，密码必须以未加密的形式存储。
`external`	AUTH=EXTERNAL

明文身份验证方法（`LOGIN` 命令，`AUTH=PLAIN`，和 `AUTH=LOGIN`）始终启用，尽管如果未指定 `plain` 和 `login` 方法，`AUTH=PLAIN` 和 `AUTH=LOGIN` 将不会自动包含在*imap_capabilities* 中。

imap_capabilities

语法	`imap_capabilities` *extension* ...;
默认值	`imap_capabilities IMAP4 IMAP4rev1 UIDPLUS;`
上下文	mail, server

设置在响应 CAPABILITY 命令时传递给客户端的 IMAP 协议 扩展列表。根据*starttls* 指令的值，*imap_auth* 指令中指定的身份验证方法和 STARTTLS 会自动添加到此列表中。

指定 IMAP 后端支持的扩展是有意义的，这些后端是客户端的代理（如果这些扩展与身份验证后使用的命令相关，当 Angie 透明地将客户端连接代理到后端时）。

imap_client_buffer

语法	`imap_client_buffer` *size*;
默认值	`imap_client_buffer 4k|8k;`
上下文	mail, server

设置用于读取 IMAP 命令的缓冲区大小。默认情况下，缓冲区大小等于一个内存页。这是 4K 或 8K，具体取决于平台。

POP3

该模块启用 POP3 邮件协议支持，允许服务器从邮件服务器下载消息。它连接到 POP3 服务器，检索消息头和内容，提供安全认证，并管理消息状态，如已下载或已删除。

指令

pop3_auth

语法	`pop3_auth` *method* ...;
默认	`pop3_auth plain;`
上下文	mail, server

设置允许的 POP3 客户端身份验证方法。支持的方法有：

`plain`	USER/PASS，AUTH PLAIN，AUTH LOGIN
`apop`	APOP。为了使此方法生效，密码必须以未加密形式存储。
`cram-md5`	AUTH=CRAM-MD5。为了使此方法生效，密码必须以未加密形式存储。
`external`	AUTH=EXTERNAL

明文身份验证方法（`USER/PASS`，`AUTH PLAIN` 和 `AUTH LOGIN`）始终启用，尽管如果没有指定 `plain` 方法，`AUTH PLAIN` 和 `AUTH LOGIN` 将不会自动包含在*pop3_capabilities* 中。

pop3_capabilities

语法	`pop3_capabilities` *extension* ...;
默认	`pop3_capabilities TOP USER UIDL;`
上下文	mail, server

设置在响应 CAPA 命令时传递给客户端的 POP3 协议 扩展列表。根据*starttls* 指令的值，*pop3_auth* 指令中指定的身份验证方法（SASL 扩展）和 STLS 会自动添加到此列表中。

指定 POP3 后端支持的扩展是有意义的，这些后端是客户端代理的（如果这些扩展与身份验证后使用的命令相关，当 Angie 透明地将客户端连接代理到后端时）。

Proxy

该模块提供对邮件协议（POP3、IMAP、SMTP）的支持，使服务器能够充当客户端和邮件服务器之间的代理。它与服务器建立连接，使用明文、SSL/TLS 或 STARTTLS 执行安全身份验证，正确路由客户端流量，并支持灵活选择身份验证方法和服务器。

指令

proxy_buffer

语法	`proxy_buffer` *size*;
默认值	`proxy_buffer 4k|8k;`
上下文	mail, server

设置用于代理的缓冲区大小。默认情况下，缓冲区大小等于一个内存页。根据平台的不同，它是 4K 或 8K。

proxy_pass_error_message

语法	`proxy_pass_error_message on | off;`
默认值	`proxy_pass_error_message off;`
上下文	mail, server

指示是否将后端在身份验证过程中获得的错误消息传递给客户端。

通常，如果在 Angie 中身份验证成功，则后端无法返回错误。如果它仍然返回错误，则意味着发生了一些内部错误。在这种情况下，后端消息可能包含不应显示给客户端的信息。然而，对于某些 POP3 服务器，正确密码的错误响应是正常行为。在这种情况下，应启用该指令。

proxy_protocol

语法	`proxy_protocol on | off;`
默认值	`proxy_protocol off;`
上下文	mail, server

为与后端的连接启用 PROXY 协议。

proxy_smtp_auth

语法	`proxy_smtp_auth on | off;`
默认值	`proxy_smtp_auth off;`
上下文	mail, server

启用或禁用使用 *AUTH* 命令在 SMTP 后端进行用户身份验证。

如果*XCLIENT* 也启用，则 *XCLIENT* 命令将不会发送 *LOGIN* 参数。

proxy_timeout

语法	`proxy_timeout` *time*;
默认值	`proxy_timeout 24h;`
上下文	mail, server

设置客户端或代理服务器连接之间两个连续读或写操作的超时。如果在此时间内未传输任何数据，则连接将被关闭。

xclient

语法	`xclient on | off;`
默认值	`xclient on;`
上下文	mail, server

启用或禁用在连接到 SMTP 后端时传递带有客户端参数的 XCLIENT 命令。

使用 XCLIENT，邮件传输代理能够将客户端信息写入日志，并根据这些数据应用各种限制。

如果启用 XCLIENT，则在连接到后端时 Angie 会传递以下命令：

- EHLO 与服务器名称
- XCLIENT
- EHLO 或 HELO，如客户端所传递

如果通过客户端 IP 地址找到 的名称指向相同地址，则它会在 XCLIENT 命令的 NAME 参数中传递。如果找不到名称，指向不同的地址，或未指定解析器，则在 NAME 参数中传递 [UNAVAILABLE]。如果在解析过程中发生错误，则使用 [TEMPUNAVAIL] 值。

如果禁用 XCLIENT，则在连接到后端时，如果客户端传递了 EHLO，Angie 会与服务器名称 一起传递 EHLO 命令，否则会与服务器名称一起传递 HELO。

RealIP

该模块用于将客户端地址和端口更改为在 PROXY 协议头中发送的地址和端口。必须通过在*listen* 指令中设置 *proxy_protocol* 参数来先前启用 PROXY 协议。

配置示例

```
listen 110 proxy_protocol;

set_real_ip_from  192.168.1.0/24;
set_real_ip_from  192.168.2.1;
set_real_ip_from  2001:0db8::/32;
```

指令

set_real_ip_from

Syntax	`set_real_ip_from` *address* | *CIDR* | `unix:`;
Default	—
Context	mail, server

定义已知发送正确替换地址的受信任地址。如果指定了特殊值 `unix:`，则所有 UNIX 域套接字都将被信任。

SMTP

该模块启用对 SMTP 邮件协议的支持，允许服务器代理客户端和邮件服务器之间的出站电子邮件流量。它建立与 SMTP 服务器的连接，支持使用 LOGIN 或 PLAIN 方法进行安全认证，提供 STARTTLS 和 SSL/TLS 加密，并根据认证结果路由客户端请求。

指令

smtp_auth

语法	`smtp_auth` *method* ...;
默认值	`smtp_auth plain login;`
上下文	mail, server

设置 SMTP 客户端允许的 SASL 认证 方法。支持的方法包括：

`plain`	AUTH PLAIN
`login`	AUTH LOGIN
`cram-md5`	AUTH CRAM-MD5。为了使此方法正常工作，密码必须以未加密的形式存储。
`external`	AUTH EXTERNAL
`none`	不需要认证

明文认证方法 (AUTH PLAIN 和 AUTH LOGIN) 始终启用，但如果未指定 plain 和 login 方法，则 AUTH PLAIN 和 AUTH LOGIN 将不会自动包含在*smtp_capabilities* 中。

smtp_capabilities

语法	smtp_capabilities *extension* ...;
默认值	—
上下文	mail, server

设置在响应 EHLO 命令时传递给客户端的 SMTP 协议扩展列表。根据*starttls* 指令的值，*smtp_auth* 指令中指定的认证方法和 STARTTLS 会自动添加到此列表中。

指定客户端被代理到的 MTA 所支持的扩展是有意义的（如果这些扩展与认证后使用的命令相关，当 Angie 透明地将客户端连接代理到后端时）。

smtp_client_buffer

语法	smtp_client_buffer *size*;
默认值	smtp_client_buffer 4k|8k;
上下文	mail, server

设置用于读取 SMTP 命令的缓冲区大小。默认情况下，缓冲区大小等于一个内存页面。根据平台不同，这通常是 4K 或 8K。

smtp_greeting_delay

语法	smtp_greeting_delay *time*;
默认值	smtp_greeting_delay 0;
上下文	mail, server

允许在发送 SMTP 问候语之前设置延迟，以拒绝在发送 SMTP 命令之前未能等待问候的客户端。

SSL

该模块为邮件代理协议（POP3、IMAP、SMTP）启用 SSL/TLS 加密支持，允许客户端和服务器之间进行安全通信。它为传入连接提供 SSL/TLS 加密，支持 STARTTLS 升级，管理证书和密钥，并控制 SSL 设置，如密码和协议版本。

当从 源代码构建时，该模块默认不会构建；应该使用 --with-mail_ssl_module 构建选项启用它。

在来自 我们仓库的软件包和镜像中，该模块已包含在构建中。

重要

此模块需要 OpenSSL 库。

配置示例

为了减少处理器负载，建议

- 将工作进程数量 设置为与处理器数量相等，
- 启用共享 会话缓存，
- 禁用内置 会话缓存，
- 并可能增加会话生命周期 （默认为 5 分钟）：

```
worker_processes auto;

mail {

    ...

    server {
        listen              993 ssl;

        ssl_protocols       TLSv1.2 TLSv1.3;
        ssl_ciphers         AES128-SHA:AES256-SHA:RC4-SHA:DES-CBC3-SHA:RC4-MD5;
        ssl_certificate     /usr/local/angie/conf/cert.pem;
        ssl_certificate_key /usr/local/angie/conf/cert.key;
        ssl_session_cache   shared:SSL:10m;
        ssl_session_timeout 10m;

    #    ...
    }
```

指令

ssl_certificate

语法	ssl_certificate *file*;
默认值	—
上下文	mail, server

指定给定服务器的 PEM 格式证书文件。如果除了主证书外还需要指定中间证书，应按以下顺序在同一文件中指定：主证书在前，然后是中间证书。PEM 格式的密钥可以放在同一文件中。

此指令可以多次指定，以加载不同类型的证书，例如 RSA 和 ECDSA：

```
server {
    listen              993 ssl;

    ssl_certificate     example.com.rsa.crt;
    ssl_certificate_key example.com.rsa.key;

    ssl_certificate     example.com.ecdsa.crt;
    ssl_certificate_key example.com.ecdsa.key;

#    ...
}
```

只有 OpenSSL 1.0.2 或更高版本支持不同证书的单独证书链。对于旧版本，只能使用一个证书链。

可以指定值"data:*certificate*" 来代替 `file`，这会加载证书而不使用中间文件。

请注意，不当使用此语法可能会带来安全隐患，例如将密钥数据写入错误日志。

ssl_certificate_key

语法	`ssl_certificate_key` *file*;
默认值	—
上下文	mail, server

指定给定服务器的 PEM 格式密钥文件。

可以指定值 `engine:`name`:id` 来代替 `file`，这会从 OpenSSL 引擎 *name* 加载指定 *id* 的密钥。

可以指定值"data:*key*" 来代替 `file`，这会加载密钥而不使用中间文件。请注意，不当使用此语法可能会带来安全隐患，例如将密钥数据写入错误日志。

ssl_ciphers

语法	`ssl_ciphers` *ciphers*;
默认值	`ssl_ciphers HIGH:!aNULL:!MD5;`
上下文	mail, server

指定启用的密码。密码以 OpenSSL 库理解的格式指定，例如：

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

密码列表取决于已安装的 OpenSSL 版本。可以使用 `openssl ciphers` 命令查看完整列表。

注意

当使用 OpenSSL 时，`ssl_ciphers` 指令 * 不会 * 配置 TLS 1.3 的加密套件。要配置 OpenSSL 的 TLS 1.3 加密套件，请使用*ssl_conf_command* 指令，该指令是为支持高级 SSL 配置而添加的。

- 在 LibreSSL 中，可以使用 `ssl_ciphers` 配置 TLS 1.3 加密套件。
- 在 BoringSSL 中，完全无法配置 TLS 1.3 加密套件。

ssl_client_certificate

语法	`ssl_client_certificate` *file*;
默认值	—
上下文	mail, server

指定用于验证 客户端证书的 PEM 格式受信任 CA 证书文件。

证书列表将发送给客户端。如果不希望这样，可以使用*ssl_trusted_certificate* 指令。

ssl_conf_command

语法	`ssl_conf_command` *name value*;
默认值	—
上下文	mail, server

设置任意 OpenSSL 配置 命令。

重要

该指令在使用 OpenSSL 1.0.2 或更高版本时受支持。要配置 OpenSSL 的 TLS 1.3 加密套件，请使用 `ciphersuites` 命令。

可以在同一级别上指定多个 *ssl_conf_command* 指令：

```
ssl_conf_command Options PrioritizeChaCha;
ssl_conf_command Ciphersuites TLS_CHACHA20_POLY1305_SHA256;
```

这些指令仅在当前级别没有定义 *ssl_conf_command* 指令时，才会从前一个配置级别继承。

小心

请注意，直接配置 OpenSSL 可能会导致意外行为。

ssl_crl

语法	`ssl_crl` *file*;
默认值	—
上下文	mail, server

指定用于验证 客户端证书的 PEM 格式撤销证书（CRL）文件。

ssl_dhparam

语法	`ssl_dhparam` *file*;
默认值	—
上下文	mail, server

指定 DHE 密码的 DH 参数文件。

小心

默认情况下未设置参数，因此不会使用 DHE 密码。

ssl_ecdh_curve

语法	`ssl_ecdh_curve` *curve*;
默认值	`ssl_ecdh_curve auto;`
上下文	mail, server

指定 ECDHE 密码的曲线。

重要

当使用 OpenSSL 1.0.2 或更高版本时，可以指定多个曲线，例如：

```
ssl_ecdh_curve prime256v1:secp384r1;
```

特殊值 `auto` 指示 Angie 在使用 OpenSSL 1.0.2 或更高版本时使用 OpenSSL 库内置的曲线列表，或在较旧版本中使用 prime256v1。

重要

当使用 OpenSSL 1.0.2 或更高版本时，此指令设置服务器支持的曲线列表。因此，为了使 ECDSA 证书正常工作，包含证书中使用的曲线非常重要。

ssl_password_file

语法	ssl_password_file *file*;
默认值	—
上下文	mail, server

指定一个文件，其中包含密钥 的密码，每个密码单独一行。加载密钥时会依次尝试这些密码。

示例：

```
mail {
    ssl_password_file /etc/keys/global.pass;
    ...

    server {
        server_name mail1.example.com;
        ssl_certificate_key /etc/keys/first.key;
    }

    server {
        server_name mail2.example.com;

        # 也可以使用命名管道代替文件
        ssl_password_file /etc/keys/fifo;
        ssl_certificate_key /etc/keys/second.key;
    }
}
```

ssl_prefer_server_ciphers

语法	ssl_prefer_server_ciphers on | off;
默认值	ssl_prefer_server_ciphers off;
上下文	mail, server

指定在使用 SSLv3 和 TLS 协议时，服务器密码应优先于客户端密码。

ssl_protocols

语法	ssl_protocols [SSLv2] [SSLv3] [TLSv1] [TLSv1.1] [TLSv1.2] [TLSv1.3];
默认值	ssl_protocols TLSv1.2 TLSv1.3;
上下文	mail, server

在 1.2.0 版本发生变更: 默认设置中添加了 TLSv1.3 参数。

启用指定的协议。

重要

TLSv1.1 和 TLSv1.2 参数仅在使用 OpenSSL 1.0.1 或更高版本时有效。

TLSv1.3 参数仅在使用 OpenSSL 1.1.1 或更高版本时有效。

ssl_session_cache

语法	`ssl_session_cache off` | `none` | [builtin[:*size*]] [shared:*name*:*size*];
默认值	`ssl_session_cache none;`
上下文	mail, server

设置存储会话参数的缓存的类型和大小。缓存可以是以下类型中的任何一种：

`off`	严格禁止使用会话缓存：Angie 明确告诉客户端会话不能重用。
`none`	温和地不允许使用会话缓存：Angie 告诉客户端会话可以重用，但实际上不在缓存中存储会话参数。
`builtin`	OpenSSL 内置的缓存；仅由一个工作进程使用。缓存大小以会话为单位。如果未给出大小，则等于 20480 个会话。使用内置缓存可能会导致内存碎片。
`shared`	在所有工作进程之间共享的缓存。缓存大小以字节为单位；一兆字节大约可以存储 4000 个会话。每个共享缓存应具有任意名称。具有相同名称的缓存可以在几个服务器中使用。它还用于自动生成、存储和定期轮换 TLS 会话票据密钥，除非明确使用*ssl_session_ticket_key* 指令配置。

可以同时使用两种缓存类型，例如：

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

但仅使用共享缓存而不使用内置缓存应更有效。

ssl_session_ticket_key

语法	`ssl_session_ticket_key` *file*;
默认值	—
上下文	mail, server

设置一个文件，其中包含用于加密和解密 TLS 会话票据的密钥。该指令在多个服务器之间需要共享相同密钥时是必要的。默认情况下，使用随机生成的密钥。

如果指定多个密钥，则仅使用第一个密钥来加密 TLS 会话票据。这允许配置密钥轮换，例如：

```
ssl_session_ticket_key current.key;
ssl_session_ticket_key previous.key;
```

该文件必须包含 80 或 48 字节的随机数据，可以使用以下命令创建：

```
openssl rand 80 > ticket.key
```

根据文件大小，使用 AES256（针对 80 字节密钥）或 AES128（针对 48 字节密钥）进行加密。

ssl_session_tickets

语法	`ssl_session_tickets on | off;`
默认值	`ssl_session_tickets on;`
上下文	mail, server

启用或禁用通过 TLS 会话票据 的会话恢复。

ssl_session_timeout

语法	`ssl_session_timeout` *time*;
默认值	`ssl_session_timeout 5m;`
上下文	mail, server

指定客户端可以重用会话参数的时间。

ssl_trusted_certificate

语法	`ssl_trusted_certificate` *file*;
默认值	—
上下文	mail, server

指定一个文件，其中包含用于验证 客户端证书的受信任 CA 证书，格式为 PEM。

与*ssl_client_certificate* 设置的证书不同，这些证书的列表不会发送给客户端。

ssl_verify_client

语法	`ssl_verify_client on | off | optional | optional_no_ca;`
默认值	`ssl_verify_client off;`
上下文	mail, server

启用客户端证书的验证。验证结果将通过认证 请求的”Auth-SSL-Verify” 头传递。如果在客户端证书验证过程中发生错误，或者客户端未提供所需的证书，则连接将被关闭。

