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 *1.12.0*

Web Server 有限责任公司

2026 年 07 月 14 日

目录

1 注解 **2**

2 一般信息 **3**

3 配置 **5**

3.1 一般信息 . . . 5
3.1.1 配置文件 . . . 5
3.1.1.1 继承 . . . 6
3.1.1.2 语法 . . . 7
3.1.1.3 设置哈希 . . . 8
3.1.2 运行时控制 . . . 8
3.1.2.1 使用信号 . . . 9
3.1.2.2 更改配置 . . . 10
3.1.2.3 轮转日志文件 . . . 10
3.1.2.4 在线升级可执行文件 . . . 11
3.1.2.5 命令行选项 . . . 11
3.1.3 连接、会话、请求、日志 . . . 12
3.1.3.1 连接处理机制 . . . 12
3.1.3.2 HTTP 请求处理 . . . 12
3.1.3.3 TCP/UDP 会话处理 . . . 13
3.1.3.4 处理请求 . . . 13
3.1.3.5 代理和负载均衡 . . . 18
3.1.3.6 日志记录 . . . 20
3.2 参考和索引 . . . 21
3.2.1 原生模块 . . . 21
3.2.1.1 核心模块 . . . 21
3.2.1.2 HTTP 模块 . . . 32
3.2.1.3 流模块 . . . 426
3.2.1.4 邮件模块 . . . 529
3.2.1.5 Google PerfTools 模块 . . . 553
3.2.1.6 WASM 模块 . . . 554

3.2.1.7 核心模块 . . . 557
3.2.1.8 HTTP 模块 . . . 557
3.2.1.9 Stream 模块 . . . 560
3.2.1.10 邮件模块 . . . 560
3.2.1.11 Google PerfTools 模块 . . . 560
3.2.1.12 WASM 模块 . . . 561
3.2.2 内置变量 . . . 561
3.2.3 NJS API 参考 . . . 565
3.2.3.1 Angie 对象 . . . 565
3.2.3.2 内置对象 . . . 584
3.2.3.3 Web API . . . 600
3.2.3.4 定时器 . . . 601
3.2.3.5 内置模块 . . . 603
3.2.4 快速访问 Angie 指令和变量 . . . 612
3.2.4.1 HTTP 和核心指令 . . . 612
3.2.4.2 流模块指令 . . . 613
3.2.4.3 变量 . . . 613
3.2.4.4 其他主题 . . . 613
3.3 说明 . . . 615
3.3.1 ACME 配置 . . . 615
3.3.1.1 配置步骤 . . . 615
3.3.1.2 实现细节 . . . 616
3.3.1.3 域名收集与证书使用 . . . 617
3.3.1.4 HTTP 验证 . . . 619
3.3.1.5 DNS 验证 . . . 620
3.3.1.6 ALPN 验证 . . . 621
3.3.1.7 基于钩子的验证 . . . 622
3.3.1.8 Stream 模块中的 ACME . . . 625
3.3.1.9 从 `certbot` 迁移 . . . 627
3.3.2 Angie 集群设置 . . . 628
3.3.2.1 准备集群节点以进行同步 . . . 628
3.3.2.2 安装 Angie 和 angie-ha-sync . . . 629
3.3.2.3 配置配置同步 . . . 630
3.3.2.4 配置 Keepalived . . . 632
3.3.2.5 测试集群运行 . . . 634
3.3.3 OIDC 身份验证设置 . . . 635
3.3.3.1 架构 . . . 635
3.3.3.2 前提条件 . . . 635
3.3.3.3 Google OAuth2 设置 . . . 636
3.3.3.4 配置设置 . . . 636
3.3.3.5 部署步骤 . . . 638
3.3.3.6 安全配置 . . . 639
3.3.3.7 身份验证流程 . . . 640
3.3.3.8 高级配置 . . . 640
3.3.4 SSL 配置 . . . 641

3.3.4.1 HTTPS 服务器优化 . . . 642
3.3.4.2 SSL 证书链 . . . 642
3.3.4.3 单个 HTTP/HTTPS 服务器 . . . 644
3.3.4.4 基于名称的 HTTPS 服务器 . . . 644
3.3.4.5 具有多个名称的 SSL 证书 . . . 645
3.3.4.6 服务器名称指示 . . . 645
3.3.5 Console Light Web 监控面板 . . . 646
3.3.5.1 Console Light . . . 646
3.3.5.2 版本历史 . . . 647
3.3.5.3 安装和配置 . . . 647
3.3.5.4 界面 . . . 649
3.3.6 自定义指标配置 . . . 663
3.3.6.1 配置步骤 . . . 663
3.3.6.2 示例 . . . 663
3.3.6.3 注意事项 . . . 664
3.3.7 从 nginx 迁移到 Angie . . . 664
3.3.7.1 安装 Angie . . . 664
3.3.7.2 更新 Angie 配置 . . . 664
3.3.7.3 测试和切换 . . . 668
3.3.7.4 使用 SSL 证书 . . . 669
3.3.7.5 适配行为不同的指令 . . . 670
3.3.7.6 配置 Angie 功能 . . . 670
3.3.8 nginx 不支持的指令 . . . 670
3.3.8.1 已删除或省略 . . . 670
3.3.8.2 已重命名 . . . 670
3.3.8.3 已弃用 . . . 670
3.3.9 配置 Grafana 仪表板 . . . 671
3.4 社区材料 . . . 672
3.4.1 文章 . . . 672
3.4.2 课程 . . . 672
3.4.3 实用指南 . . . 672
3.4.4 访谈和播客 . . . 672
3.5 面向 AI 助手的文档 . . . 672

4 故障排除 674
4.1 调试日志 . . . 674
4.1.1 指令位置 . . . 676
4.1.2 记录特定地址 . . . 678
4.1.3 循环内存缓冲区 . . . 678
4.2 核心转储 . . . 678
4.2.1 Linux: systemd . . . 678
4.2.2 Linux: 手动配置 . . . 679
4.2.3 FreeBSD . . . 680

5 知识产权 681

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 682

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

注解

本文档包含操作 Angie PRO 软件所需的必要信息。

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 服务器是一类通过 HTTP 协议向最终用户提供网络资源访问的软件。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、与地铁自助服务终端交互，甚至与”Sapsan” 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**。使用 C 语言编写。
- **L4-L7 负载均衡器**。允许在服务器之间对 TCP/UDP 协议和 HTTP 进行负载均衡。
- **代理和缓存服务器**。通过灵活的缓存机制实现 Web 服务的更快运行。
- **在所有流行平台上可用**。在 Alpine、Debian、Oracle、RED OS、Rocky 和 Ubuntu 上编译和测试。
- **高性能**。世界上最高效的 Web 服务器之一。

选择 Angie PRO 的理由:

- **与 NGINX OSS 兼容**。 Angie PRO 与 Nginx 完全兼容，允许任何现有的 Nginx 用户无需大量成本或服务停机即可过渡到 Angie PRO。
- **增强的统计和实时监控**。 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进行动态配置管理，并确保服务的完全可用性。
- **代理服务器组的动态配置**。允许通过便捷的 REST 接口管理代理服务器组设置，无需服务中断。
- **缓存元素删除**。提供通过用户友好的 API 删除缓存元素的能力，无需服务停机。
- **代理服务器的主动健康探测**。检查” 存活性”，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 实现 Angie PRO 配置变量的动态管理。
- **动态 DNS 更新**。
- **会话亲和性代理**。
- **包含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，保证功能和支持。
- **共享内存区同步**。能够在 Angie PRO 集群中使用缓存区、limit_req 等。
- **隐藏或个性化响应头中的服务器名称**。能够向用户更改或隐藏 Web 服务器的名称和版本。

与 Angie PRO 具有类似功能特性的国外软件列表包括 Nginx、Nginx Plus、Apache、Envoy、使用 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

本页面包含用于配置 Angie 的文章、参考、索引和说明。

3.1 一般信息

这些文章涵盖 Angie 的安装和配置、启动和停止 Web 服务器、管理服务器, 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;
```

3.1.1.1 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。

3.1.1.2 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";
```

带变量的字符串

许多指令接受带变量的字符串（也称为复杂值）：一种可以将字面文本与变量引用混合的参数值，变量引用例如 $name、${name} 以及正则表达式的位置捕获变量 $1 到 $9。此类值在加载配置时仅编译一次，并针对每个请求或连接重新求值，因此生成的字符串可能因请求而异。不包含 $ 的值表现为普通字面量，不会带来每个请求的额外开销。

使用花括号形式 `${name}` 可将变量名与紧随其后的文本分隔开，例如 `${name}_suffix`。位置捕获变量 `$1` 到 `$9` 仅在相应分组的正则表达式匹配之处可用，并且需要使用 PCRE 进行构建。

`$` 不支持反斜杠转义。要插入字面的 `$`，请定义一个包含它的变量，例如使用*map* 或*set*，然后在值中引用该变量。

可用变量的集合取决于上下文：*http* 子系统中的变量不同于*stream* 子系统中的变量；完整列表请参见变量索引。变量既由内置模块生成，也由*map*、*geo* 和*split_clients* 等指令生成。

对于某些指令，变量的存在会改变运行时行为：此时该值会针对每个请求动态解析，而不是在加载配置时固定下来，这可能需要*resolver*。指令*proxy_pass* 就是这样一种情况。

3.1.1.3 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** 执行以下命令:

```
$ sudo service angie start
```

建议事先检查配置语法。方法如下:

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

重新加载配置:

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

停止 Angie:

```
$ sudo service angie stop
```

安装后, 运行以下命令以确保 Angie 已启动并正在运行:

```
$ curl localhost:80
```

备注

开源版本 Angie 的运行方法可能因安装方式而异。

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。工作进程处理实际的请求。Angie 使用基于事件的模型和依赖于操作系统的机制, 在工作进程之间高效分配请求。工作进程的数量在配置文件中定义, 可以针对给定配置固定, 也可以根据可用 CPU 核心数自动调整 (参见*worker_processes*)。

配置后,Angie 还会在退出前将某些共享内存区域 (目前是*proxy_cache_path* 中的 `keys_zone`) 刷新到磁盘, 以便新的主进程可以恢复它们, 从而提高性能。如果由于区域大小更改、二进制版本不兼容或其他原因导致恢复失败,Angie 将记录警告 (`failed to restore zone at address`), 并且不会使用区域恢复机制。

3.1.2.1 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 会写入文件 `/run/angie.pid`。此文件名可以在配置时或在 `angie.conf` 中使用*pid* 指令更改。主进程支持以下信号:

TERM、INT	快速关闭
QUIT	优雅 关闭
HUP	重新加载配置, 更新时区 (仅适用于 FreeBSD 和 Linux), 使用更新后的配置启动新的工作进程,:ref: 优雅 *<worker_shutdown_timeout>* 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 关闭工作进程

您可以使用 `kill` 发送信号:

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

也可以使用信号控制单个工作进程, 尽管这是可选的。支持的信号有:

TERM、INT	快速关闭
QUIT	优雅 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*)

3.1.2.2 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, 请求它们优雅 关闭。旧的工作进程关闭其监听套接字并继续为现有客户端提供服务。在所有客户端都得到服务后, 旧的工作进程将关闭。

Angie 跟踪每个进程的配置更改。代数从服务器首次启动时的 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 v1.12.0 #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 会为接收到新配置的进程递增代数:

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

   angie: master process v1.12.0 #2 [angie]
   angie: worker process #2
```

如果前几代的任何工作进程继续运行, 它们将立即可见:

```
$ ps aux | grep angie

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

备注

不要将配置代数与” 进程编号” 混淆; 出于实际目的,Angie 不使用连续的进程编号。

3.1.2.3 轮转日志文件

要轮转日志文件, 首先重命名文件。然后, 向主进程发送 USR1 信号。主进程将重新打开所有当前打开的日志文件, 并将它们分配给运行工作进程的非特权用户。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其日志文件。工作进程也将立即打开新文件并关闭旧文件。因此, 旧文件几乎立即可用于后处理, 例如压缩。

3.1.2.4 在线升级可执行文件

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。主进程将把其当前带有进程 ID 的文件重命名为带有 .oldbin 后缀的新文件, 例如 /usr/local/angie/logs/angie.pid.oldbin, 然后启动新的可执行文件, 新可执行文件又会启动新的工作进程。

请注意, 旧的主进程不会关闭其监听套接字, 如有必要, 可以管理它以重新启动其工作进程。如果新的可执行文件未按预期执行, 您可以采取以下操作之一:

- 向旧的主进程发送 HUP 信号。这将在不重新读取配置的情况下启动新的工作进程。然后, 您可以通过向新的主进程发送 QUIT 信号来优雅 关闭所有新进程。
- 向新的主进程发送 TERM 信号。它将向其工作进程发送消息, 请求它们立即退出。如果有任何进程未退出, 请发送 KILL 信号强制它们退出。当新的主进程退出时, 旧的主进程将自动启动新的工作进程。

如果新的主进程退出, 旧的主进程将从带有进程 ID 的文件名中删除 .oldbin 后缀。

如果升级成功, 向旧的主进程发送 QUIT 信号, 只有新进程将保留。

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) 刷新到磁盘, 以便新的主进程可以恢复它们, 从而提高性能。如果由于区域大小更改、二进制版本不兼容或其他原因导致恢复失败,Angie 将记录警告 (failed to restore zone at address), 并且不会使用区域恢复机制。

3.1.2.5 命令行选项

-?、-h	显示命令行参数的帮助信息, 然后退出。
--build-env	显示有关构建环境的辅助信息, 然后退出。
-c *file*	使用 *file* 作为配置文件, 而不是使用默认文件。
-e *file*	使用 *file* 作为错误日志文件, 而不是使用默认文件。特殊值 stderr 指定标准错误输出。
-g *directives*	设置全局配置指令, 例如:angie -g "pid /var/run/angie.pid; worker_processes `sysctl -n hw.ncpu`;"。
-m、-M	显示内置模块 (-m) 或内置和已加载模块 (-M) 的列表, 然后退出。
-p *prefix*	为 angie 使用指定的 *prefix* 路径 (服务器文件所在的目录; 默认为 /usr/local/angie/)。
-q	如果设置了 -t 或 -T, 则仅显示错误消息; 否则, 无效果。
-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, 递归包含其中提到的文件。
-T	与 -t 相同, 但在递归包含配置中提到的所有文件后, 还会将汇总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

3.1.3.1 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 通常会自动选择最高效的方法。但是, 如有必要, 可以使用*use* 指令显式选择连接处理方法。

以下是可用的连接处理方法:

方法	描述
`select`	标准方法。在没有更高效方法的平台上会自动构建支持模块。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`	标准方法。在没有更高效方法的平台上会自动构建支持模块。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`	高效方法, 可用于 FreeBSD 4.1+、OpenBSD 2.9+、NetBSD 2.0 和 macOS。
`epoll`	高效方法, 可用于 Linux 2.6+。
`/dev/poll`	高效方法, 可用于 Solaris 7 11/99+、HP/UX 11.22+ (eventport)、IRIX 6.5.15+ 和 Tru64 UNIX 5.1A+。
`eventport`	`event ports` 方法可用于 Solaris 10+。(由于已知问题, 建议改用 `/dev/poll` 方法。)

3.1.3.2 HTTP 请求处理

HTTP 请求会经历一系列阶段, 在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`	处理*Rewrite* 模块中定义在 `server` 块 (但在 `location` 块之外) 的指令的阶段。
`Find-config`	特殊阶段, 根据请求 URI 选择*location*。
`Rewrite`	类似于 `Server-rewrite` 阶段, 但它应用于在上一阶段选择的 location 块中定义的*rewrite* 规则。
`Post-rewrite`	特殊阶段, 如果在 `Rewrite` 阶段修改了请求的 URI, 则将请求重定向到新的 location, 如同 `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`	验证客户端是否有权发出请求的阶段, 通常通过调用标准 Angie 模块 (如*Auth Basic*) 来实现。
`Post-access`	特殊阶段, 处理*satisfy any* 指令。
`Precontent`	标准模块指令 (如*try_files* 和*mirror*) 在此阶段注册它们的处理程序。
`Content`	通常生成响应的阶段。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* 指令也在此处理。 处理程序按顺序调用, 直到其中一个产生输出。
`Log`	最后阶段, 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以进行访问日志记录。

3.1.3.3 TCP/UDP 会话处理

来自客户端的 TCP/UDP 会话会经历一系列阶段, 在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`	检查访问权限的预备阶段。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`	发生 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预读缓冲区 的阶段, 以允许模块 (如*SSL Preread*) 在处理之前分析数据。
`Content`	实际处理数据的强制阶段, 通常涉及*Return* 模块向客户端发送响应。*proxy_pass* 指令也在此处理。
`Log`	记录客户端会话处理结果的最后阶段。在此阶段调用*Log* 模块。

3.1.3.4 处理请求

虚拟服务器选择

最初, 连接是在默认服务器的上下文中创建的。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 握手期间, 根据 SNI 提前确定。
- 处理请求行之后。
- 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。

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

- 对于*ssl_protocols* 指令, 协议列表由 OpenSSL 库在根据通过 SNI 请求的名称应用服务器配置之前设置。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, 这些指令使用默认服务器配置或由 SNI 选择的服务器配置。
- 对于*ignore_invalid_headers*、*large_client_header_buffers* 和*underscores_in_headers* 指令, 它们参与处理请求头字段, 服务器配置还取决于是否根据请求行或 `Host` 头字段进行了更新。
- 错误响应使用当前正在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该处理请求。考虑一个简单的配置, 其中所有三个虚拟服务器都监听端口 80:

```
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该处理请求。如果此头的值与任何服务器名称都不匹配, 或者请求不包含此头字段,Angie 将把请求路由到此端口的默认服务器。在上面的配置中, 默认服务器是第一个——这是 Angie 的标准默认行为。也可以使用*listen* 指令中的 `default_server` 参数显式指定哪个服务器应该是默认服务器:

```
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 (IDN) 应在*server_name* 指令中使用 ASCII(Punycode) 表示形式指定:

```
server {

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

阻止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, 可以定义一个简单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, 它匹配没有 `Host` 头字段的请求。然后返回一个特殊的非标准代码 444, 该代码会关闭连接。

组合基于名称和基于 IP 的虚拟服务器

让我们来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 首先根据*server* 块的*listen* 指令测试请求的 IP 地址和端口。然后, 它根据匹配 IP 地址和端口的*server* 块的*server_name* 条目测试请求的 `Host` 头字段。如果未找到服务器名称, 该请求将由默认服务器处理。例如, 在端口 192.168.1.1:80 上接收到的对 `www.example.com` 的请求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` 未为此端口定义。

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`，而不考虑它们的列出顺序。在上面的配置

中，唯一的前缀 location 是 `location /`，它匹配任何请求，并将作为最后的选择。然后 Angie 按照配置文件中出现的顺序检查由正则表达式定义的 location。第一个匹配的表达式会停止搜索，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 location 仅测试请求行的 URI 部分，不包括参数。这是因为查询字符串中的参数可以以各种方式指定，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` 首先由前缀 `location /` 匹配，然后由正则表达式 `.(gif|jpg|png)$` 匹配。因此，它由后者 location 处理。使用指令 `root /data/www`，请求被映射到文件 `/data/www/logo.gif`，并将文件发送给客户端。
- 请求 `/index.php` 也首先由前缀 `location /` 匹配，然后由正则表达式 `.(php)$` 匹配。因此，它由后者 location 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 服务器执行该文件。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` 仅由前缀 `location /` 匹配，因此在此 location 中处理。使用指令 `root /data/www`，请求被映射到文件 `/data/www/about.html`，并将文件发送给客户端。

处理请求 `/` 更为复杂。它仅由前缀 `location /` 匹配，因此由此 location 处理。然后*index* 指令根据其参数和 `root /data/www` 指令测试索引文件是否存在。如果文件 `/data/www/index.html` 不存在但文件 `/data/www/index.php` 存在，该指令执行到 `/index.php` 的内部重定向，Angie 再次搜索 location，就像请求是由客户端发送的一样。如前所述，重定向的请求最终将由 FastCGI 服务器处理。

上述示例通过*root* 指令将请求映射到文件，但完整的映射涉及多个指令的协同作用。一旦匹配到*location*，Angie 将请求规范化后的*$uri* 映射到文件系统路径：*root* 指令将完整 URI 追加到其值，而*alias* 指令则替换已匹配的 location 前缀。*try_files* 指令随后相对于该路径逐一探测候选文件；若均不存在，其最后一个参数将触发内部重定向，Angie 再次根据新的*$uri* 搜索 location，与上文索引重定向的机制相同。

对于相同的请求 `/images/top.gif`，以下两个 location 均将其解析到同一文件 `/data/w3/images/top.gif`：

```
location /images/ {
  root /data/w3;           # 追加：/data/w3 + /images/top.gif
}

location /images/ {
```

```
  alias /data/w3/images/;  # 替换：/images/ -> /data/w3/images/，再加 top.gif
}
```

使用*alias*时，匹配的前缀按其长度被移除，因此 location 末尾斜杠与 alias 值末尾斜杠必须保持一致。若将上面 alias 路径末尾的斜杠去掉，结果路径将变为 `/data/w3/imagestop.gif`。

3.1.3.5 代理和负载均衡

Angie 的一个常见用途是将其设置为代理服务器。在此角色中，Angie 接收请求，将它们转发到被代理的服务器，从这些服务器检索响应，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端 `backend:8080` （被代理的服务器）。还有许多其他指令可用于进一步配置代理连接。

FastCGI 代理

Angie 可用于将请求路由到运行使用各种框架和编程语言（如 PHP）构建的应用程序的 FastCGI 服务器。

与 FastCGI 服务器配合使用的最基本的 Angie 配置涉及使用*fastcgi_pass*指令代替*proxy_pass*指令，以及使用*fastcgi_param*指令设置传递给 FastCGI 服务器的参数。假设 FastCGI 服务器可在 `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。生成的配置如下：

```
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，该服务器将所有请求（静态图像请求除外）通过 FastCGI 协议路由到在 `localhost:9000` 上运行的被代理服务器。

WebSocket 代理

要将连接从 HTTP/1.1 升级到 WebSocket，需要使用 HTTP/1.1 中提供的 '协议切换 <https://datatracker.ietf.org/doc/html/rfc2616#section-14.42>'_ 机制。

然而，这里有一个微妙之处：由于 `Upgrade` 头是一个 '逐跳头 <https://datatracker.ietf.org/doc/html/rfc2616#section-13.5.1>'_，它不会从客户端传递到被代理的服务器。在正向代理中，客户端可以使用 CONNECT 方法来规避这个问题。但这种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的存在，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，如果被代理的服务器返回代码为 101（切换协议）的响应，并且客户端通过请求中的 `Upgrade` 头请求协议切换，则允许在客户端和被代理服务器之间建立隧道。

如前所述，逐跳头（包括 `Upgrade` 和 `Connection`）不会从客户端传递到被代理的服务器。因此，为了让被代理的服务器知道客户端切换到 WebSocket 协议的意图，必须显式传递这些头：

```
location /chat/ {

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

一个更复杂的示例演示了如何根据客户端请求头中是否存在 `Upgrade` 字段，来决定发送到被代理服务器的请求中 `Connection` 头字段的值：