`optional`	请求客户端证书，并在证书存在时进行验证
`optional_no_ca`	请求客户端证书，但不要求其由受信任的 CA 证书签名。这适用于外部服务执行实际证书验证的情况。证书的内容可以通过请求发送 给认证服务器访问。

ssl_verify_depth

语法	`ssl_verify_depth` *number*;
默认值	`ssl_verify_depth 1;`
上下文	mail, server

设置客户端证书链中的验证深度。

starttls

语法	`starttls on | off | only;`
默认值	`starttls off;`
上下文	mail, server

`on`	允许使用 POP3 的 STLS 命令和 IMAP 和 SMTP 的 STARTTLS 命令；
`off`	拒绝使用 STLS 和 STARTTLS 命令；
`only`	要求进行预先的 TLS 转换。

核心邮件模块实现了邮件代理服务器的基本功能：包括支持 SMTP、IMAP 和 POP3 协议、配置服务器块、邮件请求路由、用户认证以及 SSL/TLS 支持以保护邮件连接。

本节中的其他模块扩展了此功能，使您能够灵活地配置和优化邮件服务器以满足各种场景和需求。

当从 源代码构建时，默认情况下不会构建此模块；它应该通过 `--with-mail` 构建选项启用。在来自 我们仓库的软件包和镜像中，该模块包含在构建中。

配置示例

```
worker_processes auto;

error_log /var/log/angie/error.log info;

events {
    worker_connections  1024;
}

mail {
    server_name       mail.example.com;
```

```
    auth_http          localhost:9000/cgi-bin/auth.cgi;

    imap_capabilities IMAP4rev1 UIDPLUS IDLE LITERAL+ QUOTA;

    pop3_auth          plain apop cram-md5;
    pop3_capabilities LAST TOP USER PIPELINING UIDL;

    smtp_auth          login plain cram-md5;
    smtp_capabilities "SIZE 10485760" ENHANCEDSTATUSCODES 8BITMIME DSN;
    xclient            off;

    server {
        listen   25;
        protocol smtp;
    }
    server {
        listen   110;
        protocol pop3;
        proxy_pass_error_message on;
    }
    server {
        listen   143;
        protocol imap;
    }
    server {
        listen   587;
        protocol smtp;
    }
}
```

指令

listen

语法	`listen` *address*[:*port*] [`ssl`] [`proxy_protocol`] [`backlog=`*number*] [`rcvbuf=`*size*] [`sndbuf=`*size*] [`bind`] [`ipv6only=on` | `off`] [`reuseport`] [`so_keepalive=`on|off|[*keepidle*]:[*keepintvl*]:[*keepcnt*]];
默认	—
上下文	server

设置服务器接受请求的套接字的 *address* 和 *port*。可以仅指定 *port*，此时 Angie 将监听所有可用的 IPv4（以及启用时的 IPv6）接口。地址也可以是主机名，例如：

```
listen 127.0.0.1:110;
listen *:110;
listen 110;      # same as *:110
```

```
listen localhost:110;
```

IPv6 地址用方括号指定：

```
listen [::1]:110;
listen [::]:110;
```

UNIX 域套接字使用 `unix:` 前缀指定：

```
listen unix:/var/run/angie.sock;
```

重要

不同的服务器必须在不同的 *address:port* 对上监听。

`ssl`	允许指定在此端口上接受的所有连接应处于 SSL 模式。
`proxy_protocol`	允许指定在此端口上接受的所有连接应使用 PROXY 协议。获取的信息会传递给认证服务器，并可用于更改客户端地址。

`listen` 指令可以具有几个与套接字相关的系统调用的附加参数。

`backlog=`*number*	设置 `listen()` 调用中的 *backlog* 参数，限制待处理连接的最大队列长度。默认情况下，FreeBSD、DragonFly BSD 和 macOS 上的 `backlog` 设置为 `-1`，而其他平台上设置为 511。
`rcvbuf=`*size*	设置监听套接字的接收缓冲区大小（`SO_RCVBUF` 选项）。
`sndbuf=`*size*	设置监听套接字的发送缓冲区大小（`SO_SNDBUF` 选项）。
`bind`	此参数指示为给定的 *address:port* 对进行单独的 `bind()` 调用。事实是，如果有多个*listen* 指令具有相同的 *port* 但不同的地址，而其中一条 `listen` 指令在给定端口上监听所有地址 (`*:port`)，Angie 将仅 `bind()` 到 **:port*。需要注意的是，在这种情况下，将调用 `getsockname()` 系统调用以确定接受连接的地址。如果使用了 `backlog`、`rcvbuf`、`sndbuf`、`ipv6only`、`reuseport` 或 `so_keepalive` 参数，则对于给定的 *address:port* 对将始终进行单独的 `bind()` 调用。
`ipv6only=on` | `off`	此参数通过 *IPV6_V6ONLY* 套接字选项确定在通配符地址 *[::]* 上监听的 IPv6 套接字是否仅接受 IPv6 连接，还是同时接受 IPv6 和 IPv4 连接。默认情况下启用此参数。它只能在启动时设置一次。
`multipath`	启用通过 Multipath TCP (MPTCP) 接受连接，支持 Linux 内核版本 5.6 及更高版本。

`so_keepalive=on` | `off` | `[keepidle]:[keepintvl]:[keepcnt]`

配置监听套接字的"TCP keepalive" 行为。

`''`	如果省略此参数，则操作系统的设置将对套接字生效
`on`	为套接字启用 *SO_KEEPALIVE* 选项
`off`	为套接字禁用 *SO_KEEPALIVE* 选项

某些操作系统支持使用 `TCP_KEEPIDLE`、`TCP_KEEPINTVL` 和 `TCP_KEEPCNT` 套接字选项在每个套接字上设置 TCP keepalive 参数。在此类系统上（目前，Linux 2.4+、NetBSD 5+ 和 FreeBSD 9.0-STABLE），可以使用 keepidle、keepintvl 和 keepcnt 参数进行配置。可以省略一个或两个参数，此时相应套接字选项的系统默认设置将生效。

例如，

```
so_keepalive=30m::10
```

将把空闲超时（*TCP_KEEPIDLE*）设置为 30 分钟，保持探测间隔（*TCP_KEEPINTVL*）为系统默认，并将探测计数（*TCP_KEEPCNT*）设置为 10 次探测。

mail

语法	`mail { ... }`
默认	—
上下文	main

提供邮件服务器指令指定的配置文件上下文。

max_commands

Added in version 1.7.0.

语法	`max_commands` *number*;
默认	`max_commands 1000;`
上下文	mail, server

设置在身份验证期间发出的最大命令数，以增强对 DoS 攻击的保护。

max_errors

语法	`max_errors` *number*;
默认	`max_errors 5;`
上下文	mail, server

设置发生协议错误后关闭连接的次数。

protocol

语法	`protocol` *imap* | *pop3* | *smtp;*
默认	—
上下文	server

设置代理服务器的协议。支持的协议有*IMAP*、*POP3* 和*SMTP*。

如果未设置该指令，可以根据 `listen` 指令中指定的知名端口自动检测协议：

```
imap: 143, 993
pop3: 110, 995
smtp: 25, 587, 465
```

当 从源代码构建时，可以使用 `--without-mail_imap_module`、`--without-mail_pop3_module` 和 `--without-mail_smtp_module` 构建选项禁用不需要的协议。

resolver

语法	`resolver` *address* ... [`valid=`*time*] [`ipv4=on` | `off`] [`ipv6=on` | `off`] [`status_zone=`*zone*];
默认	`resolver off;`
上下文	mail, server

配置用于查找客户端主机名的名称服务器，以将其传递给认证服务器，以及在代理 SMTP 时用于*XCLIENT* 命令。例如：

```
resolver 127.0.0.53 [::1]:5353;
```

地址可以指定为域名或 IP 地址，并带有可选的端口。如果未指定端口，则使用端口 53。名称服务器按照循环方式进行查询。

默认情况下，Angie 使用响应的 TTL 值缓存答案。可选的 valid 参数允许覆盖它：

`valid`	可选 valid 参数允许覆盖缓存条目的有效期

```
resolver 127.0.0.53 [::1]:5353 valid=30s;
```

默认情况下，Angie 在解析时会同时查找 IPv4 和 IPv6 地址。

`ipv4=off`	禁用 IPv4 地址的查找
`ipv6=off`	禁用 IPv6 地址的查找
`status_zone`	可选参数，为指定区域启用统计信息收集

> **小技巧**
>
> 为防止 DNS 欺骗，建议在适当安全的受信任本地网络中配置 DNS 服务器。

> **小技巧**
>
> 在 Docker 中运行时，使用其内部 DNS 服务器地址，如 `127.0.0.11`。

resolver_timeout

语法	`resolver_timeout` *time;*
默认	`resolver_timeout 30s;`
上下文	mail, server

设置 DNS 操作的超时时间，例如：

```
resolver_timeout 5s;
```

server

语法	`server { ... }`
默认	—
上下文	mail

设置服务器的配置。

server_name

语法	`server_name` *name*;
默认	`server_name hostname;`
上下文	mail, server

设置服务器名称的使用：

- 在初始 POP3/SMTP 服务器问候中；
- 在 SASL CRAM-MD5 身份验证期间的盐值中；
- 在连接到 SMTP 后端时的 EHLO 命令中，如果启用了*XCLIENT* 命令的传递。

如果未指定该指令，则使用机器的 *hostname*。

timeout

语法	`timeout` *time;*
默认	`timeout 60s;`
上下文	mail, server

设置在开始代理到后端之前使用的超时时间。

Google PerfTools

该模块使用 Google 性能工具 启用 Angie 工作进程的性能分析。该模块旨在为 Angie 开发人员提供服务，允许他们通过提供有关内存使用、CPU 负载及其他性能指标的详细信息来分析和优化服务器性能。

当从源代码 构建时，该模块默认不被构建；需通过使用 `--with-google_perftools_module` 构建参数来启用。

> **重要**
>
> 该模块需要 gperftools 库。

配置示例

```
google_perftools_profiles /var/log/angie/perftools;
```

性能分析文件将存储为 `/var/log/angie/perftools.<` 工作进程 `PID>` 格式的文件。

指令

google_perftools_profiles

语法	`google_perftools_profiles` 文件名前缀;
默认值	—
上下文	main

设置用于存储 Angie 工作进程性能分析信息的文件名前缀。工作进程 ID 将附加在名称末尾，并以点号分隔，例如：`/var/log/angie/perftools.1234`。

WASM 模块

WAMR

该模块提供与 WebAssembly Micro Runtime 的集成，用于执行 WASM 代码，并向*wasm_modules* 上下文添加多个特定于运行时的指令。

在我们的代码库中，该模块是 动态构建的，并作为一个名为 `angie-module-wamr` 的独立软件包提供。

配置示例

```
wasm_modules {

    wamr_heap_size 16k;

    wamr_stack_size 16k;

    load fft_transform.wasm id=fft;
}
```

指令

wamr_heap_size

语法	`wamr_heap_size` *size*;
默认	`wamr_heap_size 8k;`
上下文	wasm_modules

为单个模块实例设置堆大小。

wamr_global_heap_size

语法	`wamr_global_heap_size` *size*;
默认	`wamr_global_heap_size 1m;`
上下文	wasm_modules

为整个 WAMR 运行时设置堆大小。

wamr_stack_size

语法	`wamr_stack_size` *size*;
默认	`wamr_stack_size 8k;`
上下文	wasm_modules

为单个模块实例设置栈大小。

Wasmtime

该模块支持与 Wasmtime 运行时的集成，以执行 WASM 代码，并向*wasm_modules* 上下文添加了一些特定于运行时的指令。

在我们的代码库中，该模块是以 动态方式构建的，并作为一个名为 `angie-module-wasmtime` 的单独包提供。

配置示例

```
wasm_modules {

    wasmtime_stack_size 8k;

    wasmtime_enable_wasi on;

    load fft_transform.wasm id=fft;
}
```

指令

wasmtime_enable_wasi

语法	`wasmtime_enable_wasi on | off;`
默认值	`wasmtime_enable_wasi on;`
上下文	wasm_modules

启用或禁用 WebAssembly System Interface API 的使用，这些 API 为在 Angie 上运行的 WASM 模块提供了基本的 POSIX 类功能。

> **备注**
>
> 可以使用*load* 指令明确允许使用 Angie 特定的 API。

wasmtime_stack_size

语法	`wasmtime_stack_size` *size*;
默认值	`wasmtime_stack_size 8k;`
上下文	wasm_modules

将 max_wasm_stack 值设置为特定的*size*，从而限制执行 WASM 代码时可用的最大堆栈空间。

核心模块实现了 Angie 中的基本 WASM 功能：这包括支持加载替代运行时和 WASM 模块，以及配置它们的特性和限制。

本节中的其他模块扩展了此功能，使您能够灵活配置和优化各种场景和需求的 WASM 特性。

在我们的代码库中，该模块构建为 动态，并作为一个名为 `angie-module-wasm` 的独立包提供。

配置示例

```
# 这些指令加载核心功能
load_module modules/ngx_wasm_module.so;
load_module modules/ngx_wasm_core_module.so;

load_module modules/ngx_wasmtime_module.so;

# 可在此处找到: https://git.angie.software/web-server/angie-wasm
load_module modules/ngx_http_wasm_host_module.so;

events {

}

wasm_modules {

    #use wasmtime;

    load ngx_http_handler.wasm id=handler;
    load ngx_http_vars.wasm id=vars type=reactor;
}

http {

    wasm_var vars "ngx:wasi/var-utils#sum-entry" $rvar $arg_a $arg_b $arg_c $arg_d;

    server {

        listen *:8080;

        location / {

            return 200 "sum('$arg_a','$arg_b','$arg_c','$arg_d')=$rvar\n";
        }

        location /wasm {

            client_max_body_size 20M;
            wasm_content handler "ngx:wasi/http-handler-entry#handle-request";
        }
    }
}
```

指令

load

语法	`load` *file* `id=`*identifier* [`fs=`*host_path:guest_path*]... [`api=`*api*]... [`type=command` | `reactor`]
默认	—
上下文	wasm_modules

从磁盘 *file* 加载模块并为其分配一个唯一的 *identifier* （必需参数）。在加载过程中，将验证模块以确保它可以实例化。

该指令支持以下参数：

`fs`	允许客户端访问主机上的目录。可以为不同的目录多次指定该参数。
`api`	通过列出 API 明确限制模块允许使用的 API 列表。如果模块尝试使用不可用的 API（未在此处列出），将返回"API 未找到" 错误。 默认情况下，所有 API 都可用于模块。
`type`	控制加载模块的生命周期。 • 在 `command` 模式下，机器执行一次并在执行后销毁其状态。 • 在 `reactor` 模式下，机器有效地无限期运行，允许代码多次执行。这需要仔细的内存管理：如果资源未被释放，可能会发生内存泄漏。

wasm_modules

语法	`wasm_modules { ... };`
默认	—
上下文	main

一个顶级指令，提供了应该指定 WASM 指令的配置文件上下文。它可以包含用于加载 WASM 模块的命令和配置特定运行时参数的命令。

核心模块

Core	负责管理服务文件、进程和其他 Angie 模块。

HTTP 模块

HTTP	处理 HTTP 请求和响应的核心功能，管理 HTTP 服务器、连接和静态文件。
Access	基于 IP 地址和 CIDR 范围的访问控制。

表 1 – 接上页

ACME	使用 ACME 协议为 HTTP 服务器自动获取和更新 SSL 证书。
Docker	基于 Docker 容器标签动态更新代理服务器组。
Addition	在响应体之前或之后插入指定的代码片段。
API	RESTful HTTP 接口，用于以 JSON 格式获取 Web 服务器的基本信息和统计数据，以及管理
Auth Basic	基于用户名和密码的基本 HTTP 身份验证进行访问控制。
Auth Request	通过向外部 HTTP 服务的子请求进行授权。
AutoIndex	在没有索引文件的情况下自动生成目录列表。
Browser (已弃用)	基于 `User-Agent` 头的浏览器识别。
Charset	响应编码的配置和转换。
DAV	使用 WebDAV 协议在服务器上进行文件管理。
Empty GIF	提供一个一像素透明 GIF。
FastCGI	将请求代理到 FastCGI 服务器。
FLV	Flash 视频（FLV）文件的伪流式传输。
Geo	将 IP 地址转换为指定的变量值。
GeoIP	使用 MaxMind GeoIP 数据库基于地理位置获取 IP 地址数据。
gRPC	将请求代理到 gRPC 服务器。
GunZIP	解压缩 GZip 压缩的响应，用于修改和客户端不支持压缩的情况。
GZip	使用 GZip 方法压缩响应以节省流量。
GZip Static	提供使用 GZip 方法预先压缩的静态文件。
Headers	修改响应头字段。
HTTP2	使用 HTTP/2 协议处理请求。
HTTP3	使用 HTTP/3 协议处理请求。
Image Filter	图像转换。
Index	配置索引文件，用于处理以斜杠（/）结尾的请求。
JS	通过在 njs（JavaScript 语言的子集）中指定附加逻辑来扩展功能的处理程序。
Limit Conn	限制并发请求（活动连接）的数量，以防止过载。
Limit Req	限制请求频率，以防止过载和密码猜测。
Log	配置请求日志，用于跟踪资源访问，以便进行监控和分析。
Map	基于预定义的键值对转换变量。
Memcached	从 Memcached 服务器获取响应。
Mirror	将请求镜像到其他服务器。
MP4	MP4 文件的伪流式传输。
Perl	通过在 Perl 语言中指定附加逻辑来扩展功能的处理程序。
Prometheus	以 Prometheus 兼容格式提供服务器指标，用于监控和统计收集。
Proxy	将请求反向代理到其他 HTTP 服务器。
Random Index	为以斜杠（/）结尾的请求随机选择索引文件。
RealIP	在另一个代理服务器后运行时，确定客户端地址和端口。
Referer	`Referer` 头值的验证。
Rewrite	请求 URI 修改、重定向、变量设置和条件配置选择。
SCGI	将请求代理到 SCGI 服务器。
Secure Link	创建安全链接，能够限制访问时间。
Slice	将请求拆分为多个针对单个片段的子请求，以便更好地缓存大型响应。
Split Clients	为 A/B 测试、金丝雀发布、分片和其他需要比例分组的场景创建变量。

表 1 – 接上页

SSI	处理响应中的 SSI（服务器端包含）命令。
SSL	处理 HTTPS 请求的 SSL/TLS 配置。
Stub Status (已弃用)	以文本格式提供全局连接和请求计数器。
Sub	在响应体中搜索和替换片段。
Upstream	配置用于负载均衡的代理服务器组。
Upstream Probe	为代理服务器组配置主动健康检查。
UserID	发布和处理带有唯一客户端标识符的 cookie，用于会话跟踪和分析。
uWSGI	将请求代理到 uWSGI 服务器。
XSLT	使用 XSLT 语言转换 XML 文档。