```
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，如果被代理的服务器在 60 秒内没有传输任何数据，连接将被关闭。可以使用*proxy_read_timeout* 指令增加此超时时间。或者，可以将被代理的服务器配置为定期发送 WebSocket ping 帧以重置超时并检查连接是否仍然活动。

负载均衡

在多个应用程序实例之间进行负载均衡是一种广泛使用的技术，用于优化资源利用率、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程序服务器，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;
        }
    }
}
```

在上面的示例中，同一应用程序的三个实例分别运行在 `srv1` 到 `srv3` 上。当没有显式配置负载均衡方法时，默认使用轮询。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）服务器健康探测。这些探测使用*upstream* 上下文中*server* 块内的*max_fails* 和*fail_timeout* 指令进行配置。

3.1.3.6 日志记录

备注

除了此处列出的选项外，您还可以启用调试日志。

Syslog

error_log 和*access_log* 指令支持记录到 syslog。以下参数用于配置到 syslog 的日志记录：

server=*address*	指定 syslog 服务器的地址。地址可以是域名或 IP 地址（带可选端口），或在 "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 消息的设施，如 RFC 3164 中定义。可能的设施包括："kern"、"user"、"mail"、"daemon"、"auth"、"intern"、"lpr"、"news"、"uucp"、"clock"、"authpriv"、"ftp"、"ntp"、"audit"、"alert"、"cron"、"local0".."local7"。默认值为 "local7"。
severity=*string*	定义*access_log* 的 syslog 消息的严重性级别，如 RFC 3164 中指定。可能的值与*error_log* 指令的第二个参数（级别）相同。默认值为 "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。默认标签为 "angie"。
nohostname	禁止在 syslog 消息头中添加 hostname 字段。

syslog 配置示例：

```
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;
```

3.2 参考和索引

这些摘要部分提供有关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 原生模块

本指南介绍 Angie 的原生模块, 提供配置示例, 列出其指令和参数, 以及内置变量。

3.2.1.1 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, 并处理关键任务, 如管理工作进程、配置事件驱动模型、以及处理传入的连接和请求。它包含用于设置主进程、错误日志记录以及在底层控制服务器行为的关键指令。

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。如果不设置此项, 所有工作进程都会收到新连接的通知, 这可能导致系统资源的低效使用, 特别是在新连接数量较少的情况下。

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正在处理新连接时, 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。您可以通过 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:;
    #  ...
}
```

备注

要使此指令生效，必须在构建 Angie 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，例如在工作进程重启期间发生套接字泄漏，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，但 `TZ` 变量除外。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：

- 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，因为库通常仅在初始化期间检查变量，而初始化发生在此指令生效之前。TZ 变量始终被继承并可供*Perl* 模块访问，除非明确配置为其他方式。

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，不应由用户直接设置。

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 配置级别未明确定义日志文件，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。要配置到*syslog* 的日志记录，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相同和更高严重性的消息：

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，将使用 error 作为默认日志级别。

备注

要使 debug 日志级别生效，必须在构建 Angie 时启用调试日志。

错误日志中的每个条目具有以下格式:

```
timestamp [level] PID#TID: *connection_id message
```

其中:

- `timestamp` —事件的日期和时间
- `level` —事件的日志级别
- `PID#TID` —进程和线程标识符
- `*connection_id` —连接序列号, 同一连接上的所有请求共用 (如果适用)
- `message` —错误或事件消息文本

events

语法	`events { ... };`
默认值	—
上下文	main

为影响连接处理的指令提供配置文件上下文。

include

语法	`include` *file* | *mask*;
默认值	—
上下文	any

将另一个文件或与指定 *mask* 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` *file*;
默认值	—
上下文	main

从指定文件加载动态模块。如果提供的是相对路径, 则基于 `--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;
```

如果动态模块是为不同的 Angie 构建版本构建的, 加载将失败并显示类似以下的错误:"module "..." was built for "..." but you are running "Angie"""。

lock_file

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

Angie 使用锁机制来实现*accept_mutex* 并序列化对共享内存的访问。在大多数系统上, 锁是使用原子操作管理的, 因此不需要此指令。但是, 在某些系统上, 会使用替代的 *lock file* 机制。此指令设置锁文件名的前缀。

master_process

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

决定是否启动工作进程。此指令供 Angie 开发人员使用。

multi_accept

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

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

备注

如果使用*kqueue* 连接处理方法, 则忽略此指令, 因为它会提供准备接受的新连接数量。

pcre_jit

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

为配置解析时已知的正则表达式启用或禁用" 即时编译"(PCRE JIT)。

PCRE JIT 可以显著加速正则表达式处理。

备注

JIT 在 PCRE 库 8.20 版本及以上可用, 前提是使用 `--enable-jit` 配置选项构建。当使用 PCRE 库构建 Angie 时 (`--with-pcre=`), 使用 `--with-pcre-jit` 选项启用 JIT 支持。

pid

语法	`pid` *file*	`off;`
默认值	`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 *file*。该文件以原子方式创建, 这确保其内容始终正确。`off` 设置禁用创建此文件。

备注

如果在重新配置期间修改了 *file* 设置, 但它指向先前 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;`
默认值	`ssl_object_cache_inheritable on;`
上下文	main

如果启用,SSL 对象 (SSL 证书、私钥、受信任的 CA 证书、CRL 列表) 在配置重新加载时会被继承。

如果从文件加载的 SSL 对象的修改时间和文件索引自上次配置加载以来没有更改, 则会被继承。指定为 `engine:name:id` 的私钥永远不会被继承, 而指定为 `data:value` 的私钥始终会被继承。

从变量加载的 SSL 对象无法被继承。

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` *name* `threads=`*number* [`max_queue=`*number*];
默认值	`thread_pool default threads=32 max_queue=65536;`
上下文	main

定义线程池的 *name* 和参数, 用于多线程读取和发送文件, 不阻塞 工作进程。

`threads` 参数定义池中的线程数。

如果池中的所有线程都忙于执行任务, 新任务将在队列中等待。`max_queue` 参数限制允许在队列中等待的任务数量。默认情况下, 队列中最多可以有 65536 个任务。当队列溢出时, 任务将以错误完成。

timer_resolution

语法	`timer_resolution` *interval*;
默认值	—
上下文	main

降低工作进程中的计时器分辨率, 从而减少 `gettimeofday()` 系统调用的次数。默认情况下, 每次接收到内核事件时都会调用 `gettimeofday()`。降低分辨率后,:samp:*gettimeofday()* 仅在指定间隔内调用一次。

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()`;
- 否则使用 `setitimer()`。

use

语法	`use` *method*;
默认值	—
上下文	events

指定用于连接处理 的 *method*。通常不需要显式指定, 因为 Angie 默认会使用最高效的方法。

user

语法	`user` *user* [*group*];
默认值	`user < 构建参数 --user> < 构建参数 --group>;`
上下文	main

定义工作进程使用的 *user* 和 *group* 凭据 (另请参阅 构建参数)。如果省略 *group*, 则使用与 *user* 同名的组。

worker_aio_requests

语法	`worker_aio_requests` *number*;
默认值	`worker_aio_requests 32;`
上下文	events

当使用*aio* 和*epoll* 连接处理方法时, 设置单个工作进程的最大未完成异步 I/O 操作数。

worker_connections

语法	`worker_connections` *number*;
默认值	`worker_connections 512;`
上下文	events

设置工作进程可以打开的最大同时连接数。

应该记住, 此数字包括所有连接 (例如与代理服务器的连接等), 而不仅仅是与客户端的连接。另一个考虑因素是, 实际的同时连接数不能超过当前打开文件数的最大限制, 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*];
默认值	—
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 位掩码表示。应该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, 将第二个工作进程绑定到 CPU1 和 CPU3。此设置适用于超线程。

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可用于自动绑定的 CPU:

```
worker_cpu_affinity auto 01010101;
```

备注

该指令仅在 FreeBSD 和 Linux 上可用。

worker_priority

语法	`worker_priority` *number*;
默认值	`worker_priority 0;`
上下文	main

定义工作进程的调度优先级, 类似于 `nice` 命令的做法: 负数 *number* 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` *number* | `auto`;
默认值	`worker_processes 1;`
上下文	main

定义工作进程的数量。

最佳值取决于许多因素, 包括 (但不限于) CPU 核心数、存储数据的硬盘驱动器数量以及负载模式。如果不确定, 将其设置为可用 CPU 核心数是一个不错的起点 (值"`auto`" 将尝试自动检测)。

worker_rlimit_core

语法	`worker_rlimit_core` *size*;
默认值	—
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` *number*;
默认值	—
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` *time*;
默认值	—
上下文	main

配置工作进程优雅关闭的超时时间 (以秒为单位)。当指定的时间到期时, Angie 将尝试关闭当前打开的所有连接以便于关闭。

优雅关闭通过向主进程发送*QUIT 信号* 来启动，该信号指示工作进程停止接受新连接并允许现有连接完成。工作进程继续处理活动请求直到完成，然后优雅地关闭。如果连接保持打开的时间超过 `worker_shutdown_timeout`,Angie 将强制关闭这些连接以完成关闭。此外，客户端保持活动连接仅在空闲时间至少达到*lingering_timeout* 指定的时间时才会关闭。

working_directory

语法	`working_directory` *directory*;
默认值	—
上下文	main

定义工作进程的当前工作目录。它主要在写入核心文件时使用，在这种情况下，工作进程应该对指定的目录具有写入权限。

3.2.1.2 HTTP 模块

Access

该模块基于客户端 IP 地址或网络控制对服务器资源的访问。它允许对特定 IP 地址、IP 范围或 UNIX 域套接字允许或阻止访问，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。

还可以通过使用*Auth Basic* 模块的密码或基于*Auth Request* 模块的子请求结果来限制访问。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 IPv4 网络 `10.1.1.0/16` 和 `192.168.1.0/24` (不包括特定地址 `192.168.1.1`) 以及 IPv6 网络 `2001:0db8::/32` 访问。当有许多规则时，最好使用*Geo* 模块中的变量。

指令

allow

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

允许指定网络或地址的访问。特殊值 `all` 表示所有客户端 IP 地址。

特殊值 `unix:` 允许任何 UNIX 域套接字的访问。

deny

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

拒绝指定网络或地址的访问。特殊值 `all` 表示所有客户端 IP 地址。

特殊值 `unix:` 拒绝任何 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

在此示例中, 名为 `example` 的 ACME 客户端使用默认的 HTTP 验证, 为 `example.com` 和 `www.example.com` 自动获取并续期证书:

```
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; # 如果没有服务器监听 HTTP 挑战端口,则可选
                   # (参见 'acme_http_port' 指令)

        listen 443 ssl;

        server_name example.com www.example.com;
        acme example;

        ssl_certificate $acme_cert_example;
        ssl_certificate_key $acme_cert_key_example;
    }
}
```

其他验证方法（DNS、ALPN、基于钩子）以及详细的设置说明，请参阅*ACME 配置* 部分。

指令

acme

语法	acme 名称;
默认值	—
上下文	server

指定为此*server* 块中的有效证书标识符获取证书的*ACME 客户端*。单个证书覆盖所有引用该 名称客户端的*server* 块的*server_name* 指令指定的所有有效域名和 IP 地址; 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标识符请求新证书。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

该指令仅控制证书请求中包含哪些有效证书标识符, 并不影响证书可在何处使用。任何 server 块都可以通过*$acme_cert_<name>* 变量引用该证书, 无论该块是否包含 acme 指令。从 server 块中移除 acme 仅会将该块的*server_name* 值排除在后续的证书请求之外, 但不会阻止该块使用证书。

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。

除非客户端使用 challenge=dns, 否则支持 IPv4 和 IPv6 地址。启用 DNS 验证后,IP 地址将被跳过。

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

在 1.11.0 版本发生变更.

在 1.12.0 版本发生变更.

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] [`max_key_auth_size=` 大小] [`renew_before_expiry=` 时间] [`renew_on_load`] [`retry_after_error=off`| 时间] [`challenge=dns` | `http` | `alpn`] [`profile=`*name*] [`account_key=` 文件] [`eab=`*id*[*:alg*]*:key*];
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。

每个客户端只管理一张证书; 如需获取多张独立证书, 请配置多个 `acme_client` 块 (参见为不同域名申请独立证书)。

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, 可以向其添加来自代理模块的其他设置。

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 <https://letsencrypt.org/docs/staging-environment/>'_ 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, 这很有用。 默认值:`on`。
`key_type`	证书的私钥算法类型。有效值:`rsa`,:samp:*ecdsa*。 默认值:`ecdsa`。
`key_bits`	证书密钥的位数。默认值:`ecdsa` 为 256,:samp:*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。此参数不限制 ACME 服务器响应的大小; 请使用*acme_max_response_size* 来限制响应大小。 如果未设置该参数,Angie 会根据配置的域名列表计算大致大小, 并将其用于共享内存分配。 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。 如果更新过程中获得的证书大小超过 `max_cert_size`, 则更新过程将失败并报错。 默认值: 自动计算。
`max_key_auth_siz`	限制 Angie 为 ACME 验证存储在共享内存中的密钥授权字符串的大小。如果 ACME 服务器返回的密钥授权字符串超过此值, 请求将失败并报错, 提示增大 `max_key_auth_size`。 虽然在 `acme_client` 行中指定, 但这是 `http` 块中所有客户端共享的单一设置。 默认值:`2k`。
`renew_before_exp`	证书到期前应开始更新的时间。 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`	在证书获取失败时重试前等待的时间。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`	指定 ACME 客户端的验证类型。有效值:`dns`,:samp:*http*,:samp:*alpn*。 `alpn` 值启用 TLS-ALPN-01 验证, 需要 Angie 使用支持 ALPN 的 OpenSSL 构建 (不支持 BoringSSL 或 AWS-LC 构建)。 默认值:`http`。
`profile`	向证书颁发机构请求特定的 ACME 配置档案 (profile)。配置档案是 CA 定义的证书和验证设置的变体, 例如证书的有效期, 或可请求的标识符类型。 例如,Let's Encrypt 在其 配置档案文档 中介绍了可用的配置档案。 ACME 服务器必须在其目录元数据中公布此配置档案, 否则客户端将无法获取证书。
`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, 那么另一个客户端必须在配置中更早出现。 此外, 密钥仅为设置了 `enabled=on` 参数的客户端创建。
`eab`	配置 外部账户绑定 (EAB), 将 ACME 账户绑定到已在证书颁发机构 (CA) 注册的账户。

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 构建参数 `--http-acme-client-path` 设置。

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 的主进程必须以超级用户权限运行。

acme_dns_ttl

Added in version 1.12.0.

语法	acme_dns_ttl 数字;
默认值	acme_dns_ttl 1;
上下文	http

设置模块在响应 ACME 服务器的*DNS 验证* 查询时返回的 TXT 记录的 TTL(以秒为单位)。根据 RFC 2181 的规定, 取值范围为 0 到 2147483647。

acme_hook

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

为由 名称指定的*ACME 客户端* 启用基于钩子的域名验证。当证书签发或续订需要进行域名验证时, Angie

会向放置该指令的命名 `location` 生成一个内部请求。请求的处理方式完全取决于同一 `location` 中配置的其他指令，例如*fastcgi_pass*、*proxy_pass*，或任何其他请求处理程序。

名称	由该钩子处理域名验证的*ACME 客户端* 的名称。
uri	包含变量的字符串；指定钩子调用的请求 URI。 默认值:`/`。

例如，以下配置将*hook 变量* 的值通过请求 URI 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 ...;
```

acme_http_port

Added in version 1.11.0.

在 1.11.1 版本发生变更.

语法	`acme_http_port` 端口 | *ip*[*:* 端口] | *[ip6]*[*:* 端口];
默认值	`acme_http_port 80;`
上下文	http

指定模块用于处理 HTTP ACME 验证请求的端口。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_http_port 8080;
acme_http_port 127.0.0.1;
acme_http_port [::1];
```

如果没有服务器配置为监听指定的地址和端口，模块会为 HTTP 验证创建专用监听器。

要使用 1024 或更低的端口号，Angie 的主进程必须以超级用户权限运行。

acme_max_response_size

Added in version 1.11.0.

语法	`acme_max_response_size` 大小;
默认值	`acme_max_response_size 32k;`
上下文	http

限制 ACME 服务器响应正文的最大大小。如果响应超过此限制, 请求将失败并报错。如果您看到类似 `too big subrequest response while sending to client` 的错误, 请增加该值。

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。

备注

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`,:samp:*http*,:samp:*alpn*。

`$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_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。

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。

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

`API` 模块实现了一个 HTTP RESTful 接口, 用于以 JSON 格式获取 Web 服务器的基本信息, 以及关于客户端连接、共享内存区域、DNS 查询、HTTP 请求、HTTP 响应缓存、*stream* 模块会话, 以及*limit_conn http*、*limit_conn stream*、*limit_req* 和*http upstream* 模块区域的统计信息，以及关于在配置中指定的和由 ACME 客户端获取的证书的信息。

备注

在 https://console.angie.software/api/ 查看 API 输出的实时示例。

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

```
{
    "error": "MethodNotAllowed",
    "description": "The POST method is not allowed for the requested API element \"/\"."
}
```

在 Angie PRO 中, 此接口包含一个动态配置 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*upstream* 中的各个服务器。

指令

api

语法	`api` *path*;
默认值	—
上下文	location

在 `location` 中启用 HTTP RESTful 接口。

path 参数是必需的。与*alias* 指令类似, 它设置用于替换 `location` 中指定路径的路径, 但是是在 API 树上而不是文件系统上。

如果在前缀 `location` 中指定:

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

请求 URI 中与前缀 */stats/* 匹配的部分将被替换为 *path* 参数中指定的路径:*/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, 您可以分离动态配置 *API* 和反映当前状态的不可变状态 *API*:

```
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;
}
```

备注

如果 `api` 放置在前缀带有尾部斜杠的 `location` 中 (例如, `location /name/`), 并且*auto_redirect* 指令设置为 `default`, 则不带尾部斜杠的请求将被重定向 (`/name -> /name/`)。

api_config_files

语法	`api_config_files on	off;`
默认值	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/` 的请求返回大致如下内容:

```
{
    "version":"1.12.0",
    "address":"192.168.16.5",
    "generation":1,
    "load_time":"2026-07-13T12:58:39.789Z",
    "config_files": {
        "/etc/angie/angie.conf": "...",
        "/etc/angie/mime.types": "..."
    }
}
```

默认情况下, 输出被禁用, 因为配置文件可能包含特别敏感的机密信息。

指标

Angie 在 `/status/` API 部分发布使用统计信息; 您可以通过设置适当的 `location` 来开放访问权限。完全访问:

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

部分访问的示例，如上所示：

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

配置示例

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

```
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;
        proxy_cache_valid 200 10m;

        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/`，Angie 返回：

JSON 树


```
{
    "angie": {
        "version":"1.12.0",
        "address":"192.168.16.5",
        "generation":1,
        "load_time":"2026-07-13T12:58:39.789Z"
    },

    "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
            }
        }
    }
}
```

可以通过构造适当的请求，按单个 JSON 分支请求一组指标。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/acme_clients
$ curl https://www.example.com/status/http/acme_clients/<client>
$ curl https://www.example.com/status/http/metric_zones
$ curl https://www.example.com/status/http/metric_zones/<zone>/metrics
$ 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
```

查询参数

以下查询字符串参数可改变 API 的 JSON 响应：

`pretty`
: 默认情况下，API 以带缩进和换行符的格式输出 JSON 响应。要获得紧凑的单行输出，请在查询字

符串中添加 pretty=off 参数：

```
$ curl https://www.example.com/status/connections?pretty=off

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

date

默认情况下，该模块使用 ISO 8601 格式字符串表示日期；如需改用整数 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
```

defaults (PRO)

您可以使用 defaults=on 参数获取默认参数值：

```
$ 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,
        "slow_start": 0,
        "backup": false,
        "down": false,
        "sid": ""
    }
}
```

服务器状态

/status/angie

```
{
    "version": "1.12.0",
    "build_time": "2026-07-13T16:05:43.805Z",
    "address": "192.168.16.5",
    "generation": 1,
    "load_time": "2026-07-13T16:15:43.805Z"
    "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，采用日期 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；上次配置重新加载的时间，采用日期 格式；字符串值具有毫秒精度
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名，其值是文件内容的字符串表示，例如： `{ "/etc/angie/angie.conf": "server {\n  listen 80;\n  # ...\n\n}\n" }` **警告** config_files 对象仅在启用*api_config_files* 指令时才在 /status/angie/ 中可用。

/status/angie/license (PRO)

Added in version 1.11.0: PRO

```
{
    "path": "/etc/angie/license.pem",
    "status": "valid",
    "owner": "Example Corp",
    "days_left": 30,
    "since": "2024-01-01",
    "until": "2025-01-01",
    "limits": {
        "worker_processes": 16,
        "worker_connections": 65535
    }
}
```

path	字符串；许可证文件的完整路径
status	字符串；许可证状态：missing、invalid、valid、grace、expired 或 pending
owner	字符串；来自证书主题的许可证所有者
days_left	数字；许可证状态变更前的剩余天数。负值表示许可证已过期，该值为过期后的天数
since	字符串；许可证有效期开始日期
until	字符串；许可证有效期结束日期
limits	对象；当前实例的许可限制

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

/status/slabs/<zone>

使用 slab 分配 的共享内存区的使用统计信息。这里会列出所有使用 slab 分配的共享内存区，包括*limit_conn*、*limit_req* 和*HTTP 缓存* 的区，以及*HTTP* 或*stream* upstream 的共享内存区：

```
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;
proxy_cache_valid 200 10m;
```

指定的共享内存区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
free	数字；当前空闲的内存页数
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字节数）
used	数字；当前使用的指定大小内存槽数
free	数字；当前空闲的指定大小内存槽数
reqs	数字；尝试分配指定大小内存的总次数
fails	数字；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（A 和 AAAA 查询）
srv	数字；将服务解析为地址的查询数（SRV 查询）
addr	数字；将地址解析为名称的查询数（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

备注

queries 和 responses 统计 Angie 在内部发出的每个解析请求，包括从 TTL 缓存中响应的请求。sent 统计实际发送到名称服务器的数据包；两者之间的差值反映了缓存命中数。

响应代码在 RFC 1035 的 4.1.1 节中描述。

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

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。此处，指标按*$host* 聚合，每个组作为独立区域报告：

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

指定的共享内存区将收集以下统计信息：

ssl	对象；SSL 统计信息。仅在 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次数
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；未发送响应即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）的响应的非零数量
xxx	数字；其他状态代码的响应的非零数量
data	对象；数据统计信息
received	数字；从客户端接收的总字节数
sent	数字；发送到客户端的总字节数