Stream 模块

Stream	核心流服务器功能，用于在 L4 级别平衡 TCP 和 UDP 协议。
Access	基于 IP 地址和 CIDR 范围的访问控制。
ACME	使用 ACME 协议为流服务器自动获取和更新 SSL 证书。
Geo	将 IP 地址转换为指定的变量值。
GeoIP	使用 MaxMind GeoIP 数据库基于地理位置获取 IP 地址数据。
JS	通过在 njs（JavaScript 语言的子集）中指定附加逻辑来扩展功能的处理程序。
Limit Conn	限制并发连接的数量，以防止过载。
Log	配置会话日志，用于跟踪资源访问，以便进行监控和分析。
Map	基于预定义的键值对转换变量。
MQTT Preread	在做出负载均衡决策之前，从 MQTT 连接中读取客户端标识符和用户名。
Pass	将接受的连接直接传递到配置的监听套接字。
Proxy	配置代理到其他服务器。
RDP Preread	在做出负载均衡决策之前，从 RDP 连接中读取 cookie。
RealIP	在另一个代理服务器后运行时，确定客户端地址和端口。
Return	在连接时向客户端发送指定值，无需进一步代理。
Set	设置指定的变量值。
Split Clients	为 A/B 测试、金丝雀发布、分片和其他需要比例分组的场景创建变量。
SSL	SSL/TLS 和 DTLS 协议终止。
SSL Preread	在不终止 SSL/TLS 的情况下从 `ClientHello` 消息中提取信息，并在做出负载均衡决策之前。
Upstream	配置用于负载均衡的代理服务器组。
Upstream Probe	为代理服务器组配置主动健康检查。

Mail 模块

Mail	核心邮件代理服务器功能。
Auth HTTP	通过向外部服务器发送 HTTP 请求进行用户身份验证和后续代理的服务器选择。
IMAP	IMAP 协议支持。
POP3	POP3 协议支持。
Proxy	配置代理到其他服务器。
RealIP	在另一个代理服务器后运行时，确定客户端地址和端口。
SMTP	SMTP 协议支持。
SSL	SSL/TLS 和 StartTLS 协议支持。

Google PerfTools 模块

Google PerfTools	负责与 Google Performance Tools 库集成，用于应用程序分析和性能分析。

WASM 模块

WASM	核心 WASM 功能，使 Angie 能够执行 WASM 代码。
WAMR	与 WebAssembly Micro Runtime 集成。
Wasmtime	与 Wasmtime 运行时环境集成。

3.2.2 内置变量

HTTP 模块	*Stream 模块*
$acme_cert_key_< 名称 >	*$acme_cert_key_<name>*
$acme_cert_< 名称 >	*$acme_cert_<name>*
$ancient_browser	
$angie_version	*$angie_version*
$arg_<name>	
$args	
$binary_remote_addr	*$binary_remote_addr*
$body_bytes_sent	
	$bytes_received
$bytes_sent	*$bytes_sent*
$connection	*$connection*
$connection_requests	
$connection_time	
$connections_active	
$connections_reading	
$connections_writing	
$connections_waiting	

续下页

表 2 – 接上页

HTTP 模块	Stream 模块
$content_length	
$content_type	
$cookie_<name>	
$date_local	
$date_gmt	
$document_root	
$document_uri	
$fastcgi_script_name	
$fastcgi_path_info	
$gzip_ratio	
$host	
$hostname	$hostname
$http2	
$http3	
$http_<name>	
$https	
$invalid_referer	
$is_args	
$limit_conn_status	$limit_conn_status
$limit_rate	
$limit_req_status	
$memcached_key	
$modern_browser	
	$mqtt_preread_clientid
	$mqtt_preread_username
$msec	$msec
$msie	
$p8s_value	
$pid	$pid
$pipe	
$proxy_add_x_forwarded_for	
$proxy_host	
$proxy_port	
	$protocol
$proxy_protocol_addr	$proxy_protocol_addr
$proxy_protocol_port	$proxy_protocol_port
$proxy_protocol_server_addr	$proxy_protocol_server_addr
$proxy_protocol_server_port	$proxy_protocol_server_port
$proxy_protocol_tlv_<name>	$proxy_protocol_tlv_<name>
$query_string	
$quic_connection	

续下页

表 2 – 接上页

HTTP 模块	*Stream* 模块
	$rdp_cookie, *$rdp_cookie_<name>*
$realip_remote_addr	*$realip_remote_addr*
$realip_remote_port	*$realip_remote_port*
$realpath_root	
$remote_addr	*$remote_addr*
$remote_port	*$remote_port*
$remote_user	
$request	
$request_body	
$request_body_file	
$request_completion	
$request_filename	
$request_id	
$request_length	
$request_method	
$request_time	
$request_uri	
$scheme	
$secure_link	
$secure_link_expires	
$sent_http_<name>	
$sent_trailer_<name>	
$server_addr	*$server_addr*
$server_name	
$server_port	*$server_port*
$server_protocol	
	$session_time
$slice_range	
$ssl_alpn_protocol	*$ssl_alpn_protocol*
$ssl_cipher	*$ssl_cipher*
$ssl_ciphers	*$ssl_ciphers*
$ssl_client_escaped_cert	
	$ssl_client_cert
$ssl_client_fingerprint	*$ssl_client_fingerprint*
$ssl_client_i_dn	*$ssl_client_i_dn*
$ssl_client_i_dn_legacy	
$ssl_client_raw_cert	*$ssl_client_raw_cert*
$ssl_client_s_dn	*$ssl_client_s_dn*
$ssl_client_s_dn_legacy	
$ssl_client_serial	*$ssl_client_serial*
$ssl_client_v_end	*$ssl_client_v_end*

续下页

表 2 – 接上页

HTTP 模块	Stream 模块
$ssl_client_v_remain	$ssl_client_v_remain
$ssl_client_v_start	$ssl_client_v_start
$ssl_client_verify	$ssl_client_verify
$ssl_curve	$ssl_curve
$ssl_curves	$ssl_curves
$ssl_early_data	$ssl_early_data
	$ssl_preread_alpn_protocols
	$ssl_preread_protocol
	$ssl_preread_server_name
$ssl_protocol	$ssl_protocol
$ssl_server_name	$ssl_server_name
$ssl_server_cert_type	$ssl_server_cert_type
$ssl_session_id	$ssl_session_id
$ssl_session_reused	$ssl_session_reused
$status	$status
$sticky_sessid	$sticky_sessid
$sticky_sid	$sticky_sid
$time_iso8601	$time_iso8601
$time_local	$time_local
$tcpinfo_rtt, $tcpinfo_rttvar, $tcpinfo_snd_cwnd, $tcpinfo_rcv_space	
$uid_got	
$uid_reset	
$uid_set	
$upstream_addr	$upstream_addr
$upstream_bytes_received	$upstream_bytes_received
$upstream_bytes_sent	$upstream_bytes_sent
$upstream_cache_status	
$upstream_connect_time	$upstream_connect_time
$upstream_cookie_<name>	
	$upstream_first_byte_time
$upstream_header_time	
$upstream_http_<name>	
$upstream_probe (PRO)	$upstream_probe (PRO)
$upstream_probe_body (PRO)	
	$upstream_probe_response (PRO)
$upstream_response_length	
$upstream_response_time	
	$upstream_session_time
$upstream_status	
$upstream_sticky_status	$upstream_sticky_status

续下页

表 2 – 接上页

HTTP 模块	*Stream* 模块
$upstream_trailer_<name>	
$upstream_queue_time	
$uri	

您还可以使用 https://angie.ws/en/ 的短链接服务，快速引用各个指令和变量：

3.2.3 快速访问 Angie 指令和变量

您可以通过我们的短链接服务 https://angie.ws/en/ 快速访问所有 Angie 指令和变量的文档，而无需在网站上搜索。它提供了对常用指令、变量和主题的快捷方式。

HTTP 和核心指令

在*core settings* 和*HTTP modules* 下的指令使用 `/h/` 前缀（即 `http` 的缩写）。

示例：

- *listen*: https://angie.ws/en/h/listen
- *server*: https://angie.ws/en/h/server
- *worker_connections*: https://angie.ws/en/h/worker_connections

等等。

> **备注**
>
> *Upstream* 模块中的*server* 指令位于 https://angie.ws/en/hu/server。

上游指令

Upstream 模块中的指令使用 `/hu/` 前缀（即 `http upstream` 的缩写）。示例：

- *keepalive_requests*: https://angie.ws/en/hu/keepalive_requests
- *keepalive_time*: https://angie.ws/en/hu/keepalive_time
- *keepalive_timeout*: https://angie.ws/en/hu/keepalive_timeout

等等。

流模块指令

stream modules 中的指令使用 `/s/` 前缀（即 `stream` 的缩写）：

- *listen*: https://angie.ws/en/s/listen
- *map*: https://angie.ws/en/s/map
- *server*: https://angie.ws/en/s/server

等等。

> **备注**
>
> *Upstream* 模块中的*server* 指令位于 https://angie.ws/en/su/server。

上游指令

Upstream 模块中的指令使用 `/su/` 前缀（即 `stream upstream` 的缩写）：

- *upstream*: https://angie.ws/en/su/upstream
- *server*: https://angie.ws/en/su/server
- *state (PRO)*: https://angie.ws/en/su/state

等等。

变量

变量遵循相同的前缀方案。

HTTP 变量（`/h/` 前缀）：

- *$angie_version*: https://angie.ws/en/h/\protect\TU\textdollarangie_version
- *$upstream_status*: https://angie.ws/en/h/\protect\TU\textdollarupstream_status

流变量（`/s/` 前缀）：

- *$angie_version*: https://angie.ws/en/s/\protect\TU\textdollarangie_version
- *$upstream_session_time*: https://angie.ws/en/s/\protect\TU\textdollarupstream_session_time

对于占位符变量，如 `$http_<HEADER>` 或 `$cookie_<NAME>`，请使用下划线之前的前缀：https://angie.ws/en/h/\protect\TU\textdollarhttp。

其他主题

短链接也可用于其他主题：

- 证书链
- 组合位置
- 紧凑服务器
- 配置
- 配置文件重新加载
- 配置哈希
- 控制配置更改
- 控制信号

- 循环内存缓冲区
- 调试日志
- 虚拟服务器
- HTTPS 配置
- HTTPS 优化
- 使用单独 IP 的 HTTPS
- HTTP 会话
- 继承
- 负载均衡
- 位置重定向
- 日志轮换
- 方法使用
- 命名位置
- 路径
- 选择位置
- 代理传递 URI
- 代理
- 请求处理
- 运行时 CLI 选项
- 服务升级
- SNI（服务器名称指示）
- 源构建特性
- SSL 错误代码
- 流会话
- 语法
- Syslog 日志
- 虚拟服务器选择
- WebSocket 代理

3.3 操作指南

此处提供了有关配置 Angie 特定方面的步骤说明。

3.3.1 如何从 nginx 迁移到 Angie

如果您正在从 nginx 切换到 Angie，恭喜！我们为您准备了一份指南。

请注意，该指南是为依赖 Angie 软件包版本的基本替换场景量身定制的。如果您正在处理:ref: 容器 *<docker_images>*、虚拟机、自定义路径或:ref: 模块 *<install-dynamicmodules>*，您需要相应地进行调整。

安装 Angie

我们建议使用 我们的仓库中的官方软件包；请参阅您操作系统的安装说明以安装 Angie，但在实际启动之前请停止。相反，运行 **`sudo angie -V`** 以确保一切正常：

```
$ sudo angie -V

  Angie version: Angie/1.10.0
  nginx version: nginx/1.27.5
  built by gcc 11.4.0
  configure arguments: --prefix=/etc/angie --conf-path=/etc/angie/angie.conf ...
```

正如此输出所示，当 Angie 从软件包安装时，配置 位于 `/etc/angie/` 下。

更新 Angie 配置

Angie 通常需要对现有的 nginx 配置进行最小的更改。

1. 将整个 nginx 配置复制到 `/etc/angie/`：

   ```
   $ sudo rsync -a --no-links /etc/nginx/ /etc/angie/
   ```

   我们假设 nginx 将其配置存储在 `/etc/nginx/` 下；如果您的路径不同，请调整步骤。

2. 按 Angie 期望的方式重命名主配置文件：

   ```
   $ sudo mv /etc/angie/nginx.conf /etc/angie/angie.conf
   ```

3. 更新整个 Angie 配置中的路径，从主配置文件开始。细节根据 nginx 的安装方式而有所不同，但至少您需要更新以下部分。

   任何仍指向 `/etc/nginx/` 下的*include* 路径：

   ```
   # include /etc/nginx/conf.d/*.conf;
   # include /etc/nginx/default.d/*.conf;
   # include /etc/nginx/http.d/*.conf;
   # include /etc/nginx/stream.d/*.conf;
   include /etc/angie/conf.d/*.conf;
   include /etc/angie/default.d/*.conf;
   ```

```
include /etc/angie/http.d/*.conf;
include /etc/angie/stream.d/*.conf;

# include /etc/nginx/sites-enabled/*;
include /etc/angie/sites-enabled/*;

# include /etc/nginx/modules-enabled/*;
include /etc/angie/modules-enabled/*;

# include /etc/nginx/mime.types;
include /etc/angie/mime.types;
```

对 Angie 进程管理至关重要的*PID* 文件：

```
# pid /var/run/nginx.pid;
# -- 或 --
# pid /run/nginx.pid;
pid /run/angie.pid;
```

最后，访问日志 和错误日志：

```
# access_log /var/log/nginx/access.log;
access_log /var/log/angie/access.log;

# error_log /var/log/nginx/error.log;
error_log /var/log/angie/error.log;
```

虚拟主机

如果使用类似 `sites-enabled/` 的目录来包含虚拟主机，请更新它：

```
# include /etc/nginx/sites-enabled/*;
include /etc/angie/sites-enabled/*;
```

接下来，重新创建 `/etc/angie/sites-enabled/` 下的符号链接以使其工作。

列出原始虚拟主机文件，例如：

```
$ ls -l /etc/nginx/sites-enabled/

  default -> /etc/nginx/sites-available/default
```

注意它们的实际位置；在这里，它是 `/etc/nginx/sites-available/`。

如果您之前没有将它们复制到 `/etc/angie/` 下，现在复制它们：

```
$ sudo rsync -a /etc/nginx/sites-available/ /etc/angie/sites-available/
```

最后，重新创建每个符号链接：

```
$ sudo ln -s /etc/angie/sites-available/default \
             /etc/angie/sites-enabled/default
```

动态模块

查找并 安装所有在 nginx 配置中引用的动态模块的 Angie 对应模块，例如：

```
$ sudo nginx -T | grep load_module

  load_module modules/ngx_http_geoip2_module.so;
  load_module modules/ngx_stream_geoip2_module.so;
  ...
```

这意味着您需要安装 angie-module-geoip2 软件包，等等。

有两种流行的方法来包含动态模块配置：

/usr/share/nginx/modules/

如果通过 /usr/share/nginx/modules/ 包含动态模块，更新路径：

```
# 加载动态模块。请参阅/usr/share/doc/nginx/README.dynamic。
# include /usr/share/nginx/modules/*.conf;

include /usr/share/angie/modules/*.conf;
```

接下来，复制模块配置文件：

```
$ sudo rsync -a /usr/share/nginx/modules/ /usr/share/angie/modules/
```

最后，修改每个文件中的*load_module* 路径：

```
# load_module "/usr/lib64/nginx/modules/ngx_http_geoip2_module.so";
load_module "/usr/lib64/angie/modules/ngx_http_geoip2_module.so";
```

/etc/nginx/modules-enabled/

如果使用 /etc/nginx/modules-enabled/ 目录来包含动态模块，请更新路径：

```
# include /etc/nginx/modules-enabled/*.conf;
include /etc/angie/modules-enabled/*.conf;
```

接下来，重新创建 /etc/angie/modules-enabled/ 下的符号链接以使其工作。

列出原始模块配置文件，例如：

```
$ ls -l /etc/nginx/modules-enabled/

  mod-http-geoip2.conf -> /usr/share/nginx/modules-available/mod-http-geoip2.conf
```

注意它们的实际位置；在这里，它是 /usr/share/nginx/modules-available/。

复制它们到 /usr/share/angie/：

```
$ sudo rsync -a /usr/share/nginx/modules-available/ /usr/share/angie/modules-available/
```

最后，重新创建每个符号链接：

```
$ sudo ln -s /usr/share/angie/modules-available/mod-http-geoip2.conf \
             /etc/angie/modules-enabled/mod-http-geoip2.conf
```

根目录（可选）

如果*root* 指向 /usr/share/nginx/html/ 目录，您可以将指令更改为指向 Angie。

复制目录并更新 Angie 配置中的 root 值：

```
$ sudo rsync -a /usr/share/nginx/html/ /usr/share/angie/html/
```

```
# root /usr/share/nginx/html;
root /usr/share/angie/html;
```

用户和组（可选）

虽然保持*user* 指令不变就足够了，但您可以使用 Angie 账户以获得灵活性。

更新 Angie 配置中的 user 设置：

```
# user www-data www-data;
user angie angie;
```

更改 * 所有 * 配置文件的所有者，包括 /usr/share/angie/ 中的文件，例如：

```
$ sudo chown -R angie:angie /etc/angie/
$ sudo chown -R angie:angie /usr/share/angie/
```

如果 Angie 配置有*root* 指令，更改那里指定的目录的所有者，例如：

```
$ sudo chown -R angie:angie /var/www/html/
```

总结

为了确保没有遗漏任何内容，查找并修复 Angie 配置中剩余的 nginx 提及：

```
$ grep -rn --include='*.conf' 'nginx' /etc/angie/
```

测试与切换

更新 Angie 配置后，下一步是检查其语法以确保 Angie 可以使用它，然后进行切换。验证 Angie 接受新配置：

```
$ sudo angie -t
```

此命令解析配置并报告会阻止 Angie 启动的错误；修复任何问题并重新运行命令。

停止 nginx，启动 Angie

为了最小化停机时间，在停止 nginx 后立即启动 Angie：

```
$ sudo systemctl stop nginx && sudo systemctl start angie
```

如果需要，启用 Angie 服务以在重启后启动：

```
$ sudo systemctl enable angie
```

迁移完成！就是这样；您太棒了。

禁用 nginx

确认 Angie 稳定运行后，您可以禁用或删除 nginx 以避免冲突。

您至少可以禁用服务：

```
$ sudo systemctl disable nginx
```

配置 Angie 功能

可以假设您迁移是有原因的。为什么不更进一步，配置一些 Angie 和 Angie PRO 中 nginx 没有的额外功能呢?