备注

`responses` 计数器仅统计真实的响应，即实际发送给客户端的响应。处理结束但未向客户端发送响应的请求会被计入 `discarded`，而不会增加 `responses` 计数器。特别地，`499` 并不是一个响应，而是一个仅记录在访问日志中的内部服务代码，用于标记处理结束但未发送响应的请求；这类请求会反映在 `discarded` 中，而绝不会出现在 `responses` 中。

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

备注

responses 计数器仅统计真实的响应，即实际发送给客户端的响应。处理结束但未向客户端发送响应的请求会被计入 discarded，而不会增加 responses 计数器。特别地，499 并不是一个响应，而是一个仅记录在访问日志中的内部服务代码，用于标记处理结束但未发送响应的请求；这类请求会反映在 discarded 中，而绝不会出现在 responses 中。

示例：

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

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

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

`/status/http/metric_zones/<zone>`

由 `http` 上下文中的*metric_zone* 或*metric_complex_zone* 定义的自定义指标。指标通过*metric* 指令或模块变量更新。

`discarded`	数字；因区域内存不足而丢弃的指标条目数。
`metrics`	对象；按键的指标。对于单指标区域，值为数字。对于复杂区域，值为带有指标名称的对象。对于直方图模式，值为带有桶名称的对象。

如果设置了 `discard_key` 并且某些条目已过期，它们的聚合指标将在此键下公开。

示例：

```
{
    "discarded": 3,
    "metrics": {
        "example.com": {
            "count": 42,
            "max": 8
        }
        "expired": {
            "count": 10,
            "max": 3.2
        }
    }
}
```

Stream 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。此处，指标按*$server_addr* 聚合，每个组作为独立区域报告：

```
status_zone $server_addr 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; 时出现
handshaked	数字；成功 SSL 握手的总次数
reuses	数字；SSL 握手期间会话重用的总次数
timedout	数字；超时 SSL 握手的总次数
failed	数字；失败 SSL 握手的总次数
connections	对象；连接统计信息
total	数字；客户端连接的总数
processing	数字；当前正在处理的客户端连接数
discarded	数字；未创建会话即完成的客户端连接总数
passed	数字；通过 pass 指令中继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，当无法解析客户端数据时发生，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，错误网关，例如无法选择或访问上游服务器时
service_unavailable	数字；以代码 503 完成的会话数，服务不可用，例如当访问受连接数限制时
data	对象；数据统计信息
received	数字；从客户端接收的总字节数
sent	数字；发送到客户端的总字节数

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
  }
}
```

/status/stream/metric_zones/<zone>

由 stream 上下文中的*metric_zone* 或*metric_complex_zone* 定义的自定义指标。指标通过*metric* 指令或模块变量进行更新。

discarded	数字；由于区域内存不足而丢弃的指标条目数。
metrics	对象；按键划分的指标。对于单指标区域，值为数字。对于复合区域，值为带有指标名称的对象。对于直方图模式，值为带有桶名称的对象。

如果设置了 discard_key 并且某些条目已过期，则它们的聚合指标将在此键下公开。

示例：

```
{
    "discarded": 3,
    "metrics": {
        "127.0.0.1": {
            "count": 42,
            "max": 8
        },
        "expired": {
            "count": 10,
            "max": 3.2
        }
    }
}
```

HTTP 缓存

```
proxy_cache cache_zone;
proxy_cache_valid 200 10m;
```

/status/http/caches/<cache>

对于使用*proxy_cache*配置的每个区域，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；配置的缓存最大大小限制
cold	布尔值；当 *cache loader* 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；从缓存读取的总字节数
stale	对象；从缓存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；从缓存读取的总字节数
updating	对象；在响应更新期间从缓存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；从缓存读取的总字节数
revalidated	对象；从缓存获取的已过期并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；从缓存读取的总字节数
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总字节数
responses_written	数字；写入缓存的响应总数
bytes_written	数字；写入缓存的总字节数
expired	对象；未从缓存获取的已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总字节数
responses_written	数字；写入缓存的响应总数
bytes_written	数字；写入缓存的总字节数
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总字节数
responses_written	数字；写入缓存的响应总数
bytes_written	数字；写入缓存的总字节数

在 Angie PRO 中，如果使用*proxy_cache_path* 指令启用了 *cache sharding*，各个分片将作为 shards 对象的对象成员公开：

shards	对象；将各个分片列为成员
<shard>	对象；表示单个分片，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 *cache loader* 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
        }
    }
}
```

ACME 客户端

`/status/http/acme_clients/<client>`

对于 `http` 块中配置的每个*acme_client*，返回当前客户端和证书状态：

```
{
  "state": "ready",
  "certificate": "valid",
  "details": "The client is ready to request a certificate.",
  "next_run": "2026-07-13T16:15:43.805Z"
}
```

`state`	字符串；ACME 客户端状态。可能的值：`ready`、`requesting`、`disabled`、`failed`。
`certificate`	字符串；证书状态。可能的值：`valid`、`expired`、`missing`、`mismatch`、`error`。
`details`	字符串；上次 ACME 操作的简短状态详情。
`next_run`	日期；下次计划请求或续订证书的尝试时间。当 `state` 为 `disabled` 或 `requesting` 时不返回。

limit_conn

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

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

每个配置的*http* 中的 *limit_conn* 或*stream* 中的 *limit_conn* 上下文的对象，包含以下字段：

```
{
  "passed": 73,
  "skipped": 0,
```

```
  "rejected": 0,
  "exhausted": 0
}
```

passed	数字；通过的连接总数
skipped	数字；使用零长度键或键超过 255 字节而通过的连接总数
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>

每个配置的*limit_req* 的对象，包含以下字段：

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

passed	数字；通过的请求总数
skipped	数字；使用零长度键或键超过 255 字节而通过的请求总数
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拒绝的请求总数

HTTP 上游

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;
}
```

该共享内存区的内存使用情况可在*/status/slabs/* API 部分中按区名查看。

`/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；包含上游节点的指标作为子对象，子对象的名称是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；在*server* 指令中指定的服务名称（如果已配置）
backup	布尔值；备份服务器为 true
weight	数字；配置的权重
state	字符串；节点的当前状态以及发送给它的请求： • busy：表示发送到服务器的请求数量已达到*max_conns* 设置的限制，不再向其发送新请求； • down：手动禁用，不发送请求； • draining：类似于 down，但来自先前绑定会话的请求（通过*sticky*）仍会发送； • recovering：根据*slow_start* 从故障中恢复，随着时间推移发送越来越多的请求； • unavailable：达到*max_fails* 限制，仅在*fail_timeout* 定义的间隔内发送试探性客户端请求； • up：正常运行，正常发送请求； Angie PRO 中的附加状态： • checking：配置为 essential 并正在检查中，仅发送探测请求； • unhealthy：不可操作，仅发送探测请求。
selected	对象；节点选择统计信息
current	数字；当前到节点的连接数
total	数字；转发到节点的请求总数
last	字符串或数字；节点最后被选择的时间，格式为日期
max_conns	数字；配置的到节点的同时活动连接的最大 数量（如果指定）
responses	对象；响应统计信息
<code>	数字；状态码为 <code> (100-599) 的响应的非零数量
xxx	数字；其他状态码响应的非零数量
data	对象；数据统计信息
received	数字；从节点接收的字节总数
sent	数字；发送到节点的字节总数
health	对象；健康统计信息
fails	数字；与节点通信失败的尝试总数
unavailable	数字；由于达到*max_fails* 限制而导致节点变为 unavailable 的次数
downtime	数字；节点 unavailable 无法选择的总时间（以毫秒为单位）
downstart	字符串或数字；节点变为 unavailable 的时间，格式为日期。仅当节点处于 unavailable 状态时才包含此字段；其他状态下不存在该字段
header_time (PRO 1.3.0+)	数字；从服务器接收响应头的平均时间（以毫秒为单位）；参见*response_time_factor*
response_time (PRO 1.3.0+)	数字；从服务器接收完整响应的平均时间（以毫秒为单位）；参见*response_time_factor*
sid	字符串；上游组中服务器的配置 *id*
feedback (PRO)	数字；由*feedback* 负载均衡方法当前使用的平均反馈值，仅在使用此方法时存在
keepalive	数字；当前缓存连接数
backup_switch	对象；包含活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；当前用于负载均衡请求的活动组级别。如果活动组是主组，则值为 0

health/probes (PRO)

如果上游配置了*upstream_probe (PRO)* 探测，health 对象还有一个 probes 子对象用于存储服务器的健康探测计数器，而 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": "2026-07-13T09:56:07Z"
            }
        }
    }
}
```

state 的 checking 值不计入 downtime，表示配置了 essential 探测的服务器尚未被检查；unhealthy 值表示服务器出现故障。这两种状态都意味着服务器不包含在负载均衡中。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；此服务器的探测总数
fails	数字；失败探测总数
last	字符串或数字；最后探测时间，格式为日期

queue (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求和：

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；因客户端过早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

Stream 上游

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;
}
```

该共享内存区的内存使用情况可在*/status/slabs/* API 部分中按区名查看。

/status/stream/upstreams/<upstream>

其中 *<upstream>* 是使用*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；包含上游服务器组中各对等节点的指标，作为子对象，其名称是对等节点地址的规范表示形式。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；在*server* 指令中指定的服务名称（如果已配置）
backup	布尔值；备份服务器为 true
weight	数字；为对等节点设置的权重
state	字符串；对等节点的当前状态以及发送给它的请求： • busy：表示发送到服务器的请求数已达到*max_conns* 设置的限制，不再向其发送新请求 • down：手动禁用，不发送请求 • draining：类似于 down，但来自先前绑定会话的请求（通过*sticky*）仍会发送 • recovering：根据*slow_start* 从故障中恢复，随着时间推移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 中的附加状态： • checking：配置为 essential 并正在检查中，仅发送探测请求 • unhealthy：不可用，仅发送探测请求
selected	对象；选择此对等节点进行连接的统计信息
current	数字；当前到对等节点的连接数
total	数字；转发到对等节点的连接总数
last	字符串或数字；上次选择对等节点的时间，格式为日期
max_conns	数字；到对等节点的同时活动连接的最大 数量（如果已设置）
data	对象；数据传输统计信息
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；对等节点健康统计信息
fails	数字；尝试到达对等节点失败的总次数
unavailable	数字；由于达到*max_fails* 值，对等节点变为 unavailable 的总次数
downtime	数字；对等节点处于 unavailable 状态（无法被选择）的总时间（以毫秒为单位）
downstart	字符串或数字；对等节点上次变为 unavailable 的时间，格式为日期。仅当对等节点处于 unavailable 状态时才包含此字段；其他状态下不存在该字段
connect_time	数字；与服务器建立连接的平均时间（以毫秒为单位）；参见*response_time_factor* 指令
first_byte_time	数字；从服务器接收第一个字节的平均时间（以毫秒为单位）；参见*response_time_factor* 指令
last_byte_time	数字；从服务器接收完整响应的平均时间（以毫秒为单位）；参见*response_time_factor* 指令
feedback (PRO)	数字；由*feedback* 负载均衡方法当前使用的平均反馈值，仅在使用此方法时存在
backup_switch (PRO 1.10.0+)	对象；包含活动备份逻辑的当前状态，如果为上游配置了*backup_switch (PRO)* 则显示
active	数字；当前用于负载均衡的活动组的级别。如果活动组是主组，则值为 0
timeout	数字；剩余等待时间（以毫秒为单位），之后负载均衡器将重新检查较低级