3.3.2 ACME 配置

Angie 中的*ACME* 模块能够通过 ACME 协议 自动获取证书。ACME 协议支持各种域验证方法（也称为" 验证"）；此模块实现了*HTTP* 验证、*DNS* 验证 和基于钩子的验证，通过自定义外部服务。

配置步骤

启用证书请求的常规步骤：

- **在:samp:‘http‘ 块中配置 ACME 客户端**，使用*acme_client* 指令，指定唯一的客户端名称和其他参数。可以配置多个 ACME 客户端。
- **指定请求证书的域**：对于在所有使用*acme* 的 `server` 块中的*server_name* 指令列出的所有域名，将颁发一个单一的证书，指向同一个 ACME 客户端。
- **设置请求处理和 ACME 回调**：这对于验证域所有权是必需的。设置取决于选择的域验证方法：

方法	用户要求	多域	通配符域
HTTP 验证	在 Angie 服务器上开放 80 端口以接收连接。	_	
DNS 验证	在 Angie 服务器上开放 53 端口（或*acme_dns_port* 指定的端口）以接收连接。 为 `_acme-challenge.` 子域设置指向您 Angie 服务器的 NS 记录。	_	_
基于钩子的验证	创建一个外部服务（脚本或应用程序),可以根据 Angie 的请求更新 DNS 记录或通过 Web 服务器提供特殊响应。	_	_

- **使用获得的证书和密钥配置 SSL**：模块使证书和密钥作为嵌入变量 可用，可以在配置 中用于填充*ssl_certificate* 和*ssl_certificate_key*。

  有关 SSL 设置说明，请参见*SSL 配置*。

> **小技巧**
>
> 证书获取和更新过程依赖于多个服务的运行，可能需要一些时间。请保持耐心，如有问题或疑问，请查阅调试日志。

实现细节

客户端密钥和证书以 PEM 编码 存储在由 `--http-acme-client-path` 构建选项指定的目录的相应子目录中：

```
$ ls /var/lib/angie/acme/example/

  account.key   certificate.pem   private.key
```

ACME 客户端需要在 CA 服务器上有一个账号。它使用私钥 (`account.key`) 创建和管理该账号；如果没有密钥，则在启动时生成。然后客户端使用此密钥在服务器上注册该账号。

> **备注**
>
> 如果您已经有一个账号密钥，请在启动前将其放置在客户端的子目录中，以便重用该账号。也可以使用*acme_client* 中的 `account_key` 参数指定该密钥文件。

ACME 客户端还使用一个单独的密钥 (`private.key`) 用于证书签名请求（CSR）；如果需要，此证书密钥也会在启动时自动创建。

启动时，如果尚无证书，客户端会为其管理的所有域签名并发送 CSR 到 CA 服务器请求证书。服务器通

过*HTTP* 或*DNS* 验证 验证域所有权，并颁发证书，客户端将其保存在本地 (`certificate.pem`)。

如前所述，使用同一个 ACME 客户端的所有域名将获得一个单一证书，因此可能是多域证书。所有包含的名称列表可以在证书的 主题备用名称（SAN）部分中找到。您可以在命令行中检查此信息：

```
$ openssl x509 -in certificate.pem -noout -text | grep -A5 "Subject Alternative Name"
```

当证书即将到期或域列表发生变化时，客户端会签名并发送另一个 CSR 到 CA 服务器。服务器再次验证所有权并颁发新证书，客户端将其安装在本地，替换之前的证书。

在配置 中，获取的证书及其对应的密钥可通过前缀变量*$acme_cert_<名称>*和*$acme_cert_key_<名称>*访问。它们的值是相应文件的内容，应与*ssl_certificate* 和*ssl_certificate_key* 指令一起使用：

```
server {

    listen 443 ssl;

    server_name example.com www.example.com;
    acme example;

    ssl_certificate $acme_cert_example;
    ssl_certificate_key $acme_cert_key_example;
}
```

HTTP 验证

验证是自动处理的。此过程涉及 ACME 服务器，在收到请求后，通过 HTTP 检索特殊文件令牌 ，地址为 `/.well-known/acme-challenge/<TOKEN>`。我们的 ACME 模块拦截此类请求并自动处理它们。当收到带有正确内容的预期响应时，ACME 服务器确认域所有权。

配置示例

在此示例中，名为 `example` 的 ACME 客户端管理 `example.com` 和 `www.example.com` 的证书（请注意，HTTP 验证不支持通配符证书）：

```
http {

    resolver 127.0.0.53; # acme_client指令所需

    acme_client example https://acme-v02.api.letsencrypt.org/directory;

    server {

        listen 80; # 可能位于不同的'server'块中
        # 具有不同的域列表
        # 或甚至没有

        listen 443 ssl;
```

```
        server_name example.com www.example.com;
        acme example;

        ssl_certificate $acme_cert_example;
        ssl_certificate_key $acme_cert_key_example;
    }
}
```

如前所述，80 端口必须开放以处理 HTTP ACME 回调。然而，如本示例所示，针对此端口的*listen* 指令可以放置在单独的*server* 块中。如果没有现有的包含此指令的块，可以创建一个仅限于 ACME 回调的新块：

```
server {

    listen 80;
    return 444; # 无响应，连接关闭
}
```

为什么这样做有效？模块在读取头信息后但在选择虚拟服务器和处理*rewrite* 和*location* 指令之前拦截对 `/.well-known/acme-challenge/<TOKEN>` 的请求。如果 `<TOKEN>` 的值与特定回调的预期值匹配，则处理此类拦截请求。不会访问目录；请求完全由模块处理。

DNS 验证

验证是自动处理的。当处理证书请求时，ACME 服务器对正在验证的域的 `_acme-challenge.` 子域执行特殊 DNS 查询 。收到预期响应后，ACME 服务器确认域所有权。

我们的 ACME 模块自动跟踪和处理此类请求，前提是您的 DNS 记录已正确配置，将 Angie 服务器指定为 `_acme-challenge.` 子域的权威名称服务器。

例如，要使用 IP 地址为 `93.184.215.14` 的 Angie 服务器验证域 `example.com`，您的域的 DNS 配置应包含以下记录：

```
_acme-challenge.example.com. 60    IN      NS       ns.example.com.
            ns.example.com. 60    IN       A       93.184.215.14
```

此配置将 `_acme-challenge.example.com` 的 DNS 解析委托给 `ns.example.com`，确保 `ns.example.com` 可通过其 IP 地址 (`93.184.215.14`) 访问。

此方法允许请求通配符证书，例如在 主题备用名称（SAN）部分包含 `*.example.com` 的证书。要明确请求子域的证书，如 `www.example.com`，您必须使用上述方法单独验证该子域。

> **注意**
>
> 此场景的适用性在很大程度上取决于您的 DNS 提供商的功能；某些提供商不允许此类配置。

配置示例

配置通常与前一节中的示例相似。无需 HTTP 特定设置；相反，在*acme_client* 指令中设置 `challenge=dns`。

在此示例中，名为 `example` 的 ACME 客户端管理 `example.com` 和 `*.example.com` 的证书：

```
http {

    resolver 127.0.0.53;

    acme_client example https://acme-v02.api.letsencrypt.org/directory
        challenge=dns;

    server {

        server_name example.com *.example.com;
        acme example;

        ssl_certificate $acme_cert_example;
        ssl_certificate_key $acme_cert_key_example;
    }
}
```

基于钩子的验证

与前面的方法不同，此验证需要额外的工作。这里，ACME 服务器执行常规的*HTTP 验证* 或*DNS 验证*，但不是直接与 Angie 服务器通信，而是与由 Angie 服务器通过钩子调用 (*acme_hook*) 管理的外部服务通信。这个服务反过来配置一个单独的 DNS 或 HTTP 服务器，ACME 服务器向其发送请求。

一旦 ACME 服务器从这样配置的 DNS 或 HTTP 服务器收到预期响应，它确认域所有权。

Angie 在需要通过 ACME 协议更新证书时调用外部服务。调用通过使用*fastcgi_pass*、*scgi_pass* 等指令将请求和数据代理到 FastCGI、SCGI 或类似服务器来传递。

该服务必须返回 `2xx` 代码，可以通过 `Status` 头完成。任何其他代码都被视为错误，证书更新将中止。服务的输出将被忽略。

配置示例

此处配置了 ACME 客户端 `example` 用于通过 DNS 调用进行域验证，这由 `acme_client` 指令中的 `challenge=dns` 参数指示。

`server` 块适用于所有 `example.com` 子域（例如 `*.example.com`），并使用 ACME 客户端 `example` 管理证书，如 `acme` 指令所指定。

还配置了一个命名的 `location` 块，通过它进行 DNS 验证的外部服务调用。`acme_hook` 指令将服务器与 ACME 客户端 `example` 关联。请求通过 `fastcgi_pass` 指令代理到在本地 9000 端口运行的 FastCGI 服务器。`fastcgi_param` 指令将有关 ACME 客户端、钩子、调用类型、域、令牌和授权密钥的数据传递给外部服务。

```
acme_client example https://acme-v02.api.letsencrypt.org/directory
    challenge=dns;

server {

    listen 80;

    server_name *.example.com;

    acme example;

    ssl_certificate $acme_cert_example;
    ssl_certificate_key $acme_cert_key_example;

    location @acme_hook_location {

        acme_hook example;

        fastcgi_pass localhost:9000;

        fastcgi_param ACME_CLIENT $acme_hook_client;
        fastcgi_param ACME_HOOK $acme_hook_name;
        fastcgi_param ACME_CHALLENGE $acme_hook_challenge;
        fastcgi_param ACME_DOMAIN $acme_hook_domain;
        fastcgi_param ACME_TOKEN $acme_hook_token;
        fastcgi_param ACME_KEYAUTH $acme_hook_keyauth;

        include fastcgi.conf;
    }
}
```

以下是一个相应的 Perl 外部 FastCGI 服务示例：

```
#!/usr/bin/perl

use strict; use warnings;

use FCGI;

my $socket = FCGI::OpenSocket(":9000", 5);
my $request = FCGI::Request(\*STDIN, \*STDOUT, \*STDERR, \%ENV, $socket);

while ($request->Accept() >= 0) {
    print "\r\n";

    my $client =    $ENV{ACME_CLIENT};
    my $hook =      $ENV{ACME_HOOK};
    my $challenge = $ENV{ACME_CHALLENGE};
```

```
    my $domain =    $ENV{ACME_DOMAIN};
    my $token =     $ENV{ACME_TOKEN};
    my $keyauth =   $ENV{ACME_KEYAUTH};

    if ($hook eq 'add') {

        DNS_set_TXT_record("_acme-challenge.$domain.", $keyauth);

    } elsif ($hook eq 'remove') {

        DNS_clear_TXT_record("_acme-challenge.$domain.");
    }
};

FCGI::CloseSocket($socket);
```

这里的 `DNS_set_TXT_record()` 和 `DNS_clear_TXT_record()` 是假定用于在某个外部 DNS 服务器的配置中添加和删除 TXT 记录的函数，ACME 服务器将查询该服务器。这些记录必须包含由 Angie 服务器传递的数据，以允许外部 DNS 服务器成功通过类似于*DNS 验证* 部分所述的验证。此类函数的实现细节超出了本指南的范围；例如，也可以通过请求 URI 传递参数：

```
# ...

location @acme_hook_location {

    acme_hook example uri=/acme_hook/$acme_hook_name?domain=$acme_hook_domain&key=$acme_hook_
↪keyauth;

    fastcgi_pass localhost:9000;

    fastcgi_param REQUEST_URI $request_uri;
    fastcgi_param ACME_CLIENT $acme_hook_client;
    fastcgi_param ACME_CHALLENGE $acme_hook_challenge;
    fastcgi_param ACME_TOKEN $acme_hook_token;

    include fastcgi.conf;
}
```

另一个使用 PHP-FPM 配置 FastCGI 服务的示例：

```
location @acme_hook_location {

    acme_hook example;
    root /var/www/dns;
    fastcgi_pass unix:/run/php-fpm/php-dns.sock;
    fastcgi_index hook.php;
    fastcgi_param SCRIPT_FILENAME /var/www/dns/hook.php;
    include fastcgi_params;
```

```
    fastcgi_param ACME_CLIENT $acme_hook_client;
    fastcgi_param ACME_HOOK $acme_hook_name;
    fastcgi_param ACME_CHALLENGE $acme_hook_challenge;
    fastcgi_param ACME_DOMAIN $acme_hook_domain;
    fastcgi_param ACME_TOKEN $acme_hook_token;
    fastcgi_param ACME_KEYAUTH $acme_hook_keyauth;
}
```

```
[dns]
listen = /run/php-fpm/php-dns.sock
listen.mode = 0666
user = angie
group = angie
chdir = /var/www/dns
# ...
```

传递的参数可以在 PHP 中通过 `$_SERVER['...']` 访问。

流模块中的 ACME

流模块*ACME* 支持为 TCP 流量自动颁发和使用证书。为了正常运行，必须首先配置 HTTP 等效项：ACME 客户端必须在 `http` 上下文中声明，并且 `stream` 块必须放置在配置中的 `http` 块 * 之后 *。

配置示例

默认情况下，使用 HTTP 验证来获取证书。如*HTTP 验证* 中所述，这需要一个监听 80 端口的 HTTP 服务器：

```
# HTTP部分
http {

    resolver 127.0.0.53;

    # 用于stream部分的ACME客户端
    acme_client example https://acme-v02.api.letsencrypt.org/directory;

    # 用于HTTP验证的服务器
    server {

        listen 80;
        return 444;
    }
}

# Stream部分
stream {
```

```
    server {

        listen 12345 ssl;
        proxy_pass backend_upstream;

        ssl_certificate $acme_cert_example;
        ssl_certificate_key $acme_cert_key_example;

        server_name example.com www.example.com;
        acme example; # 引用HTTP部分中定义的ACME客户端
    }

    upstream backend_upstream {

        server 127.0.0.1:54321;
    }
}
```

也可以通过在*acme_client* 指令中配置 `challenge=dns` 来使用 DNS 验证；在这种情况下，不需要服务器。

从 certbot 迁移

如果在从 *nginx* 迁移到 *Angie* 之前，您使用 certbot 从 Let's Encrypt 获取和续订 SSL 证书，请按照以下步骤迁移到使用我们的 ACME 模块。

假设您最初使用以下命令配置证书：

```
$ sudo certbot --nginx -d example.com -d www.example.com
```

自动创建的配置通常位于 `/etc/nginx/sites-available/example.conf`，大致如下：

```
server {

    listen 80;
    server_name example.com www.example.com;
    return 301 https://$host$request_uri;
}

server {

    listen 443 ssl;
    server_name example.com www.example.com;

    root /var/www/example;
    index index.html;

    ssl_certificate /etc/letsencrypt/live/example.com/fullchain.pem;
    ssl_certificate_key /etc/letsencrypt/live/example.com/privkey.pem;
```

```
    include /etc/letsencrypt/options-ssl-nginx.conf;
    ssl_dhparam /etc/letsencrypt/ssl-dhparams.pem;

    location / {

        try_files $uri $uri/ =404;
    }
}
```

在上面的示例中，突出显示的行需要修改。根据您的情况和偏好，使用 ACME 模块配置*HTTP 验证* 或*DNS 验证*。

最终的*Angie 配置* 可能如下所示：

```
http {

    resolver 127.0.0.53;

    acme_client example https://acme-v02.api.letsencrypt.org/directory;

    server {

        listen 80;
        server_name example.com www.example.com;
        return 301 https://$host$request_uri;
    }

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

        root /var/www/example;
        index index.html;

        acme                 example;

        ssl_certificate      $acme_cert_example;
        ssl_certificate_key  $acme_cert_key_example;

        location / {
            try_files $uri $uri/ =404;
        }
    }
}
```

记得在更改后重新加载配置：

```
$ sudo kill -HUP $(cat /run/angie.pid)
```

确认此配置正常工作后，您可以删除 `certbot` 证书，并且如果不再在其他地方使用，可以禁用或完全删除 certbot，例如：

```
$ sudo rm -rf /etc/letsencrypt

$ sudo systemctl stop certbot.timer
$ sudo systemctl disable certbot.timer
$ # -- 或 --
$ sudo rm /etc/cron.d/certbot

$ sudo apt remove certbot
$ # -- 或 --
$ sudo dnf remove certbot
```

3.3.3 SSL 配置

要配置 HTTPS 服务器，必须在*server* 块的监听套接字上启用*ssl* 参数，并且需要指定服务器证书和私钥文件的位置：

```
server {
    listen              443 ssl;
    server_name         www.example.com;
    ssl_certificate     www.example.com.crt;
    ssl_certificate_key www.example.com.key;
    ssl_protocols       TLSv1.2 TLSv1.3;
    ssl_ciphers         HIGH:!aNULL:!MD5;
#...
}
```

服务器证书是一个公共实体。它会发送给每个连接到服务器的客户端。私钥是一个安全实体，应存储在访问受限的文件中；但它必须对 Angie 的主进程可读。私钥也可以与证书存储在同一个文件中。

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

在这种情况下，文件的访问权限也应受到限制。即使证书和密钥存储在一个文件中，只有证书会发送给客户端。

指令*ssl_protocols* 和*ssl_ciphers* 可用于限制连接，仅包含 SSL/TLS 的强版本和密码。默认情况下，Angie 使用：

```
ssl_protocols TLSv1.2 TLSv1.3;
ssl_ciphers HIGH:!aNULL:!MD5;
```

因此，通常不需要显式配置它们。

HTTPS 服务器优化

SSL 操作消耗额外的 CPU 资源。在多处理器系统上，应运行多个*worker processes*，数量不能少于可用的 CPU 核心数。最耗 CPU 的操作是 SSL 握手。减少每个客户端的这些操作数量有两种方法：第一种是启用*keepalive* 连接，通过一个连接发送多个请求，第二种是重用 SSL 会话参数，以避免并行和后续连接的 SSL 握手。会话存储在一个由*ssl_session_cache* 指令配置的共享 SSL 会话缓存中。1MB 的缓存大约包含 4000 个会话。默认缓存超时为 5 分钟。可以使用*ssl_session_timeout* 指令增加。以下是一个针对具有 10MB 共享会话缓存的多核系统优化的配置示例：

```
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       TLSv1.2 TLSv1.3;
        ssl_ciphers         HIGH:!aNULL:!MD5;
    #...
```

SSL 证书链

某些浏览器可能会对由知名证书颁发机构签署的证书提出警告，而其他浏览器可能不会。这是因为颁发机构使用了一个中间证书来签署服务器证书，而该中间证书并不在某个特定浏览器分发的知名受信任证书机构的证书库中。在这种情况下，颁发机构提供了一组链式证书，这些证书应与签署的服务器证书串联在一起。服务器证书必须在组合文件中位于链式证书之前：

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

生成的文件应与*ssl_certificate* 指令一起使用：

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

如果服务器证书和捆绑证书的顺序连接错误，Angie 将无法启动并显示错误信息：

SSL_CTX_use_PrivateKey_file(" ... /www.example.com.key") failed
(SSL: error:0B080074:x509 certificate routines: X509_check_private_key:key values

mismatch)

因为 Angie 尝试使用捆绑证书的第一个证书而不是服务器证书来使用私钥。

浏览器通常会存储它们收到的由受信任机构签署的中间证书，因此实际使用的浏览器可能已经拥有所需的中间证书，并且可能不会对未发送链式捆绑的证书提出警告。为了确保服务器发送完整的证书链，可以使用 `openssl` 命令行工具，例如：

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

小技巧

在使用*SNI* 测试配置时，重要的是要指定 *-servername* 选项，因为 *openssl* 默认不使用 SNI。

在这个例子中，www.GoDaddy.com 服务器证书 #0 的主题（"s"）由一个颁发者（"i"）签署，该颁发者本身是证书 #1 的主题，该证书由一个颁发者签署，该颁发者本身是证书 #2 的主题，该证书由知名颁发者 ValiCert, Inc. 签署，其证书存储在浏览器的内置证书库中。

如果没有添加证书捆绑，只有服务器证书 #0 将会显示。

单个 HTTP/HTTPS 服务器

可以配置一个同时处理 HTTP 和 HTTPS 请求的单个服务器：

```
server {
    listen              80;
    listen              443 ssl;
    server_name         www.example.com;
```

```
    ssl_certificate     www.example.com.crt;
    ssl_certificate_key www.example.com.key;
#...
}
```

基于名称的 HTTPS 服务器

配置两个或多个在单个 IP 地址上监听的 HTTPS 服务器时，会出现常见问题：

```
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;
#...
}
```

使用此配置，浏览器会接收到默认服务器的证书，即 *www.example.com*，无论请求的服务器名称是什么。这是由 SSL 协议行为造成的。SSL 连接在浏览器发送 HTTP 请求之前建立，而 Angie 并不知道请求的服务器名称。因此，它只能提供默认服务器的证书。

解决此问题的最古老且最稳健的方法是为每个 HTTPS 服务器分配一个独立的 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;
#...
}
```

具有多个名称的 SSL 证书

还有其他方法可以在多个 HTTPS 服务器之间共享一个 IP 地址。然而，它们都有各自的缺点。一种方法是使用在 `SubjectAltName` 证书字段中包含多个名称的证书，例如 `www.example.com` 和 `www.example.org`。但是，`SubjectAltName` 字段的长度是有限的。

另一种方法是使用通配符名称的证书，例如 `*.example.org`。通配符证书保护指定域的所有子域，但仅限于一个级别。该证书匹配 `www.example.org`，但不匹配 `example.org` 和 `www.sub.example.org`。这两种方法也可以结合使用。证书可以在 `SubjectAltName` 字段中包含精确名称和通配符名称，例如 `example.org` 和 `*.example.org`。

最好将具有多个名称的证书文件及其私钥文件放在 `http` 配置级别，以便在所有服务器中继承它们的单一内存副本：

```
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;
#...
}
```

服务器名称指示

在单个 IP 地址上运行多个 HTTPS 服务器的更通用的解决方案是 TLS 服务器名称指示扩展（SNI，RFC 6066），该扩展允许浏览器在 SSL 握手期间传递请求的服务器名称，因此，服务器将知道它应该使用哪个证书进行连接。SNI 目前被大多数现代浏览器支持，但一些旧的或特殊的客户端可能不会使用。

> **小技巧**
>
> 仅可以在 SNI 中传递域名；然而，一些浏览器可能会错误地将服务器的 IP 地址作为其名称传递，如果请求包含文字 IP 地址。对此不应依赖。

如果 Angie 是使用 SNI 支持构建的，则在使用 `-V` 开关运行时，Angie 将显示此信息：

```
$ angie -V
...
TLS SNI support enabled
...
```

然而，如果启用 SNI 的 Angie 动态链接到没有 SNI 支持的 OpenSSL 库，Angie 将显示警告：

Angie was built with SNI support, however, now it is linked dynamically to an OpenSSL library which has no tlsext support, therefore SNI is not available

3.3.4 Console Light 网络监控面板

Angie 提供了广泛的监控其工作的可能性；除了*metrics* API 和*Prometheus* 模块外，您还可以使用安装在服务器旁边的可视化控制台。

Console Light

Console Light 是一个轻量级的实时活动监控界面，显示关键的服务器负载和性能指标。控制台基于 Angie 的*API 接口*；活动监控数据是实时生成的。此外，如果 API 本身提供此功能，控制台还允许您动态修改 Angie 配置。

已部署和配置的控制台示例：https://console.angie.software/

版本历史

版本	发布日期	变更内容
1.7.2	07.04.2025	在”HTTP/TCP/UDP Upstreams” 页面的过滤器控制器中添加了”busy（繁忙）” 选项。
1.7.1	04.04.2025	修复了”HTTP 区域” 页面中”HTTP/Location 区域” 表格的不正确值。
1.7.0	02.04.2025	• 鼠标悬停时以字节为单位显示精确的数据量 • 统计 API 中新增上游节点的”busy（繁忙）” 状态，表示该节点已达到由 `max_conns` 参数配置的连接数上限 • 修复了文档链接
1.6.1	27.01.2025	• 修复拼写错误 • 修复开发阶段的项目构建问题
1.6.0	23.01.2025	• 添加国际化支持，支持语言：`en`、`ru` • 表格组件新增固定表头功能 • 支持以 pebibytes（PiB）为单位的数据度量 • 修复首页中*HTTP Upstreams* 组件中错误的值计数 • 现在在*HTTP Upstreams* 页面中正确使用默认值作为响应上下文
1.5.0		未公开发布
1.4.0	08.08.2024	网站 favicon 图标新增监控状态显示功能
1.3.0	28.04.2024	在上游上下文中新增将服务器切换到 `draining` 状态的功能
1.2.1	26.12.2023	在 `Stream` 上下文中添加主动健康检查功能
1.2.0	25.12.2023	在 `Stream` 上下文中添加上游编辑功能

安装和配置

Console Light 以 `angie-console-light` (Angie) 和 `angie-pro-console-light` (Angie PRO) 的形式发布在 我们的存储库中，可以像任何其他包一样安装；您也可以从 我们的网站 或 GitHub 下载源码。

安装后，通过在服务器配置 的*server* 块内添加以下*location* 片段来配置控制台（注意注释）：

```
location /console/ {

    # 仅限本地访问
    allow 127.0.0.1;
    deny all;

    auto_redirect on;

    alias /usr/share/angie-console-light/html/;
    # 仅适用于 FreeBSD:
    # alias /usr/local/www/angie-console-light/html/;
    index index.html;

    location /console/api/ {
        api /status/;
    }

    # 在身份验证后使编辑功能正常工作（仅限 PRO）
    location /console/api/config/ {

        auth_basic           "Protected site";
        auth_basic_user_file conf/htpasswd;

        api /config/;
    }
}
```

不要忘记应用修改后的配置：

```
$ sudo angie -t && sudo service angie reload
```

之后，控制台将在 `server` 块指定的服务器上可用，路径在 `location` 中指定；上述路径设置为 `/console/`。

如上例所示，可以为任意 API 部分启用身份验证，例如：

```
location /console/server_zones/ {
    auth_basic           "Protected site";
    auth_basic_user_file conf/htpasswd;
}
```

您还可以限制对配置控制台 `location` 任意部分的访问，例如：

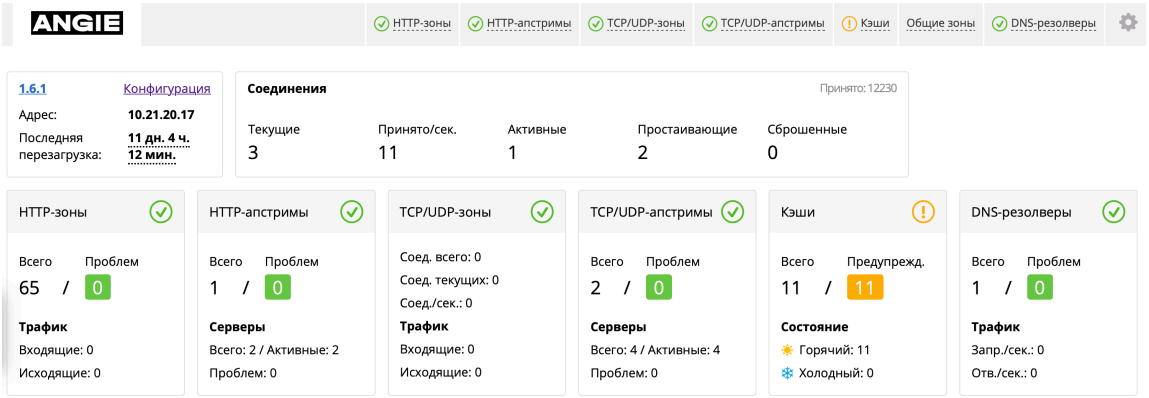
```
location /console/api/resolvers/ {
    deny all;
}
```

界面

控制台是一个单一屏幕，包含一组选项卡，每个选项卡包含几个监控数据小部件。

> **提示**
>
> 在下面的部分中，界面元素从左到右描述。

Angie 选项卡



这是主选项卡，显示基于多个 API 部分数据的主要 Angie 监控指标。

关于小部件

显示 Angie 版本号，并提供到相关文档的链接，以及服务器地址和最后一次配置重载 时间。

此外，如果启用了 *api_config_files* 指令，*Configs* 链接将打开服务器上加载的配置文件列表。每个文件都可以在紧凑视图中查看，并带有语法高亮。

连接小部件

显示从 API 部分 `/status/connections/` 生成的基本服务器连接统计数据：

当前	当前连接数
每秒接受	每秒接受的连接数
活动	活动连接数
空闲	空闲连接数
丢弃	丢弃的连接数

还可用：

接受	自上次服务器重启以来接受的总连接数

HTTP 区域小部件

> **注意**
>
> 该小部件需要在 `server` 或 `location` 上下文中设置*status_zone* 指令。

显示从 API 部分*/status/http/server_zones/* 生成的 `http` 上下文的共享内存区域统计数据：

总计	总区域数
问题	有问题的区域数
流量	总流入和流出流量

HTTP 上游小部件

> **注意**
>
> 该小部件需要在 `http` 上下文的*upstream* 块中设置*zone* 指令。

显示从 API 部分*/status/http/upstreams/* 生成的 `http` 上下文的上游统计数据：

总计	总上游数
问题	有问题的上游数
服务器	按状态划分的服务器统计

TCP/UDP 区域小部件

> **注意**
>
> 该小部件需要设置以下指令：
>
> - `status_zone` 在*server* 或*stream* 上下文中；
> - `limit_conn` 在*server* 或*stream* 上下文中；
> - *limit_conn_zone* 在 `stream` 上下文中。
>
> 示例：
> ```
> stream {
> ```

```
    # ...
    limit_conn_zone $connection zone=limit-conn-stream:10m;

    server {

        # ...
        limit_conn limit-conn-stream 1;
        status_zone foo;
    }
}
```

显示从 API 部分*/status/stream/server_zones/* 生成的 `stream` 上下文的共享内存区域统计数据：

连接总数	总客户端连接数
当前连接	当前客户端连接数
每秒连接	每秒处理的连接数

TCP/UDP 上游小部件

> **⚠ 注意**
>
> 该小部件需要在 `stream` 上下文的*upstream* 块中设置*zone* 指令。

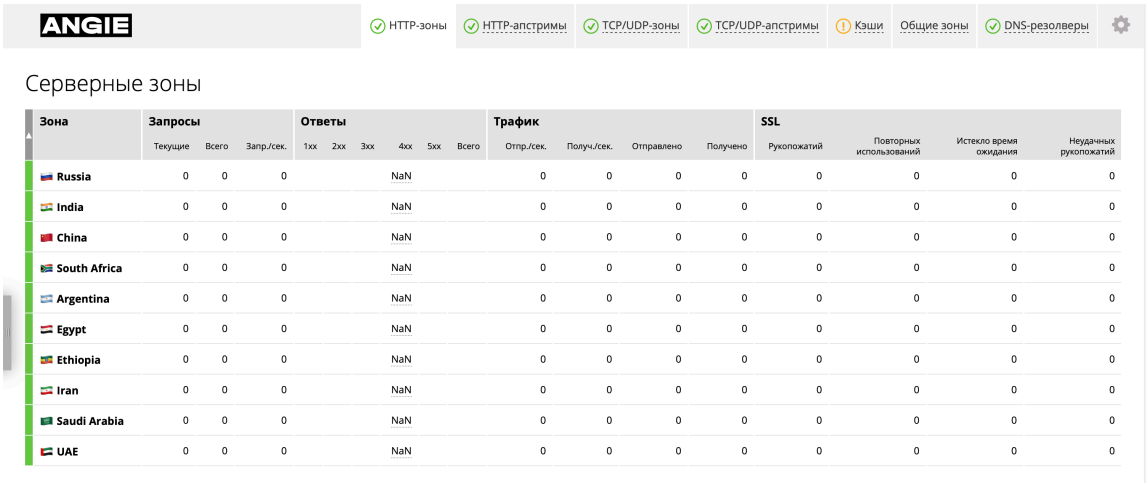
显示从 API 部分*/status/stream/upstreams/* 生成的 `stream` 上下文的上游统计数据：

总计	总上游数
问题	有问题的上游数
服务器	按状态划分的服务器统计

HTTP 区域选项卡

> **⚠ 注意**
>
> 该选项卡需要在 `server` 或 `location` 上下文中设置*status_zone* 指令。

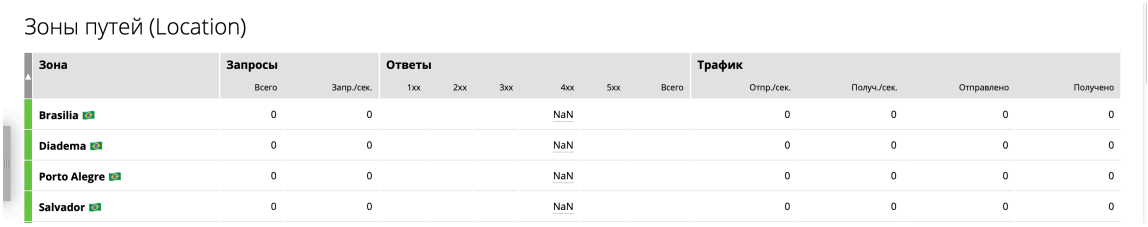
服务器区域部分



总结了从 API 部分*/status/http/server_zones/* 生成的 `http` 上下文中 `server` 上下文的共享内存区域监控统计数据。为每个区域提供以下数据：

区域	区域名称 **提示** 单击 区域旁边的箭头，可以按字母顺序或配置顺序对区域进行排序。
请求	请求总数以及每秒请求数
响应	按状态代码划分的响应数，以及响应总数
流量	出入流量率，以及出入流量总量
SSL	累计计数：成功的 SSL 握手数；SSL 会话重用数；超时过期的 SSL 握手数；不成功的 SSL 握手数

位置区域部分



总结了从 API 部分*/status/http/location_zones/* 生成的 `http` 上下文中 `location` 上下文的共享内存区域监控统计数据。为每个区域提供以下数据：

区域	区域名称 **提示** 单击 区域旁边的箭头，可以按字母顺序或配置顺序对区域进行排序。
请求	请求总数以及每秒请求数
响应	按状态代码划分的响应数，以及响应总数
流量	出入流量率，以及出入流量总量

连接限制区域部分

显示从 API 部分*/status/http/limit_conns/* 生成的 `http` 上下文中的 `limit_conn` 区域统计数据。为每个区域提供以下数据：

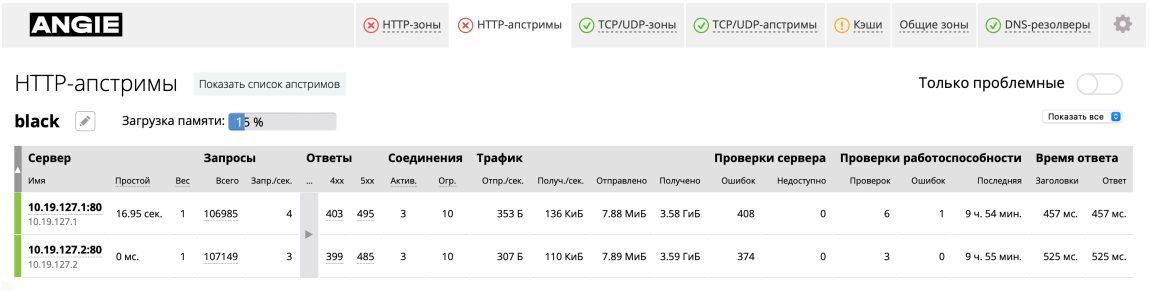
区域	区域名称 **提示** 单击 区域旁边的图标，以打开或关闭包含以下指标的图表。
通过	总代理连接数
拒绝	总拒绝连接数
耗尽	由于区域存储耗尽而丢弃的总连接数
跳过	通过的总连接数（键为零或大于 255 字节）

请求限制区域部分

显示从 API 部分*/status/http/limit_reqs/* 生成的 `http` 上下文中的 `limit_reqs` 区域统计数据。为每个区域提供以下数据：

区域	区域名称 **提示** 单击 区域旁边的图标，以打开或关闭包含以下指标的图表。
通过	总代理连接数
延迟	总延迟连接数
拒绝	总拒绝连接数
耗尽	由于区域存储耗尽而丢弃的总连接数
跳过	通过的总连接数（键为零或大于 255 字节）

HTTP 上游选项卡



注意

该选项卡需要在 `http` 上下文的*upstream* 块中设置*zone* 指令。

此选项卡总结了从 API 部分*/status/http/upstreams/* 生成的 `http` 上下文的上游监控统计数据。

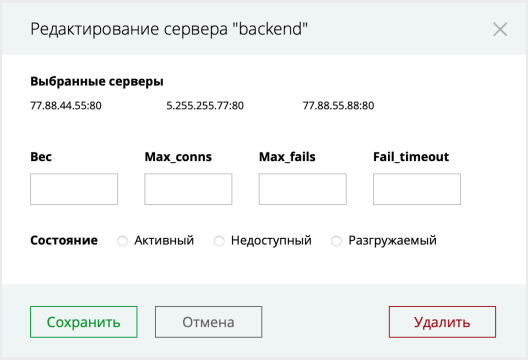
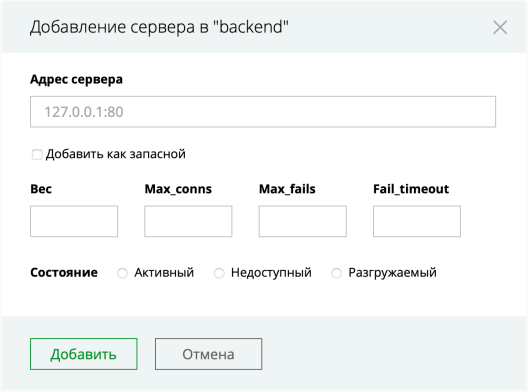
- 显示上游列表按钮切换上游摘要，显示有问题的上游和服务器数量。
- 仅显示失败开关切换为仅关注有问题上游的输出模式。
- 编辑按钮切换 上游编辑界面。
- 每个上游表右侧的下拉列表允许按状态过滤其服务器（正常，失败，检查中，下线，繁忙）。

对于每个上游，除了其名称和共享内存区域利用率外，还提供以下数据：

服务器	上游服务器的名称、停机时间和权重 **提示** 单击 服务器旁边的箭头，可以按状态或配置顺序对服务器进行排序。
请求	请求的总数和处理速度
响应	按状态代码划分的响应数
连接	活动连接数及其最大限制（如设置）
流量	出入流量率，以及出入流量总量
服务器检查	达到服务器的不成功尝试次数以及服务器被视为不可用的次数（API 中的 `health` 对象）
健康监测	服务器探测总数、不成功的探测次数以及最后一次探测的时间

编辑上游

在 Angie PRO 中，每个上游旁边有一个编辑按钮；单击后会出现两个更多按钮：

编辑选中的	编辑上游中选中的服务器。允许同时设置服务器的以下参数：权重，最大连接限制（Max_conns），将服务器视为不可用的最大失败次数（Max_fails），*max_fails* 的失败累积窗口（Fail_timeout），状态（活动 - 启用，下线 - 禁用或 正在排空 - 仅接收之前通过 sticky 绑定的会话请求）。 您还可以删除选中的服务器。 
添加服务器	向上游添加服务器。允许您设置以下参数：地址，是否备份，权重，最大连接限制（Max_conns），将服务器视为不可用的最大失败次数（Max_fails），失败累积窗口（Fail_timeout），状态（活动 - 启用，下线 - 禁用或 正在排空 - 仅接收之前通过 sticky 绑定的会话请求）。 

TCP/UDP 区域选项卡

注意

该选项卡需要设置以下指令：

- status_zone 在*server* 或*stream* 上下文中；
- limit_conn 在*server* 或*stream* 上下文中；
- *limit_conn_zone* 在 stream 上下文中。

示例：