在 Angie PRO 中，如果上游配置了*upstream_probe (PRO)* 探测，`health` 对象还包含一个 `probes` 子对象，用于存储服务器的健康探测计数器，而 `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": 2,
                "fails": 2,
                "last": "2026-07-13T11:03:54Z"
            }
        }
    }
}
```

`state` 的 `checking` 值表示配置了 `essential` 参数探测的服务器尚未被检查；`unhealthy` 值表示服务器不可用。这两种状态还意味着服务器不包含在负载均衡中。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；此服务器的探测总数
`fails`	数字；失败的探测数
`last`	字符串或数字；上次探测的时间，格式为日期

证书

`/certificates/`

API 的一个顶级对象，独立于*/status/* 统计信息树，用于报告 Angie 已加载的 TLS 证书。

如果证书或密钥文件名包含变量（包括由 ACME 客户端通过*$acme_cert_<name>* 变量提供的变量），则该证书不会列在 `static` 下。由多个服务器共享的证书只列出一次。

以下示例加载一个静态证书，并通过 ACME 客户端获取另一个证书。ACME 客户端需要配置*resolver* 来解析 directory 主机名：

```
http {
    resolver 127.0.0.53;

    acme_client example https://acme.example.com/directory;

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

```
        ssl_certificate     example.com.crt;
        ssl_certificate_key example.com.key;

        location /certificates/ {
            api /certificates/;
        }
    }

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

        acme                example;
        ssl_certificate     $acme_cert_example;
        ssl_certificate_key $acme_cert_key_example;
    }
}
```

对于请求 `curl https://www.example.com/certificates/`，Angie 返回：

```
{
    "static": {
        "example.com.crt": {
            "key": "RSA (2048 bits)",
            "chain": [
                {
                    "subject": {
                        "common_name": "example.com",
                        "alt_names": [
                            "example.com",
                            "www.example.com"
                        ],
                        "organization": "Example, Inc."
                    },

                    "issuer": {
                        "common_name": "Example Root CA",
                        "country": "US",
                        "organization": "Example, Inc."
                    },

                    "validity": {
                        "since": "Sep 18 19:46:19 2022 GMT",
                        "until": "Jun 15 19:46:19 2025 GMT"
                    }
                }
            ]
        }
```

```
    },

    "acme_clients": {
        "example": {
            "key": "EC (prime256v1)",
            "chain": [
                {
                    "subject": {
                        "common_name": "acme.example.com"
                    },

                    "issuer": {
                        "common_name": "Example ACME CA"
                    },

                    "validity": {
                        "since": "Sep 18 19:46:19 2022 GMT",
                        "until": "Dec 17 19:46:19 2022 GMT"
                    }
                }
            ]
        }
    }
}
```


`static`	对象；使用 HTTP 中的*ssl_certificate*、Stream 中的*ssl_certificate* 或 Mail 中的*ssl_certificate* 设置的服务器证书，每个证书以其证书文件名为键。在使用 `--with-ntls` 构建的版本中，还包括使用 HTTP 中的*proxy_ssl_certificate* 或 Stream 中的*proxy_ssl_certificate* 设置的、用于向被代理服务器进行身份验证的证书
`acme_clients`	对象；当 Angie 使用*HTTP ACME 模块* 构建时可用，*Stream ACME 模块* 也需要该模块。包含*ACME 客户端* 为 HTTP 和 Stream 服务器获取的证书。每个条目以*acme_client* 名称为键。仅列出已经获取了证书的客户端

`static` 和 `acme_clients` 的每个条目都描述单个证书包：

key	字符串；私钥类型和大小（或曲线），例如 RSA (2048 bits)、EC (prime256v1) 或 DH (2048 bits)
chain	数组；证书包中的证书，从叶（服务器）证书开始，后跟任何中间证书，直至根证书。每个元素都是描述单个证书的对象：
subject	对象；主体可分辨名称
common_name	字符串；通用名称（CN）
alt_names	字符串数组；来自主体备用名称（Subject Alternative Name）扩展的 DNS 名称和 IP 地址
country	字符串；国家/地区（C）
state_or_province	字符串；州或省名称（ST）
organization	字符串；组织（O）
issuer	对象；颁发者可分辨名称，字段与 subject 相同
validity	对象；证书有效期
since	字符串；有效期的开始（notBefore）
until	字符串；有效期的结束（notAfter）

subject 和 issuer 的字段仅在证书中存在时才出现。

动态配置 API (PRO)

该 API 包含一个 /config 节，可通过 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：新设置作为一个整体应用，或者完全不应用。出错时，Angie 会报告原因。

/config 的子节

目前，/config 节中可用于*HTTP* 和*stream* 模块的上游内各个服务器的配置；可进行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节点，包括删除现有对等节点或添加新对等节点。

URI 路径参数：

<upstream>	上游的名称；要通过 /config 进行配置，必须配置*zone* 指令，定义一个共享内存区域。
<name>	上游中对等节点的名称，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 地址；可以在此定义可选的端口。

例如，以下配置：

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

允许以下对等节点名称：

```
$ 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/
```

此 API 子节允许设置 `weight`、`max_conns`、`max_fails`、`fail_timeout`、`slow_start`、`backup`、`down` 和 `sid` 参数，如*server* 中所述。

备注

这里没有单独的 `drain` 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,
    "slow_start": 0,
    "backup": true,
    "down": false,
    "sid": ""
}
```

实际可用的参数仅限于*upstream* 当前负载均衡方法所支持的参数。因此，如果上游配置了 `random` 方法：

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

您将无法添加定义了 `backup` 的新对等节点：

```
$ 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."
}
```

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

允许配置单个上游对等节点，包括删除现有对等节点或添加新对等节点。

URI 路径参数：

<upstream>	upstream 块的名称；要通过 /config 进行配置，必须配置*zone* 指令，定义一个共享内存区域。
<name>	上游中对等节点的名称，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 地址；可以在此定义可选的端口。

例如，以下配置：

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

允许以下对等节点名称：

```
$ 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 weight、max_conns、max_fails、fail_timeout、slow_start、backup 和 down 参数，如*server* 中所述。

备注

这里没有单独的 drain 参数；要启用 drain 模式，将 down 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,
    "slow_start": 0,
    "backup": true,
    "down": false,
}
```

实际可用的参数仅限于*upstream* 当前负载均衡方法所支持的参数。因此，如果上游配置了 `random` 方法：

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

您将无法添加定义了 `backup` 的新对等节点：

```
$ 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."
}
```

删除对等节点时，您可以设置 `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

让我们以下面的上游配置为例，来考虑适用于本节的每个 HTTP 方法的语义：

```
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 方法查询 /config 中任何现有路径上的实体，就像它对其他 API 部分所做的那样。

例如，/config/http/upstreams/backend/servers/ 上游服务器分支支持以下查询：

```
$ 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 参数获取默认参数值，详见查询参数 部分。

PUT

PUT HTTP 方法在指定路径创建一个新的 JSON 实体，或 * 完全 * 替换现有实体。

例如，要向 backend 上游中的 backend.example.com 服务器添加之前未指定的 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 方法删除指定路径上 * 先前定义 * 的设置；这样做时，如果存在默认值，它会恢复默认值。

例如，要删除 backend 上游中 backend.example.com 服务器之前修改的 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 参数验证更改：

```
$ 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,
    "slow_start": 0,
    "backup": false,
    "down": false,
    "sid": ""
}
```

max_fails 参数已恢复为其默认值。

删除服务器时，您可以设置 `connection_drop=<value>` 参数（PRO）来覆盖*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 方法在指定路径创建一个新实体，或部分替换或补充现有实体（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存在于配置中，则会被覆盖；否则，它们会被添加。

例如，要更改 backend 上游中 backend.example.com 服务器的 down 参数，同时保持其余部分不变：

```
$ 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
}
```

请注意，随 PATCH 请求提供的 JSON 对象 * 被合并 * 到现有对象中，而不是像 PUT 那样完全替换它。

null 值是一种特殊情况；它们用于在此类合并期间删除特定的配置项。

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 参数并同时更新 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 值的 down 参数已被删除；max_conns 值已更新。

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）。

警告

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:samp:*User-Agent* 请求头字段的值。

变量

$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；

$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；

`$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:samp:*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）、`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:samp:*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:samp:*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:samp:*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:samp:*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:samp:*X-Accel-Charset*响应头字段中设置字符集。可以使用*:ref:‘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:samp:*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。

备注

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

Added in version 1.10.0.

该模块基于 Docker 容器标签，在*HTTP* 和*stream* 上下文中提供代理服务器组的动态配置。要使该功能正常工作，必须在组中配置共享内存区域（参见*http* 和*stream* 的 `zone` 说明）。

备注

该模块支持与 Docker 及其替代品（如 Podman）配合使用，这些替代品实现了兼容的 API。建议使用 Podman 4.9.3 及以上版本。

该模块通过 API 连接到 Docker 守护进程，与其交互的方法由*docker_endpoint* 指令指定。在获取正在运行的容器列表后，Angie 会分析它们是否存在合适的标签。如果容器描述中包含带有端口的标签，则该容器的地址和端口，以及该容器其他标签中的参数，将自动添加到 Angie 配置中相应的 `upstream` 块。

备注

同一个容器可以添加到多个 `upstream` 组。要实现这一点，只需指定多组具有不同组名和端口的标签即可。

这在容器在不同端口上运行多个不同服务时特别有用——每个服务可以关联到自己的组。

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 地址将添加到指定的组；
- 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，服务器被标记为 `down`，使用 **`docker unpause`** 时——标记为 `up`。

配置示例

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

`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;
    }
}
```

收到容器事件后，Angie 会查找形式为 `angie.http.upstreams.<name>.port=<port>`（用于 HTTP 上下文）或 `angie.stream.upstreams.<name>.port=<port>`（用于 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 `angie.network` 标签，则为第一个可用的地址）将添加到相应的代理服务器组。

如果容器停止或被删除，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`。

Angie 识别的带有标签的 `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

标签在代理服务器组中指定服务器参数，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。
angie.network=<docker-network>	用于获取容器 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>	最大并发连接数（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>	计算失败尝试的时间间隔（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>	启用具有可配置时间段的 slow_start 模式。

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。

可以使用*client* 上下文进一步配置连接，该模块在其中添加了两个命名的 `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* 指令以及其他几个最佳默认设置，可以向其中添加代理 模块的其他设置。

如果指定了该指令，Angie 将使用指定的方法打开到 Docker 的连接，请求正在运行的容器列表，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（`/var/run/docker.sock` 或其他）访问 Docker 守护进程，Angie 运行所使用的用户 必须对此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` 足以容纳大约 25 个容器。

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

DoH

Added in version 1.12.0.