```
stream {

    # ...
    limit_conn_zone $connection zone=limit-conn-stream:10m;

    server {

        # ...
        limit_conn limit-conn-stream 1;
        status_zone foo;
    }
}
```

TCP/UDP 区域部分

TCP/UDP-зоны

Зона	Соединения			Сессии				Трафик				SSL		
	Текущих	Всего	Соед./сек.	2xx	4xx	5xx	Всего	Отпр./сек.	Получ./сек.	Отправлено	Получено	Рукопожатий	Неудачных рукопожатий	Повторных использований
sing_chorus	3	403	0	372	0	28	400	0	0	4.41 ГБ	6.51 МБ	375	0	180

总结了从 API 部分*/status/stream/server_zones/* 生成的 `stream` 上下文中 `server` 上下文的共享内存区域监控统计数据。为每个区域提供以下数据：

区域	区域名称
连接	当前和总连接数，以及每秒连接数
会话	按状态代码划分的会话数，以及会话总数
流量	出入流量率，以及出入流量总量
SSL	累计计数：成功的 SSL 握手数；不成功的 SSL 握手数；SSL 会话重用数

连接限制区域部分

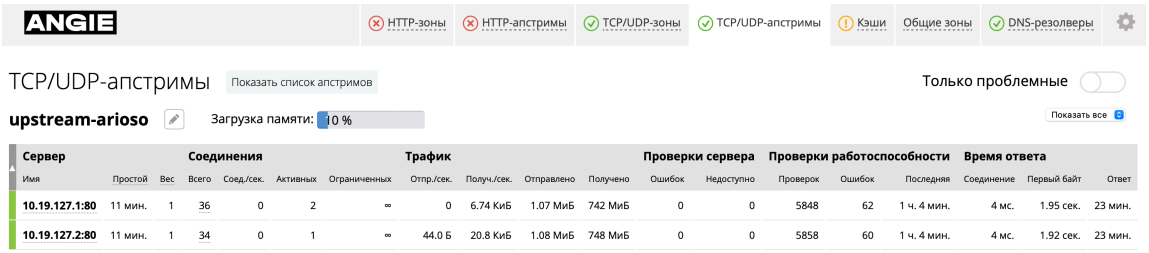
Зоны ограничения соединений (Limit Conn)

	Зона	Передано	Отклонено	Сброшено	Пропущено
	limit-conn-stream	403	0	0	0

显示从 API 部分*/status/stream/limit_conns/* 生成的 `stream` 上下文中的 `limit_conn` 区域统计数据。为每个区域提供以下数据：

区域	区域名称 **提示** 单击 区域旁边的图标，以打开或关闭包含以下指标的图表。
通过	总代理连接数
拒绝	总拒绝连接数
耗尽	由于区域存储耗尽而丢弃的总连接数
跳过	通过的总连接数（键为零或大于 255 字节）

TCP/UDP 上游选项卡



注意

该选项卡需要在 `stream` 上下文的*upstream* 块中设置*zone* 指令。

此选项卡总结了从 API 部分*/status/stream/upstreams/* 生成的 `stream` 上下文的上游监控统计数据。

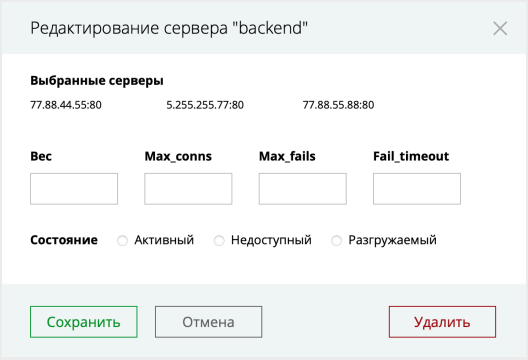
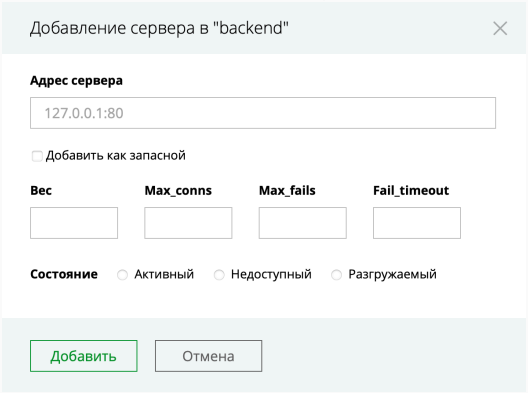
- 显示上游列表按钮切换上游摘要，显示有问题的上游和服务器数量。
- 仅显示失败开关切换为仅关注有问题上游的输出模式。
- 编辑按钮打开 上游编辑小部件。
- 每个上游表右侧的下拉列表允许按状态过滤其服务器（正常，失败，检查中，下线）。

对于每个上游提供以下数据：

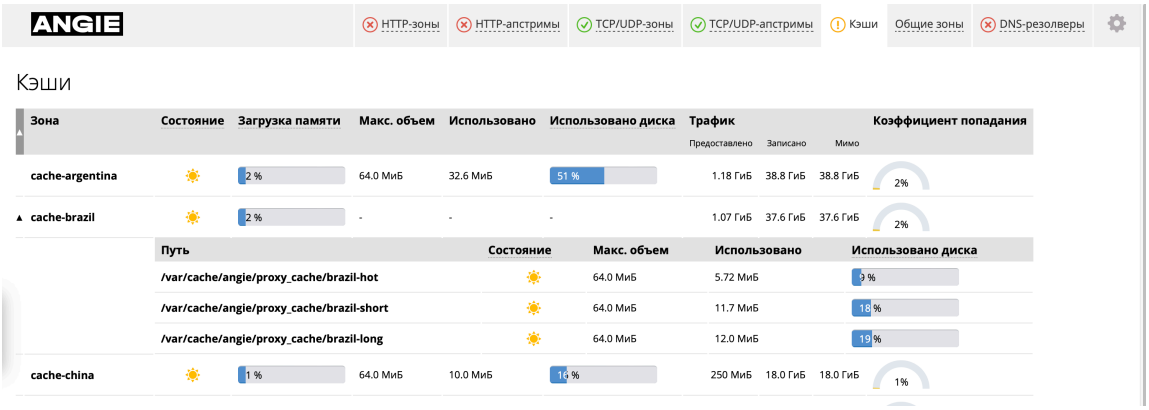
服务器	上游服务器的名称、停机时间和权重 **提示** 单击 服务器旁边的箭头，可以按状态或配置顺序对服务器进行排序。
连接	活动连接数及其最大限制（如设置）
流量	出入流量率，以及出入流量总量
服务器检查	达到服务器的不成功尝试次数以及服务器被视为不可用的次数（API 中的 `health` 对象）

编辑上游

在 Angie PRO 中，每个上游旁边有一个编辑按钮；单击后会出现两个更多按钮：

编辑选中的	编辑上游中选中的服务器。允许同时设置服务器的以下参数：权重，最大连接限制（`Max_conns`），将服务器视为不可用的最大失败次数（`Max_fails`），*max_fails* 的失败累积窗口（`Fail_timeout`），状态（活动 - 启用，下线 - 禁用或 正在排空 - 仅接收之前通过 `sticky` 绑定的会话请求）。 您还可以删除选中的服务器。 
添加服务器	向上游添加服务器。允许您设置以下参数：地址，是否备份，权重，最大连接限制（`Max_conns`），将服务器视为不可用的最大失败次数（`Max_fails`），失败累积窗口（`Fail_timeout`），状态（活动 - 启用，下线 - 禁用或 正在排空 - 仅接收之前通过 `sticky` 绑定的会话请求）。 

缓存选项卡



注意

该选项卡需要在 `http` 上下文中设置*proxy_cache_path* 指令。

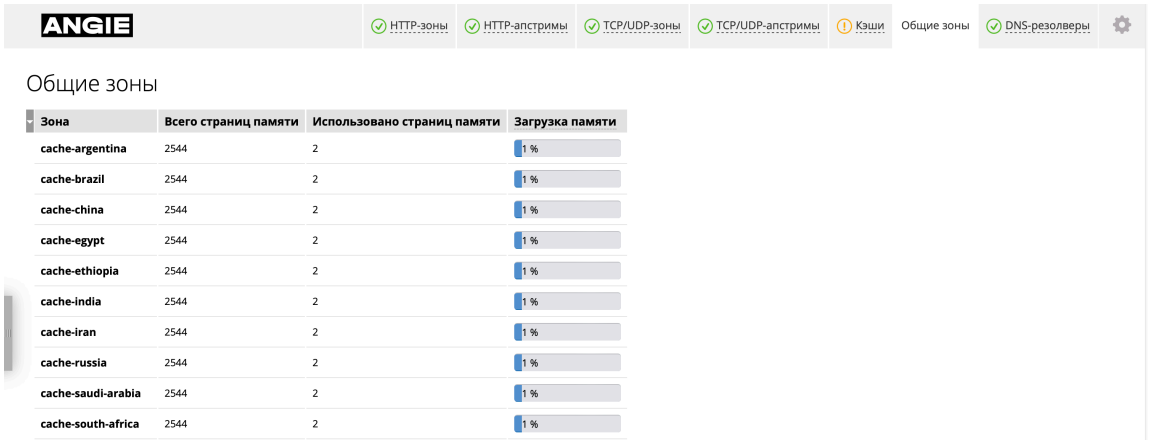
总结了从 API 部分*/status/http/caches/* 生成的 `http` 上下文中 `proxy_cache` 区域的监控统计数据。为每个区域提供以下数据：

区域	区域名称 **提示** 单击 区域旁边的图标，以打开或关闭所有具有分片 的区域列表。
状态	缓存状态：冷（元数据加载中）或热（元数据已加载）
内存使用	内存利用率
最大大小	最大内存大小
已使用	已使用的内存大小
磁盘使用	磁盘利用率
流量	从缓存传输的流量、写入到缓存的流量以及绕过缓存返回的流量
命中率	缓存命中率（从缓存传输的流量占总量的比例）

如果为某个区域启用了分片，则它被标记为下拉列表，其中包含各个分片：

路径	分片磁盘路径
状态	分片状态：冷（元数据加载中）或热（元数据已加载）
最大大小	最大内存大小
已使用	已使用的内存大小
磁盘使用	磁盘利用率

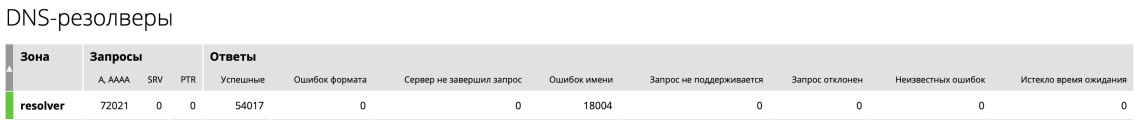
共享区域选项卡



此选项卡总结了所有上下文中 ** 所有 ** 共享内存区域的监控统计数据。为每个区域提供以下数据：

区域	区域名称 **提示** 单击 区域旁边的箭头，可以按大小或配置顺序对区域进行排序。
总内存页	总内存页数
已使用内存页	已使用的内存页数
内存使用	区域的内存利用率

DNS 解析器选项卡



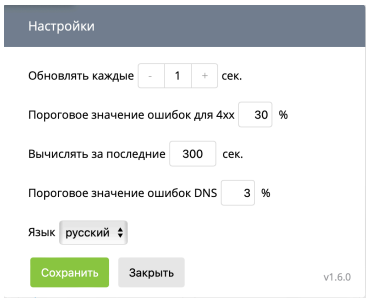
注意

该选项卡需要在 `http` 上下文中设置*resolver* 指令。

总结了从 API 部分*/status/resolvers/* 生成的 DNS 共享内存区域中的查询统计数据。为每个区域提供以下数据：

区域	区域名称 **提示** 单击 区域旁边的箭头，可以按状态或配置顺序对区域进行排序。
请求	A 和 AAAA、SRV、PTR 请求数
响应	按相关代码划分的响应数（成功、格式错误、服务器故障、名称错误、未实现、拒绝等）

设置小部件



允许您配置控制台的常规设置：

- 数据更新频率。默认值为 1 秒。
- 使黄色警告出现在相应服务器响应部分的 4xx 状态阈值比率。默认值为 7%。
- 用于计算缓存命中率的时间窗口。默认值为 300 秒。
- 使解析器变红的错误计数阈值。默认值为 3%。
- 控制台界面语言。可用选项：英语（English）和俄语（ ）。默认情况下，控制台语言基于浏览器中设置的区域设置选择。

控制台控制面板

所有选项卡在页面左中部都有一个拉出的侧边栏，其中有两个按钮 。顶部按钮暂停或重新启动来自 API 的数据更新，底部按钮在更新暂停时允许手动更新数据。

3.3.5 配置 Prometheus 仪表板

要在 Grafana 中配置 Angie 的 Prometheus 仪表板，请按照以下步骤操作：

1. 使用*Prometheus* 模块，在配置文件 的 `http` 块中添加以下*include* 指令：

```
http {
    include prometheus_all.conf;
```

```
    # ...
}
```

以及在一个使用指定 IP 地址和端口组合的单独 `server` 块中的 `location` 内添加相应的*prometheus*指令，例如：

```
server {

    listen 192.168.1.100:80;

    location = /p8s {
        prometheus all;
    }

    # ...

}
```

它们在 `location` 指定的端点以 Prometheus 格式导出 Angie 指标。

2. 将以下配置添加到 Prometheus 中，指定在前面步骤中设置的 `server` 的 IP 地址和端口：

```
scrape_configs:
  - job_name: "angie"
    scrape_interval: 15s
    metrics_path: "/p8s"
    static_configs:
      - targets: ["192.168.1.100:80"]
```

它每 15 秒收集一次指标，使用在上一步中配置的 `/p8s` 路径。

> **备注**
>
> 确保全局 `scrape_interval` 的值不大于此处的值。

3. 将 Angie 的 Prometheus 仪表板 导入到 Grafana 中。

章节 4

故障排除

如果您遇到技术问题，并且在其他部分找不到解决方案，请在 社区论坛 或 Telegram 频道 上提问。

客户技术支持：

- https://support.angie.software
- support@angie.software

4.1 调试日志

在执行自我诊断或按照支持建议进行操作之前，应启用调试日志。

为此，请使用启用调试日志的可执行文件运行 Angie：

Linux

在 Linux 的 预构建包中，`angie-debug` 文件是启用调试日志构建的：

```
$ ls -l /usr/sbin/ | grep angie

   lrwxrwxrwx 1 root root      13 Sep 21 18:58 angie -> angie-nodebug
   -rwxr-xr-x 1 root root 1561224 Sep 21 18:58 angie-debug
   -rwxr-xr-x 1 root root 1426056 Sep 21 18:58 angie-nodebug
```

配置运行 `angie-debug`：

```
$ sudo ln -fs angie-debug /usr/sbin/angie
$ sudo angie -t && sudo service angie upgrade
```

这将启动实时可执行文件升级。

调试后要恢复到常规可执行文件：

```
$ sudo ln -fs angie-nodebug /usr/sbin/angie
$ sudo angie -t && sudo service angie upgrade
```

FreeBSD

在 FreeBSD 的 预构建包中，`angie-debug` 文件是启用调试日志构建的：

```
$ ls -l /usr/local/sbin/ | grep angie

  lrwxrwxrwx 1 root root      13 Sep 21 18:58 angie -> angie-nodebug
  -rwxr-xr-x 1 root root 1561224 Sep 21 18:58 angie-debug
  -rwxr-xr-x 1 root root 1426056 Sep 21 18:58 angie-nodebug
```

配置运行 `angie-debug`：

```
$ sudo ln -fs angie-debug /usr/local/sbin/angie
$ sudo angie -t && sudo service angie upgrade
```

这将启动实时可执行文件升级。

调试后要恢复到常规可执行文件：

```
$ sudo ln -fs angie-nodebug /usr/local/sbin/angie
$ sudo angie -t && sudo service angie upgrade
```

从源构建

当从源构建 Angie 时，编译前 启用调试：

```
$ ./configure --with-debug ...
```

安装后，使用 **`angie -V`** 可以验证是否启用了调试日志：

```
$ angie -V

  ...
  configure arguments: --with-debug ...
```

备注

使用启用调试支持的可执行文件可能会略微降低性能；启用调试日志可能会显著降低性能并增加磁盘空间使用。

要启用调试日志，请在配置中使用*error_log* 指令设置 `debug` 级别：

```
error_log /path/to/log debug;
```

并重新加载配置：

```
$ sudo angie -t && sudo service angie reload
```

> **备注**
>
> 如果切换到未启用调试支持的可执行文件，但在*error_log* 指令中保留 debug 级别，Angie 将以 info 级别记录条目。

在配置中覆盖*error_log* 而不指定 debug 级别将禁用调试日志。在此处，在*server* 级别覆盖日志会为单个服务器禁用调试日志：

```
error_log /path/to/log debug;

http {
   server {
     error_log /path/to/log;
    # ...
```

要避免这种情况，请移除覆盖*error_log* 的行，或在其中设置 debug 级别：

```
error_log /path/to/log debug;

http {
   server {
     error_log /path/to/log debug;
   #  ...
```

4.1.1 指令位置

error_log 指令的位置会影响收集的调试信息的完整性。

在较低配置级别（例如，在 server 或 location 块内）指定的指令会覆盖在较高级别（例如，在主配置级别或 http 块内）指定的日志设置。

为特定服务器禁用调试日志

如果全局启用了调试日志，但为单个服务器指定了不带 debug 级别的*error_log*，则不会为该服务器收集调试信息。

```
error_log /var/log/angie/error.log debug; # 全局调试日志

http {

    server {
```

```
        listen 80;
        server_name example.com;

        error_log /var/log/angie/example.com.error.log;
        # example.com 的调试日志被禁用，文件包含 info 级别

        # ...
    }

    server {

        listen 80;
        server_name another.com;

        # 此服务器将使用全局调试日志
        # ...
    }
}
```

在服务器级别保留调试日志

要为特定服务器保留调试信息收集但将其定向到不同的文件，您必须同时指定 debug 级别：

```
error_log /var/log/angie/error.log debug; # 全局调试日志

http {

    server {

        listen 80;
        server_name example.com;

        error_log /var/log/angie/example.com.error.log debug;
        # example.com 的调试日志已启用但写入单独的文件

        # ...
    }
}
```

因此，要全局启用调试日志但为单个块覆盖日志文件，也请在这些覆盖中指定 debug 级别。否则，如果在*error_log* 指令中未指定日志级别，将默认使用 error 级别，这些块的调试信息将丢失。

4.1.2 记录特定地址

您可以仅为指定的客户端地址 启用调试日志：

```
error_log /path/to/log;

events {
  debug_connection 192.168.1.1;
  debug_connection 192.168.10.0/24;
}
```

4.1.3 循环内存缓冲区

调试日志可以写入循环内存缓冲区：

```
error_log memory:32m debug;
```

在 debug 级别写入内存缓冲区即使在高负载下也不会显著影响性能。在这种情况下，可以使用 GDB 脚本提取日志，例如：

```
set $log = ngx_cycle->log

while $log->writer != ngx_log_memory_writer
  set $log = $log->next
end

set $buf = (ngx_log_memory_buf_t *) $log->wdata
dump binary memory debug_log.txt $buf->start $buf->end
```

4.2 核心转储

核心转储有助于调查崩溃。在联系支持 时请包含它们。对于从 我们的仓库构建的版本，我们在特殊包中提供调试符号。它们与原始包同名，但添加了 -dbg 后缀，例如 angie-dbg。

> **备注**
>
> 本节假设您以 root 用户身份运行 Angie（推荐）。

4.2.1 Linux: systemd

要在将 Angie 作为 **systemd** 服务运行时启用核心转储保存(例如,当 从包安装时),请修改 /lib/systemd/system/angie.service 文件中的 服务设置：

```
[Service]
...
LimitCORE=infinity
LimitNOFILE=65535
```

或更新 /etc/systemd/system.conf 文件中的 全局设置：

```
[Manager]
...
DefaultLimitCORE=infinity
DefaultLimitNOFILE=65535
```

然后重新加载服务配置并重启 Angie 以重现崩溃条件：

```
$ sudo systemctl daemon-reload
$ sudo systemctl restart angie.service
```

崩溃后，查找核心转储文件：

```
$ sudo coredumpctl -1 # 可选

   TIME                            PID   UID   GID SIG COREFILE  EXE
   --- 2025-07-03 11:05:40 GMT    1157     0     0  11 present   /usr/sbin/angie

$ sudo ls -al /var/lib/systemd/coredump/  # 默认位置，另见 /etc/systemd/coredump.conf 和 /etc/
↪systemd/coredump.conf.d/*.conf

  ...
  -rw-r----- 1 root root 177662 Jul 27 11:05 core.angie.0.6135489c850b4fb4a74795ebbc1e382a.
↪1157.1590577472000000.lz4
```

4.2.2 Linux: 手动配置

检查 /etc/security/limits.conf 文件中的 核心转储设置，必要时修改它们：

```
root soft core 0          # 默认禁用核心转储
root hard core unlimited  # 允许增加大小限制
```

然后使用 ulimit 增加核心转储大小限制，然后重启 Angie 以重现崩溃条件：

```
$ sudo ulimit -c unlimited
$ sudo cd <Angie 安装目录路径>
$ sudo sbin/angie  # 或 sbin/angie-debug
```

崩溃后，查找核心转储文件：

```
$ sudo ls -al <Angie 工作目录路径>  # 默认位置，见 /proc/sys/kernel/core_pattern
  ...
  -rw-r----- 1 root root 177662 Jul 27 11:05 core.1157
```

4.2.3 FreeBSD

检查 /etc/sysctl.conf 文件中的 核心转储设置，必要时修改它们：

```
kern.coredump=1                              # 应为 1
kern.corefile=/path/to/core/files/%N.core    # 需要正确路径
```

或在运行时更新设置：

```
$ sudo sysctl kern.coredump=1
$ sudo sysctl kern.corefile=/path/to/core/files/%N.core
```

然后重启 Angie 以重现崩溃条件。如果 Angie 作为服务安装：