实现了一个由 RFC 8484 定义的 DNS over HTTPS (DoH) 服务器。接受通过 HTTP/HTTPS 承载的 DNS 查询, 并通过 UDP 或 TCP 将其代理到一组 DNS 服务器。

DoH 客户端提交 DNS 查询有两种方式: 一种是作为 `GET` 请求, 将 DNS 消息以 Base64URL 编码后放在 `dns` 查询字符串参数中; 另一种是作为 `POST` 请求, 将原始 DNS 消息放在请求体中。这两种情况下,DNS 消息的媒体类型都是 `application/dns-message`。

模块接受的带有非零 `RCODE` 的 DNS 响应会以 200 (OK) 状态返回给客户端。其中 `RCODE` 保留在 DNS 消息中, 并且不会在*doh_next_upstream* 的 `invalid_response` 条件下触发重试。

doh_transport 指令控制查询使用 UDP 还是 TCP。为了防止路径外 (off-path) 响应欺骗, 模块会随机化发送给 DNS 服务器的查询 ID, 并在响应中恢复客户端的原始 ID。

备注

这种查询 ID 随机化使用非加密的生成器, 只能提供有限的保护; 请通过妥善保护的可信网络访问 DNS 服务器, 正如*resolver* 指令所建议的那样。

响应的 `Cache-Control` 头字段由响应的最小 TTL 派生而来: 它被设置为 `max-age=<ttl>`, 或者在无法确定可缓存的 TTL 时设置为 `no-store`。

在从源代码构建时, 可以使用 `--without-http_doh_module` 构建选项禁用该模块。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

将 DNS over HTTPS 查询代理到一组 DNS 服务器:

```
http {
    upstream dns_upstream {
        server 127.0.0.1:53;
    }

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

        ssl_certificate     dns.example.com.crt;
        ssl_certificate_key dns.example.com.key;

        location /dns-query {
            doh_pass dns_upstream;
        }
    }
}
```

指令

doh_bind

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

使到 DNS 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` 取消从上一级配置继承的*doh_bind* 指令的效果, 允许系统自动分配本地 IP 地址和端口。

`transparent` 参数允许到 DNS 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:

```
doh_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 上不需要这样做, 因为当指定了 `transparent` 参数时, 工作进程会从主进程继承 `CAP_NET_RAW` 能力。

doh_buffer_size

语法	`doh_buffer_size` *size*;
默认值	`doh_buffer_size 16k;`
上下文	http, server, location

设置用于读取从 DNS 服务器接收的响应的缓冲区的 `size`。该 `size` 不能小于 512 字节。

doh_connect_timeout

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

定义与 DNS 服务器建立连接的超时时间。

doh_max_size

语法	`doh_max_size` *size*;
默认值	一个操作系统内存页的大小
上下文	http, server, location

限制从客户端接受的 DNS 查询消息的大小。配置的 `size` 必须至少为 12 字节。短于 12 字节 (DNS 消息头的大小) 或大于 `size` 的查询将被拒绝。

doh_next_upstream

语法	`doh_next_upstream error | timeout | invalid_response | off ...;`
默认值	`doh_next_upstream error timeout;`
上下文	http, server, location

指定在哪些情况下应将查询传递给*upstream* 组中的下一个服务器:

`error`	在与服务器建立连接、向其传递查询或读取响应时发生了错误;
`timeout`	在与服务器建立连接、向其传递查询或读取响应时发生了超时;
`invalid_response`	服务器返回了无效的 DNS 响应;
`off`	禁用将查询传递给下一个服务器。

将查询传递给下一个服务器可以受到尝试次数和时间的限制。

doh_next_upstream_timeout

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

限制可以将查询传递给下一个服务器的时间。设置为 `0` 将禁用此限制。

doh_next_upstream_tries

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

限制将查询传递给下一个服务器的可能尝试次数。设置为 `0` 将禁用此限制。

doh_pass

语法	`doh_pass` *address*;
默认值	—
上下文	location, if in location

设置将 DNS 查询代理到的 DNS 服务器的地址。该地址可以指定为域名或带端口的 IP 地址:

```
doh_pass 127.0.0.1:53;
```

或指定为服务器组:

```
doh_pass dns_upstream;
```

查询通过 UDP 或 TCP 转发给服务器, 具体由*doh_transport* 指令配置。在一个 location 中只能指定一个*doh_pass* 指令。

doh_read_timeout

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

定义从 DNS 服务器读取响应的超时时间。该超时只在两次连续的读操作之间设置, 而不是针对整个响应的传输。如果 DNS 服务器在此时间内没有传输任何内容, 则连接将被关闭。

doh_send_timeout

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

设置向 DNS 服务器传输请求的超时时间。该超时只在两次连续的写操作之间设置, 而不是针对整个请求的传输。如果 DNS 服务器在此时间内没有收到任何内容, 则连接将被关闭。

doh_socket_keepalive

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

为到 DNS 服务器的出站连接启用 `SO_KEEPALIVE` 套接字选项。当设置为 `off` 时, 操作系统的套接字设置保持有效。

doh_transport

语法	`doh_transport tcp | udp | auto;`
默认值	`doh_transport auto;`
上下文	http, server, location

选择用于将 DNS 查询转发给 DNS 服务器的传输协议:

auto	如果上游的*keepalive* 缓存中有空闲的 TCP 连接, 则通过它 (TCP) 发送查询; 否则先通过 UDP 发送, 如果服务器返回被截断的消息 (设置了 TC 标志), 则通过 TCP 重试;
udp	查询只通过 UDP 发送; 被截断的响应会被视为错误, 查询不会通过 TCP 重试;
tcp	查询只通过 TCP 发送。

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

使到 FastCGI 服务器的出站连接源自指定的本地 IP 地址及可选端口。参数值可以包含变量。特殊值 `off` 取消从上一配置级别继承的 *fastcgi_bind* 指令的效果，允许系统自动分配本地 IP 地址和端口。

`transparent` 参数允许到 FastCGI 服务器的出站连接源自非本地 IP 地址，例如来自客户端的真实 IP 地址：

```
fastcgi_bind $remote_addr transparent;
```

要使此参数生效，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，工作进程会从主进程继承 *CAP_NET_RAW* 能力。

备注

还应配置内核路由表以拦截来自 FastCGI 服务器的网络流量。

fastcgi_buffer_size

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

设置用于读取从 FastCGI 服务器接收的响应的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。根据平台不同，这可能是 4K 或 8K。但它可以设置得更小。

fastcgi_buffering

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

启用或禁用来自 FastCGI 服务器的响应的缓冲。

`on`	Angie 尽快从 FastCGI 服务器接收响应，将其保存到由*fastcgi_buffer_size* 和*fastcgi_buffers* 指令设置的缓冲区中。向客户端发送会并行进行：已填满的缓冲区会被传递用于发送，但其总大小受*fastcgi_busy_buffers_size* 限制。如果缓冲区未被完全填满，则不会被传递用于发送，除非它包含响应的最后一部分。因此，当需要即时传输每几个字节时，缓冲读取并不适合。如果整个响应无法放入内存，则可以将其中一部分保存到磁盘上的临时文件 中。写入临时文件由*fastcgi_max_temp_file_size* 和*fastcgi_temp_file_write_size* 指令控制。
`off`	响应在接收到时立即传递给客户端。Angie 以”读取—发送”的循环工作，不会等待缓冲区完全填满：例如从 4K 缓冲区读取到 10 字节时就会立即发送。同时，如果整个响应可以放入缓冲区,Angie 可以完整读取它。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。使用 `off` 时,:ref:*fastcgi_limit_rate* 不生效。

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

fastcgi_buffers

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

设置用于从 FastCGI 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。根据平台不同，这可能是 4K 或 8K。

fastcgi_busy_buffers_size

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

当启用来自 FastCGI 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向客户端发送响应的缓冲区的总大小。同时，其余缓冲区可用于读取响应，如果需要，还可以将部分响应缓冲到临时文件。默认情况下，大小受*fastcgi_buffer_size* 和*fastcgi_buffers* 指令设置的两个缓冲区大小的限制。

fastcgi_cache

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

定义用于缓存的共享内存区域。同一区域可以在多个地方使用。参数值可以包含变量。`off` 参数禁用从上一配置级别继承的缓存。

备注

使用*fastcgi_cache* 时，通常还需要设置*fastcgi_cache_valid* 指令来明确指定响应的缓存时间。如果未设置,Angie 不使用默认值，而是根据来自服务器的 HTTP 响应头按以下优先级顺序确定响应缓存时间:

1. `X-Accel-Expires` 头 (最高优先级)。
2. 带有 `max-age` 或 `s-maxage` 参数的 `Cache-Control` 头。
3. `Expires` 头。

如果这些头都不包含有效值或根本不存在，响应将不会被缓存，因为无法确定其过期时间。

fastcgi_cache_background_update

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

允许启动后台子请求来更新过期的缓存项，同时将陈旧的缓存响应返回给客户端。请注意，必须允许 在更新时使用陈旧的缓存响应。

fastcgi_cache_bypass

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

定义不从缓存中获取响应的条件。如果字符串参数中至少有一个值不为空且不等于”0”，则不会从缓存中获取响应:

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

参数值可以包含变量。

可以与*fastcgi_no_cache* 指令一起使用。

fastcgi_cache_key

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

定义缓存的键。键可以包含变量。例如

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

fastcgi_cache_lock

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

启用后，一次只允许一个请求通过将请求传递给 FastCGI 服务器来填充根据*fastcgi_cache_key* 指令标识的新缓存元素。同一缓存元素的其他请求将等待响应出现在缓存中或此元素的缓存锁被释放，最长等待时间由*fastcgi_cache_lock_timeout* 指令设置。

fastcgi_cache_lock_age

语法	`fastcgi_cache_lock_age` *time*;
默认值	`fastcgi_cache_lock_age 5s;`
上下文	http, server, location

如果传递给 FastCGI 服务器以填充新缓存元素的最后一个请求在指定时间内未完成，则可以再传递一个请求给 FastCGI 服务器。

fastcgi_cache_lock_timeout

语法	`fastcgi_cache_lock_timeout` *time*;
默认值	`fastcgi_cache_lock_timeout 5s;`
上下文	http, server, location

设置*fastcgi_cache_lock* 的超时时间。当超时时间到期时，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` *number*;
默认值	—
上下文	http, server, location

为字节范围请求设置以字节为单位的偏移量。如果范围超出偏移量，范围请求将被传递给 FastCGI 服务器，且响应不会被缓存。

fastcgi_cache_methods

语法	`fastcgi_cache_methods GET | HEAD | POST ...;`
默认值	`fastcgi_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" 方法始终会被添加到列表中，但建议显式指定它们。另请参阅*fastcgi_no_cache* 指令。

fastcgi_cache_min_uses

语法	`fastcgi_cache_min_uses` *number*;
默认值	`fastcgi_cache_min_uses 1;`
上下文	http, server, location

设置请求次数，达到该次数后响应将被缓存。

警告

缓存元数据存储在共享内存中。手动删除缓存文件不会重置计数器，可能导致不可预测的行为。要完全重置缓存，请停止服务器，删除缓存目录，然后重新启动。

备注

第三方缓存清除模块 (例如 external-cache-purge) 仅删除文件，但不会重置 *fastcgi_cache_min_uses* 计数器。该指令旨在保护缓存免受不频繁请求的污染，在清除时重置计数器可能会对性能产生负面影响。

fastcgi_cache_path

语法	`fastcgi_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, 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都是对代理 URL 应用 MD5 函数的结果。

`levels`	定义缓存的层次结构级别: 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, 然后重命名该文件。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, 文件将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, 建议对于任何给定位置, 将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	确定用于临时文件的目录
on	如果未指定该参数或设置为 on，将使用给定 *location* 的*fastcgi_temp_path* 指令指定的目录
off	临时文件将直接放置在缓存目录中
keys_zone	设置用于存储所有活动键和数据信息的共享内存区域的名称和大小。缓存元数据存储在共享内存中。 1 兆字节的区域足以存储约 8000 个键。 当使用 keys_zone 指定可选的 *file* 时，Angie 会在主进程终止时将该区域的内容转储到磁盘，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它，以确保更可靠的数据持久性并减少缓存加载时间。 如果由于区域大小更改、二进制版本不兼容或其他原因而无法恢复该区域，Angie 将记录警告（failed to restore zone at address）并且不会使用区域恢复机制。相反，不兼容的文件将被重命名为 .old；您可以删除它，或恢复其名称并将 Angie 还原到最初创建此文件的配置和版本。 **警告** 确保正确指定 *file* 的路径，并具有 Angie 使用所需的访问权限，并受到保护以防止未经授权的访问；相对路径基于 prefix。
inactive	如果在此参数指定的时间内未访问缓存数据，则无论其新鲜度如何，数据都将被删除；每次访问都会重置此计时器。这与下文所述的、由 ** 缓存管理器 ** 执行的基于大小的清除不同：即使缓存未达到 max_size，基于不活动的删除也会进行，并且会忽略由*fastcgi_cache_valid* 配置的有效期。 默认情况下，inactive 为 10 分钟。

特殊的 **缓存管理器**进程监控最大缓存大小以及缓存所在文件系统的最小可用空间量。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。数据以迭代方式删除。

max_size	最大缓存大小
min_free	缓存所在文件系统的最小可用空间量
manager_files	限制一次迭代期间要删除的项目数 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	配置迭代之间的暂停时间 默认值为 50 毫秒

Angie 启动一分钟后, 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。加载也以迭代方式完成。

`loader_files`	一次迭代中要加载的最大缓存项目数 默认值:100
`loader_threshold`	限制一次迭代的时间 默认值:200 毫秒
`loader_sleep`	迭代之间保持暂停的时间 默认值:50 毫秒

fastcgi_cache_revalidate

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

启用使用带有 `If-Modified-Since` 和 `If-None-Match` 头字段的条件请求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 ...;`
默认值	`fastcgi_cache_use_stale off;`
上下文	http, server, location

确定在与 FastCGI 服务器通信期间发生错误时, 在哪些情况下可以使用陈旧的缓存响应。该指令的参数与*fastcgi_next_upstream* 指令的参数匹配。

`error`	如果无法选择 FastCGI 服务器来处理请求, 则允许使用陈旧的缓存响应。
`updating`	一个附加参数, 如果缓存响应当前正在更新, 则允许使用陈旧的缓存响应。这可以在更新缓存数据时最小化对 FastCGI 服务器的访问次数。

也可以直接在响应头中允许使用陈旧的缓存响应, 在响应变陈旧后的指定秒数内。

- `Cache-Control` 头字段的 stale-while-revalidate 扩展允许在缓存响应当前正在更新时使用陈旧的缓存响应。
- `Cache-Control` 头字段的 stale-if-error 扩展允许在发生错误时使用陈旧的缓存响应。

备注

此方法的优先级低于设置指令参数。

为了在填充新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

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

为代码 200 和 302 的响应设置 10 分钟的缓存, 为代码 404 的响应设置 1 分钟的缓存。

如果只指定了缓存时间,