```
$ sudo service angie restart
```

如果 Angie 手动安装：

```
$ sudo cd <Angie 安装目录路径>
$ sudo sbin/angie
```

崩溃后，查找核心转储文件：

```
$ sudo ls -al <核心转储文件路径>

  ...
  -rw------- 1 root root 9912320 Jul 27 11:05 angie.core
```

章节 5

知识产权

软件产品 Angie PRO 的文档是 Web Server, LLC 的知识产权。该文档是在对软件产品 nginx 文档进行修改（修订）后创建的。

索引

A

absolute_redirect (*http*), 331
accept_mutex (*core*), 21
accept_mutex_delay (*core*), 21
access_log (*http*), 146
access_log (*stream*), 389
acme (*http*), 32
acme (*stream*), 374
acme_client (*http*), 33
acme_client_path (*http*), 34
acme_dns_port (*http*), 35
acme_hook (*http*), 35
add_after_body (*http*), 37
add_before_body (*http*), 37
add_header (*http*), 127
add_trailer (*http*), 127
addition_types (*http*), 37
aio (*http*), 331
aio_write (*http*), 332
alias (*http*), 332
allow (*http*), 31
allow (*stream*), 373
ancient_browser (*http*), 79
ancient_browser_value (*http*), 79
api (*http*), 38
api_config_files (*http*), 39
auth_basic (*http*), 73
auth_basic_user_file (*http*), 73
auth_delay (*http*), 333
auth_http, 456
auth_http_header, 456
auth_http_pass_client_cert, 456
auth_http_timeout, 456
auth_request (*http*), 75
auth_request_set (*http*), 75
auto_redirect (*http*), 333
autoindex (*http*), 75
autoindex_exact_size (*http*), 75
autoindex_format (*http*), 76
autoindex_localtime (*http*), 78

B

backup_switch (*http*), 275
backup_switch (*stream*), 430
bind_conn (*http*), 275
break (*http*), 224

C

charset (*http*), 81
charset_map (*http*), 81
charset_types (*http*), 82
chunked_transfer_encoding (*http*), 334
client (*http*), 334
client_body_buffer_size (*http*), 334
client_body_in_file_only (*http*), 335
client_body_in_single_buffer (*http*), 335
client_body_temp_path (*http*), 335
client_body_timeout (*http*), 335
client_header_buffer_size (*http*), 336
client_header_timeout (*http*), 336
client_max_body_size (*http*), 336
connection_pool_size (*http*), 336
create_full_put_path (*http*), 83

D

daemon (*core*), 21
dav_access (*http*), 83

dav_methods (*http*), 84
debug_connection (*core*), 21
debug_points (*core*), 22
default_type (*http*), 337
deny (*http*), 31
deny (*stream*), 373
directio (*http*), 337
directio_alignment (*http*), 337
disable_symlinks (*http*), 337
docker_endpoint (*http*), 88
docker_max_object_size (*http*), 88

E

empty_gif (*http*), 89
env (*core*), 22
error_log (*core*), 23
error_page (*http*), 338
etag (*http*), 339
events (*core*), 24
expires (*http*), 128

F

fastcgi_bind (*http*), 90
fastcgi_buffer_size (*http*), 90
fastcgi_buffering (*http*), 90
fastcgi_buffers (*http*), 91
fastcgi_busy_buffers_size (*http*), 91
fastcgi_cache (*http*), 91
fastcgi_cache_background_update (*http*), 91
fastcgi_cache_bypass (*http*), 92
fastcgi_cache_key (*http*), 92
fastcgi_cache_lock (*http*), 92
fastcgi_cache_lock_age (*http*), 92
fastcgi_cache_lock_timeout (*http*), 93
fastcgi_cache_max_range_offset (*http*), 93
fastcgi_cache_methods (*http*), 93
fastcgi_cache_min_uses (*http*), 93
fastcgi_cache_path (*http*), 94
fastcgi_cache_revalidate (*http*), 95
fastcgi_cache_use_stale (*http*), 95
fastcgi_cache_valid (*http*), 96
fastcgi_catch_stderr (*http*), 97
fastcgi_connect_timeout (*http*), 97
fastcgi_connection_drop (*http*), 97
fastcgi_force_ranges (*http*), 97
fastcgi_hide_header (*http*), 98
fastcgi_ignore_client_abort (*http*), 98
fastcgi_ignore_headers (*http*), 98
fastcgi_index (*http*), 99
fastcgi_intercept_errors (*http*), 99
fastcgi_keep_conn (*http*), 99
fastcgi_limit_rate (*http*), 99
fastcgi_max_temp_file_size (*http*), 100
fastcgi_next_upstream (*http*), 100
fastcgi_next_upstream_timeout (*http*), 101
fastcgi_next_upstream_tries (*http*), 101
fastcgi_no_cache (*http*), 102
fastcgi_param (*http*), 102
fastcgi_pass (*http*), 103
fastcgi_pass_header (*http*), 103
fastcgi_pass_request_body (*http*), 103
fastcgi_pass_request_headers (*http*), 104
fastcgi_read_timeout (*http*), 104
fastcgi_request_buffering (*http*), 104
fastcgi_send_lowat (*http*), 104
fastcgi_send_timeout (*http*), 105
fastcgi_socket_keepalive (*http*), 105
fastcgi_split_path_info (*http*), 105
fastcgi_store (*http*), 106
fastcgi_store_access (*http*), 107
fastcgi_temp_file_write_size (*http*), 107
fastcgi_temp_path (*http*), 107
feedback (*http*), 276
feedback (*stream*), 430
flv (*http*), 109

G

geo (*http*), 109
geo (*stream*), 376
geoip_city (*http*), 111
geoip_city (*stream*), 378
geoip_country (*http*), 111
geoip_country (*stream*), 378
geoip_org (*http*), 112
geoip_org (*stream*), 378
geoip_proxy (*http*), 112
geoip_proxy_recursive (*http*), 112
google_perftools_profiles, 478
grpc_bind (*http*), 113
grpc_buffer_size (*http*), 114

grpc_connect_timeout (*http*), 114
grpc_connection_drop (*http*), 114
grpc_hide_header (*http*), 114
grpc_ignore_headers (*http*), 115
grpc_intercept_errors (*http*), 115
grpc_next_upstream (*http*), 115
grpc_next_upstream_timeout (*http*), 116
grpc_next_upstream_tries (*http*), 117
grpc_pass (*http*), 117
grpc_pass_header (*http*), 117
grpc_read_timeout (*http*), 118
grpc_send_timeout (*http*), 118
grpc_set_header (*http*), 118
grpc_socket_keepalive (*http*), 118
grpc_ssl_certificate (*http*), 119
grpc_ssl_certificate_cache (*http*), 119
grpc_ssl_certificate_key (*http*), 119
grpc_ssl_ciphers (*http*), 120
grpc_ssl_conf_command (*http*), 120
grpc_ssl_crl (*http*), 121
grpc_ssl_name (*http*), 121
grpc_ssl_password_file (*http*), 121
grpc_ssl_protocols (*http*), 121
grpc_ssl_server_name (*http*), 122
grpc_ssl_session_reuse (*http*), 122
grpc_ssl_trusted_certificate (*http*), 122
grpc_ssl_verify (*http*), 122
grpc_ssl_verify_depth (*http*), 122
gunzip (*http*), 123
gunzip_buffers (*http*), 123
gzip (*http*), 124
gzip_buffers (*http*), 124
gzip_comp_level (*http*), 124
gzip_disable (*http*), 124
gzip_http_version (*http*), 125
gzip_min_length (*http*), 125
gzip_proxied (*http*), 125
gzip_static (*http*), 126
gzip_types (*http*), 125
gzip_vary (*http*), 126

H

hash (*http*), 277
hash (*stream*), 432
http (*http*), 340
http2 (*http*), 322
http2_body_preread_size (*http*), 322
http2_chunk_size (*http*), 322
http2_max_concurrent_pushes (*http*), 323
http2_max_concurrent_streams (*http*), 323
http2_push (*http*), 323
http2_push_preload (*http*), 323
http2_recv_buffer_size (*http*), 324
http3 (*http*), 325
http3_hq (*http*), 325
http3_max_concurrent_streams (*http*), 326
http3_max_table_capacity (*http*), 326
http3_stream_buffer_size (*http*), 326

I

if (*http*), 225
if_modified_since (*http*), 340
ignore_invalid_headers (*http*), 340
image_filter (*http*), 129
image_filter_buffer (*http*), 130
image_filter_interlace (*http*), 130
image_filter_jpeg_quality (*http*), 130
image_filter_sharpen (*http*), 130
image_filter_transparency (*http*), 131
image_filter_webp_quality (*http*), 131
imap_auth, 459
imap_capabilities, 459
imap_client_buffer, 460
include (*core*), 24
index (*http*), 131
internal (*http*), 340
ip_hash (*http*), 278

J

js_access (*stream*), 381
js_body_filter (*http*), 135
js_content (*http*), 135
js_fetch_buffer_size (*http*), 135
js_fetch_buffer_size (*stream*), 381
js_fetch_ciphers (*http*), 136
js_fetch_ciphers (*stream*), 381
js_fetch_max_response_buffer_size (*http*), 136
js_fetch_max_response_buffer_size (*stream*), 382
js_fetch_protocols (*http*), 136

js_fetch_protocols (*stream*), 382
js_fetch_timeout (*http*), 136
js_fetch_timeout (*stream*), 382
js_fetch_trusted_certificate (*http*), 136
js_fetch_trusted_certificate (*stream*), 382
js_fetch_verify (*http*), 137
js_fetch_verify (*stream*), 382
js_fetch_verify_depth (*http*), 137
js_fetch_verify_depth (*stream*), 383
js_filter (*stream*), 383
js_header_filter (*http*), 137
js_import (*http*), 137
js_import (*stream*), 383
js_path (*http*), 138
js_path (*stream*), 384
js_preload_object (*http*), 138
js_preload_object (*stream*), 384
js_preread (*stream*), 384
js_set (*http*), 138
js_set (*stream*), 385
js_shared_dict_zone (*http*), 139
js_shared_dict_zone (*stream*), 385
js_var (*http*), 140
js_var (*stream*), 386

K

keepalive (*http*), 278
keepalive_disable (*http*), 341
keepalive_requests (*http*), 280, 341
keepalive_time (*http*), 280, 341
keepalive_timeout (*http*), 281, 342

L

large_client_header_buffers (*http*), 342
least_conn (*http*), 281
least_conn (*stream*), 432
least_time (*http*), 281
least_time (*stream*), 432
limit_conn (*http*), 141
limit_conn (*stream*), 387
limit_conn_dry_run (*http*), 141
limit_conn_dry_run (*stream*), 387
limit_conn_log_level (*http*), 142
limit_conn_log_level (*stream*), 388
limit_conn_status (*http*), 142
limit_conn_zone (*http*), 142
limit_conn_zone (*stream*), 388
limit_except (*http*), 342
limit_rate (*http*), 343
limit_rate_after (*http*), 343
limit_req (*http*), 143
limit_req_dry_run (*http*), 144
limit_req_log_level (*http*), 144
limit_req_status (*http*), 144
limit_req_zone (*http*), 144
lingering_close (*http*), 344
lingering_time (*http*), 344
lingering_timeout (*http*), 344
listen, 473
listen (*http*), 345
listen (*stream*), 444
load, 482
load_module (*core*), 24
location (*http*), 348
lock_file (*core*), 24
log_format (*http*), 147
log_format (*stream*), 390
log_not_found (*http*), 350
log_subrequest (*http*), 350

M

mail, 475
map (*http*), 149
map (*stream*), 391
map_hash_bucket_size (*http*), 150
map_hash_bucket_size (*stream*), 392
map_hash_max_size (*http*), 150
map_hash_max_size (*stream*), 393
master_process (*core*), 25
max_commands, 475
max_errors, 475
max_headers (*http*), 351
max_ranges (*http*), 351
memcached_bind (*http*), 151
memcached_buffer_size (*http*), 151
memcached_connect_timeout (*http*), 152
memcached_gzip_flag (*http*), 152
memcached_next_upstream (*http*), 152
memcached_next_upstream_timeout (*http*), 153
memcached_next_upstream_tries (*http*), 153

memcached_pass (*http*), 153
memcached_read_timeout (*http*), 154
memcached_send_timeout (*http*), 154
memcached_socket_keepalive (*http*), 154
merge_slashes (*http*), 351
min_delete_depth (*http*), 84
mirror (*http*), 155
mirror_request_body (*http*), 155
modern_browser (*http*), 79
modern_browser_value (*http*), 80
mp4 (*http*), 157
mp4_buffer_size (*http*), 157
mp4_limit_rate (*http*), 157
mp4_limit_rate_after (*http*), 158
mp4_max_buffer_size (*http*), 157
mp4_start_key_frame (*http*), 158
mqtt_preread (*stream*), 393
msie_padding (*http*), 351
msie_refresh (*http*), 352
multi_accept (*core*), 25

O

open_file_cache (*http*), 352
open_file_cache_errors (*http*), 352
open_file_cache_min_uses (*http*), 353
open_file_cache_valid (*http*), 353
open_log_file_cache (*http*), 147
open_log_file_cache (*stream*), 390
output_buffers (*http*), 353
override_charset (*http*), 82

P

pass (*stream*), 395
pcre_jit (*core*), 25
perl (*http*), 160
perl_modules (*http*), 160
perl_require (*http*), 160
perl_set (*http*), 161
pid (*core*), 26
pop3_auth, 460
pop3_capabilities, 460
port_in_redirect (*http*), 353
postpone_output (*http*), 353
preread_buffer_size (*stream*), 447
preread_timeout (*stream*), 447
prometheus (*http*), 185
prometheus_template (*http*), 185
protocol, 475
proxy_bind (*http*), 188
proxy_bind (*stream*), 396
proxy_buffer, 461
proxy_buffer_size (*http*), 188
proxy_buffer_size (*stream*), 396
proxy_buffering (*http*), 189
proxy_buffers (*http*), 189
proxy_busy_buffers_size (*http*), 189
proxy_cache (*http*), 190
proxy_cache_background_update (*http*), 191
proxy_cache_bypass (*http*), 191
proxy_cache_convert_head (*http*), 191
proxy_cache_key (*http*), 191
proxy_cache_lock (*http*), 192
proxy_cache_lock_age (*http*), 192
proxy_cache_lock_timeout (*http*), 192
proxy_cache_max_range_offset (*http*), 192
proxy_cache_methods (*http*), 193
proxy_cache_min_uses (*http*), 193
proxy_cache_path (*http*), 193
proxy_cache_revalidate (*http*), 195
proxy_cache_use_stale (*http*), 195
proxy_cache_valid (*http*), 196
proxy_connect_timeout (*http*), 197
proxy_connect_timeout (*stream*), 396
proxy_connection_drop (*http*), 197
proxy_connection_drop (*stream*), 396
proxy_cookie_domain (*http*), 197
proxy_cookie_flags (*http*), 198
proxy_cookie_path (*http*), 199
proxy_download_rate (*stream*), 397
proxy_force_ranges (*http*), 199
proxy_half_close (*stream*), 397
proxy_headers_hash_bucket_size (*http*), 199
proxy_headers_hash_max_size (*http*), 200
proxy_hide_header (*http*), 200
proxy_http3_hq (*http*), 200
proxy_http3_max_concurrent_streams (*http*), 201
proxy_http3_max_table_capacity (*http*), 201
proxy_http3_stream_buffer_size (*http*), 201
proxy_http_version (*http*), 200

proxy_ignore_client_abort (*http*), 201
proxy_ignore_headers (*http*), 202
proxy_intercept_errors (*http*), 202
proxy_limit_rate (*http*), 202
proxy_max_temp_file_size (*http*), 203
proxy_method (*http*), 203
proxy_next_upstream (*http*), 203
proxy_next_upstream (*stream*), 398
proxy_next_upstream_timeout (*http*), 205
proxy_next_upstream_timeout (*stream*), 398
proxy_next_upstream_tries (*http*), 205
proxy_next_upstream_tries (*stream*), 398
proxy_no_cache (*http*), 205
proxy_pass (*http*), 205
proxy_pass (*stream*), 398
proxy_pass_error_message, 461
proxy_pass_header (*http*), 207
proxy_pass_request_body (*http*), 207
proxy_pass_request_headers (*http*), 207
proxy_pass_trailers (*http*), 208
proxy_protocol, 461
proxy_protocol (*stream*), 399
proxy_protocol_timeout (*stream*), 447
proxy_quic_active_connection_id_limit (*http*), 208
proxy_quic_gso (*http*), 208
proxy_quic_host_key (*http*), 209
proxy_read_timeout (*http*), 209
proxy_redirect (*http*), 209
proxy_request_buffering (*http*), 210
proxy_requests (*stream*), 399
proxy_responses (*stream*), 399
proxy_send_lowat (*http*), 211
proxy_send_timeout (*http*), 211
proxy_set_body (*http*), 211
proxy_set_header (*http*), 212
proxy_smtp_auth, 461
proxy_socket_keepalive (*http*), 212
proxy_socket_keepalive (*stream*), 399
proxy_ssl (*stream*), 400
proxy_ssl_certificate (*http*), 213
proxy_ssl_certificate (*stream*), 400
proxy_ssl_certificate_cache (*http*), 213
proxy_ssl_certificate_key (*http*), 214
proxy_ssl_certificate_key (*stream*), 400
proxy_ssl_ciphers (*http*), 214
proxy_ssl_ciphers (*stream*), 401
proxy_ssl_conf_command (*http*), 215
proxy_ssl_conf_command (*stream*), 401
proxy_ssl_crl (*http*), 215
proxy_ssl_crl (*stream*), 402
proxy_ssl_name (*http*), 215
proxy_ssl_name (*stream*), 402
proxy_ssl_ntls (*http*), 216
proxy_ssl_ntls (*stream*), 402
proxy_ssl_password_file (*http*), 216
proxy_ssl_password_file (*stream*), 403
proxy_ssl_protocols (*http*), 217
proxy_ssl_protocols (*stream*), 403
proxy_ssl_server_name (*http*), 217
proxy_ssl_server_name (*stream*), 404
proxy_ssl_session_reuse (*http*), 217
proxy_ssl_session_reuse (*stream*), 404
proxy_ssl_trusted_certificate (*http*), 217
proxy_ssl_trusted_certificate (*stream*), 404
proxy_ssl_verify (*http*), 218
proxy_ssl_verify (*stream*), 404
proxy_ssl_verify_depth (*http*), 218
proxy_ssl_verify_depth (*stream*), 404
proxy_store (*http*), 218
proxy_store_access (*http*), 219
proxy_temp_file_write_size (*http*), 219
proxy_temp_path (*http*), 220
proxy_timeout, 462
proxy_timeout (*stream*), 405
proxy_upload_rate (*stream*), 405

Q

queue (*http*), 282
quic_active_connection_id_limit (*http*), 326
quic_bpf (*http*), 327
quic_gso (*http*), 327
quic_host_key (*http*), 327
quic_retry (*http*), 328

R

random (*http*), 282
random (*stream*), 433
random_index (*http*), 221
rdp_preread (*stream*), 406

read_ahead (*http*), 354
real_ip_header (*http*), 222
real_ip_recursive (*http*), 222
recursive_error_pages (*http*), 354
referer_hash_bucket_size (*http*), 223
referer_hash_max_size (*http*), 223
request_pool_size (*http*), 354
reset_timedout_connection (*http*), 354
resolver, 476
resolver (*http*), 355
resolver (*stream*), 447
resolver_timeout, 477
resolver_timeout (*http*), 356
resolver_timeout (*stream*), 448
response_time_factor (*http*), 283
response_time_factor (*stream*), 433
return (*http*), 226
return (*stream*), 408
rewrite (*http*), 226
rewrite_log (*http*), 227
root (*http*), 356

S

satisfy (*http*), 356
scgi_bind (*http*), 229
scgi_buffer_size (*http*), 230
scgi_buffering (*http*), 230
scgi_buffers (*http*), 230
scgi_busy_buffers_size (*http*), 231
scgi_cache (*http*), 231
scgi_cache_background_update (*http*), 231
scgi_cache_bypass (*http*), 232
scgi_cache_key (*http*), 232
scgi_cache_lock (*http*), 232
scgi_cache_lock_age (*http*), 232
scgi_cache_lock_timeout (*http*), 233
scgi_cache_max_range_offset (*http*), 233
scgi_cache_methods (*http*), 233
scgi_cache_min_uses (*http*), 233
scgi_cache_path (*http*), 234
scgi_cache_revalidate (*http*), 235
scgi_cache_use_stale (*http*), 235
scgi_cache_valid (*http*), 236
scgi_connect_timeout (*http*), 237
scgi_connection_drop (*http*), 237
scgi_force_ranges (*http*), 237
scgi_hide_header (*http*), 237
scgi_ignore_client_abort (*http*), 238
scgi_ignore_headers (*http*), 238
scgi_limit_rate (*http*), 238
scgi_max_temp_file_size (*http*), 239
scgi_next_upstream (*http*), 239
scgi_next_upstream_timeout (*http*), 240
scgi_next_upstream_tries (*http*), 240
scgi_no_cache (*http*), 241
scgi_param (*http*), 241
scgi_pass (*http*), 242
scgi_pass_header (*http*), 242
scgi_pass_request_body (*http*), 242
scgi_pass_request_headers (*http*), 242
scgi_read_timeout (*http*), 243
scgi_request_buffering (*http*), 243
scgi_send_timeout (*http*), 243
scgi_socket_keepalive (*http*), 243
scgi_store (*http*), 244
scgi_store_access (*http*), 244
scgi_temp_file_write_size (*http*), 245
scgi_temp_path (*http*), 245
secure_link (*http*), 246
secure_link_md5 (*http*), 246
secure_link_secret (*http*), 247
send_lowat (*http*), 357
send_timeout (*http*), 357
sendfile (*http*), 357
sendfile_max_chunk (*http*), 358
server, 477
server (*http*), 283, 358
server (*stream*), 427, 448
server_name, 477
server_name (*http*), 358
server_name (*stream*), 449
server_name_in_redirect (*http*), 360
server_names_hash_bucket_size (*http*), 360
server_names_hash_bucket_size (*stream*), 450
server_names_hash_max_size (*http*), 361
server_names_hash_max_size (*stream*), 450
server_tokens (*http*), 361
set (*http*), 227
set (*stream*), 408
set_real_ip_from, 463

set_real_ip_from (*http*), 221
set_real_ip_from (*stream*), 407
slice (*http*), 249
smtp_auth, 463
smtp_capabilities, 464
smtp_client_buffer, 464
smtp_greeting_delay, 464
source_charset (*http*), 82
split_clients (*http*), 250
split_clients (*stream*), 409
ssi (*http*), 250
ssi_last_modified (*http*), 251
ssi_min_file_chunk (*http*), 251
ssi_silent_errors (*http*), 251
ssi_types (*http*), 251
ssi_value_length (*http*), 251
ssl_alpn (*stream*), 411
ssl_buffer_size (*http*), 256
ssl_certificate, 465
ssl_certificate (*http*), 257
ssl_certificate (*stream*), 411
ssl_certificate_cache (*http*), 258
ssl_certificate_key, 466
ssl_certificate_key (*http*), 258
ssl_certificate_key (*stream*), 412
ssl_ciphers, 466
ssl_ciphers (*http*), 259
ssl_ciphers (*stream*), 413
ssl_client_certificate, 467
ssl_client_certificate (*http*), 260
ssl_client_certificate (*stream*), 413
ssl_conf_command, 467
ssl_conf_command (*http*), 260
ssl_conf_command (*stream*), 413
ssl_crl, 467
ssl_crl (*http*), 260
ssl_crl (*stream*), 414
ssl_dhparam, 468
ssl_dhparam (*http*), 261
ssl_dhparam (*stream*), 414
ssl_early_data (*http*), 261
ssl_early_data (*stream*), 415
ssl_ecdh_curve, 468
ssl_ecdh_curve (*http*), 261
ssl_ecdh_curve (*stream*), 415
ssl_engine (*core*), 26
ssl_handshake_timeout (*stream*), 415
ssl_ntls (*http*), 262
ssl_ntls (*stream*), 417
ssl_object_cache_inheritable (*core*), 26
ssl_ocsp (*http*), 262
ssl_ocsp (*stream*), 416
ssl_ocsp_cache (*http*), 263
ssl_ocsp_cache (*stream*), 416
ssl_ocsp_responder (*http*), 263, 416
ssl_password_file, 468
ssl_password_file (*http*), 263
ssl_password_file (*stream*), 417
ssl_prefer_server_ciphers, 469
ssl_prefer_server_ciphers (*http*), 264
ssl_prefer_server_ciphers (*stream*), 418
ssl_preread (*stream*), 425
ssl_protocols, 469
ssl_protocols (*http*), 264
ssl_protocols (*stream*), 418
ssl_reject_handshake (*http*), 264
ssl_session_cache, 470
ssl_session_cache (*http*), 265
ssl_session_cache (*stream*), 418
ssl_session_ticket_key, 470
ssl_session_ticket_key (*http*), 266
ssl_session_ticket_key (*stream*), 419
ssl_session_tickets, 471
ssl_session_tickets (*http*), 266
ssl_session_tickets (*stream*), 419
ssl_session_timeout, 471
ssl_session_timeout (*http*), 266
ssl_session_timeout (*stream*), 419
ssl_stapling (*http*), 266
ssl_stapling (*stream*), 420
ssl_stapling_file (*http*), 267
ssl_stapling_file (*stream*), 420
ssl_stapling_responder (*http*), 267
ssl_stapling_responder (*stream*), 420
ssl_stapling_verify (*http*), 267
ssl_stapling_verify (*stream*), 421
ssl_trusted_certificate, 471
ssl_trusted_certificate (*http*), 267
ssl_trusted_certificate (*stream*), 421
ssl_verify_client, 471

ssl_verify_client (*http*), 268
ssl_verify_client (*stream*), 421
ssl_verify_depth, 472
ssl_verify_depth (*http*), 268
ssl_verify_depth (*stream*), 421
starttls, 472
state (*http*), 286
state (stream) ((stream upstream module), 429
status_zone (*http*), 361
status_zone (*stream*), 450
sticky (*http*), 286
sticky (*stream*), 434
sticky_secret (*http*), 291
sticky_secret (*stream*), 438
sticky_strict (*http*), 291
sticky_strict (*stream*), 437
stream (*stream*), 451
stub_status (*http*), 272
sub_filter (*http*), 273
sub_filter_last_modified (*http*), 273
sub_filter_once (*http*), 274
sub_filter_types (*http*), 274
subrequest_output_buffer_size (*http*), 362

T

tcp_nodelay (*http*), 363
tcp_nodelay (*stream*), 452
tcp_nopush (*http*), 363
thread_pool (*core*), 27
timeout, 477
timer_resolution (*core*), 27
try_files (*http*), 363
types (*http*), 365
types_hash_bucket_size (*http*), 366
types_hash_max_size (*http*), 366

U

underscores_in_headers (*http*), 366
uninitialized_variable_warn (*http*), 228
upstream (*http*), 292
upstream (*stream*), 426
upstream_probe (*http*), 295
upstream_probe (*stream*), 440
upstream_probe_timeout (*stream*), 405
use (*core*), 27
user (*core*), 28
userid (*http*), 298
userid_domain (*http*), 298
userid_expires (*http*), 298
userid_flags (*http*), 299
userid_mark (*http*), 299
userid_name (*http*), 299
userid_p3p (*http*), 299
userid_path (*http*), 300
userid_service (*http*), 300
uwsgi_bind (*http*), 301
uwsgi_buffer_size (*http*), 301
uwsgi_buffering (*http*), 301
uwsgi_buffers (*http*), 302
uwsgi_busy_buffers_size (*http*), 302
uwsgi_cache (*http*), 302
uwsgi_cache_background_update (*http*), 303
uwsgi_cache_bypass (*http*), 303
uwsgi_cache_key (*http*), 303
uwsgi_cache_lock (*http*), 304
uwsgi_cache_lock_age (*http*), 304
uwsgi_cache_lock_timeout (*http*), 304
uwsgi_cache_max_range_offset (*http*), 304
uwsgi_cache_methods (*http*), 305
uwsgi_cache_min_uses (*http*), 305
uwsgi_cache_path (*http*), 305
uwsgi_cache_revalidate (*http*), 306
uwsgi_cache_use_stale (*http*), 307
uwsgi_cache_valid (*http*), 307
uwsgi_connect_timeout (*http*), 308
uwsgi_connection_drop (*http*), 308
uwsgi_force_ranges (*http*), 309
uwsgi_hide_header (*http*), 309
uwsgi_ignore_client_abort (*http*), 309
uwsgi_ignore_headers (*http*), 309
uwsgi_intercept_errors (*http*), 310
uwsgi_limit_rate (*http*), 310
uwsgi_max_temp_file_size (*http*), 310
uwsgi_modifier1 (*http*), 311
uwsgi_modifier2 (*http*), 311
uwsgi_next_upstream (*http*), 311
uwsgi_next_upstream_timeout (*http*), 312
uwsgi_next_upstream_tries (*http*), 312
uwsgi_no_cache (*http*), 313

uwsgi_param (*http*), 313
uwsgi_pass (*http*), 314
uwsgi_pass_header (*http*), 314
uwsgi_pass_request_body (*http*), 314
uwsgi_pass_request_headers (*http*), 314
uwsgi_read_timeout (*http*), 315
uwsgi_request_buffering (*http*), 315
uwsgi_send_timeout (*http*), 315
uwsgi_socket_keepalive (*http*), 315
uwsgi_ssl_certificate (*http*), 316
uwsgi_ssl_certificate_cache (*http*), 316
uwsgi_ssl_certificate_key (*http*), 316
uwsgi_ssl_ciphers (*http*), 317
uwsgi_ssl_conf_command (*http*), 317
uwsgi_ssl_crl (*http*), 318
uwsgi_ssl_name (*http*), 318
uwsgi_ssl_password_file (*http*), 318
uwsgi_ssl_protocols (*http*), 318
uwsgi_ssl_server_name (*http*), 319
uwsgi_ssl_session_reuse (*http*), 319
uwsgi_ssl_trusted_certificate (*http*), 319
uwsgi_ssl_verify (*http*), 319
uwsgi_ssl_verify_depth (*http*), 319
uwsgi_store (*http*), 320
uwsgi_store_access (*http*), 320
uwsgi_temp_file_write_size (*http*), 321
uwsgi_temp_path (*http*), 321

V

valid_referers (*http*), 223
variables_hash_bucket_size (*http*), 366
variables_hash_bucket_size (*stream*), 452
variables_hash_max_size (*http*), 367
variables_hash_max_size (*stream*), 452

W

wamr_global_heap_size, 479
wamr_heap_size, 479
wamr_stack_size, 479
wasm_modules, 482
wasmtime_enable_wasi, 480
wasmtime_stack_size, 480
worker_aio_requests (*core*), 28
worker_connections (*core*), 28
worker_cpu_affinity (*core*), 28
worker_priority (*core*), 29
worker_processes (*core*), 29
worker_rlimit_core (*core*), 30
worker_rlimit_nofile (*core*), 30
worker_shutdown_timeout (*core*), 30
working_directory (*core*), 30

X

xclient, 462
xml_entities (*http*), 329
xslt_last_modified (*http*), 329
xslt_param (*http*), 329
xslt_string_param (*http*), 329
xslt_stylesheet (*http*), 330
xslt_types (*http*), 330

Z

zone (*http*), 292
zone (*stream*), 430