```
fastcgi_cache_valid 5m;
```

则只缓存 200、301 和 302 响应。

此外, 可以使用 `any` 参数指定缓存任何响应:

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

备注

缓存参数也可以直接在响应头中设置。这比使用指令设置缓存时间具有更高的优先级。

- `X-Accel-Expires` 头字段以秒为单位设置响应的缓存时间。零值将禁用响应的缓存。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), 直到该时间响应可以被缓存。
- 如果头不包含 `X-Accel-Expires` 字段, 则可以在头字段 `Expires` 或 `Cache-Control` 中设置缓存参数。
- 如果头包含 `Set-Cookie` 字段, 则此类响应将不会被缓存。
- 如果头包含具有特殊值"*" 的 `Vary` 字段, 则此类响应将不会被缓存。如果头包含具有其他值的 `Vary` 字段, 则此类响应将根据相应的请求头字段进行缓存。

可以使用*fastcgi_ignore_headers* 指令禁用对这些响应头字段中的一个或多个的处理。

fastcgi_catch_stderr

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

设置要在从 FastCGI 服务器接收的响应的错误流中搜索的字符串。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*;
默认值	`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，此超时时间通常不能超过 75 秒。

fastcgi_connection_drop

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

启用在代理服务器从组中删除或通过重新解析 进程或*API* 命令 `DELETE` 标记为永久不可用后，终止到该服务器的所有连接。

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；设置为 `on` 时，连接将立即断开。

fastcgi_force_ranges

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

无论 FastCGI 服务器响应中的 `Accept-Ranges` 字段如何，都为来自该服务器的缓存和非缓存响应启用字节范围支持。

fastcgi_hide_header

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

默认情况下,Angie 不会将 FastCGI 服务器响应中的头字段 `Status` 和 `X-Accel-...` 传递给客户端。`fastcgi_hide_header` 指令设置不会被传递的其他字段。相反，如果需要允许传递字段，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

确定当客户端在不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 FastCGI 服务器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，处理这些头字段将产生以下效果：

- `X-Accel-Expires`、`Expires`、`Cache-Control`、`Set-Cookie` 和 `Vary` 设置响应缓存的参数；
- `X-Accel-Redirect` 执行到指定 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` 设置响应所需的字符集。

fastcgi_index

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

设置将附加在以斜杠结尾的 URI 之后的文件名，该文件名将用于*$fastcgi_script_name* 变量的值。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

确定代码大于或等于 300 的 FastCGI 服务器响应应该传递给客户端，还是被拦截并重定向到 Angie 以使用*error_page* 指令进行处理。

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 服务器将在发送响应后立即关闭连接。但是，当此指令设置为值 `on` 时，Angie 将指示 FastCGI 服务器保持连接打开。这对于到 FastCGI 服务器的*keepalive* 连接正常工作尤其必要。

fastcgi_limit_rate

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

限制从代理服务器读取响应的速度。*rate* 以每秒字节数为单位指定；可以使用变量。

0	禁用速率限制

备注

该限制是针对每个请求设置的，因此如果 Angie 同时打开两个到 FastCGI 服务器的连接，总速率将是指定限制的两倍。该限制仅在启用 FastCGI 服务器响应的缓冲 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 FastCGI 服务器响应的缓冲 时，如果整个响应无法放入*fastcgi_buffer_size* 和*fastcgi_buffers* 指令设置的缓冲区，响应的一部分可以保存到临时文件中。该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*fastcgi_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。

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

指定在哪些情况下应将请求传递给下一个服务器：

`error`	与服务器建立连接、向其传递请求或读取响应头时发生错误；
`timeout`	与服务器建立连接、向其传递请求或读取响应头时发生超时；
`invalid_header`	服务器返回空响应或无效响应；
`http_500`	服务器返回代码为 500 的响应；
`http_503`	服务器返回代码为 503 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 方法（`POST`、`LOCK`、`PATCH`）的请求在已发送到上游服务器后不会传递给下一个服务器；启用此选项将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

应该注意的是，只有在尚未向客户端发送任何内容时，才可能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。

该指令还定义了什么被视为与服务器通信的失败尝试。

`error`	始终被视为失败尝试，即使它们未在指令中指定
`timeout`	
`invalid_header`	
`http_500`	仅在指令中指定时才被视为失败尝试
`http_503`	
`http_429`	
`http_403`	从不被视为失败尝试
`http_404`	

将请求传递给下一个服务器可以通过尝试次数 和时间 进行限制。

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

限制可以将请求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

定义响应不会保存到缓存的条件。如果字符串参数中至少有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;
```

参数值可以包含变量。

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。当且仅当当前级别上没有定义*fastcgi_param* 指令时，这些指令才从上一级配置继承。

备注

在 Angie 附带的标准 `fastcgi.conf` 和 `fastcgi_params` 文件中，`REQUEST_METHOD` 被设置为 `$upstream_request_method`。这确保当缓存将 `HEAD` 转换为 `GET` 时，上游请求方法反映该转换。

以下示例显示了 PHP 所需的最小设置：

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

SCRIPT_FILENAME 参数在 PHP 中用于确定脚本名称，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 使用 `--enable-force-cgi-redirect` 配置参数构建，则还应传递值为"200" 的 REDIRECT_STATUS 参数：

```
fastcgi_param REDIRECT_STATUS 200;
```

如果指令使用 `if_not_empty` 指定，则仅当参数值不为空时才会将该参数传递给服务器：

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

fastcgi_pass

语法	`fastcgi_pass` *address*;
默认值	—
上下文	location, if in location

设置 FastCGI 服务器的地址。地址可以指定为域名或 IP 地址以及端口：

```
fastcgi_pass localhost:9000;
```

或作为 UNIX 域套接字路径：

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

如果域名解析为多个地址，则所有地址都将以轮询方式使用。此外，地址可以指定为服务器组。

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，则使用解析器 确定。

备注

If `fastcgi_pass` is placed in a `location` whose prefix ends with a slash (for example, `location /name/`), and the *auto_redirect* directive is set to `default`, requests without a trailing slash will be redirected (`/name -> /name/`).

fastcgi_pass_header

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

允许将 FastCGI 服务器的原本被禁用的 头字段传递给客户端。

fastcgi_pass_request_body

语法	`fastcgi_pass_request_body on | off;`
默认值	`fastcgi_pass_request_body on;`
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	`fastcgi_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 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, 从客户端读取 整个请求体。
`off`	请求体在接收时立即发送到 FastCGI 服务器。在这种情况下, 如果 Angie 已经开始发送请求体, 则无法将请求传递到下一个服务器。

fastcgi_send_lowat

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

如果该指令设置为非零值,Angie 将尝试通过使用*kqueue* 方法的 *NOTE_LOWAT* 标志或 *SO_SNDLOWAT* 套接字选项 (使用指定的大小), 来最小化到 FastCGI 服务器的出站连接上的发送操作次数。

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。该超时仅在两次连续的写操作之间设置, 而不是用于整个请求的传输。如果 FastCGI 服务器在此时间内没有接收到任何内容, 连接将被关闭。

fastcgi_socket_keepalive

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

配置到 FastCGI 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`	为套接字启用 *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，用于捕获*$fastcgi_path_info* 变量的值。该正则表达式应该有两个捕获组：第一个成为*$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;
```

对于 `/show.php/article/0001` 请求，'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`	使用与*alias* 或*root* 指令对应的路径保存文件。
`off`	禁用保存文件

此外，可以使用带变量的字符串显式设置文件名：

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

文件的修改时间根据接收到的 `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。但是请注意，在这种情况下，文件将跨两个文件系统复制，而不是廉价的重命名操作。因此建议对于任何给定位置，保存的文件和由*fastcgi_temp_path* 指令设置的临时文件目录都放在同一个文件系统上。

此指令可用于创建静态不变文件的本地副本，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

设置新创建的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 group 或 all 访问权限, 则可以省略 user 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

当启用将 FastCGI 服务器的响应缓冲到临时文件时, 限制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;` (路径取决于 `--http-fastcgi-temp-path` 构建参数)
上下文	http, server, location

定义一个目录, 用于存储从 FastCGI 服务器接收的数据的临时文件。在指定目录下可以使用最多三级子目录层次结构。例如, 在以下配置中

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

临时文件可能如下所示:

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

另请参阅*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可用。例如,:samp:*User-Agent* 头字段作为 HTTP_USER_AGENT 参数传递。除了 HTTP 请求头字段之外，还可以使用*fastcgi_param* 指令传递任意参数。

内置变量

http_fastcgi 模块支持内置变量，可用于通过*fastcgi_param* 指令设置参数:

$fastcgi_path_info

由*fastcgi_split_path_info* 指令设置的第二个捕获组的值。此变量可用于设置 PATH_INFO 参数。

$fastcgi_script_name

请求 URI，或者如果 URI 以斜杠结尾，则为请求 URI 加上由*fastcgi_index* 指令配置的索引文件名。此变量可用于设置 SCRIPT_FILENAME 和 PATH_TRANSLATED 参数，这些参数特别用于确定 PHP 中的脚本名称。例如，对于 `/info/` 请求，使用以下指令

```
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`。

当使用*fastcgi_split_path_info* 指令时,‘$fastcgi_script_name‘ 变量等于该指令设置的第一个捕获组的值。

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

语法	geo [*$address*] *$variable* { ... }
默认值	—
上下文	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` 中发送的最后一个地址，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` 中发送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。
`volatile`	表示该变量不可缓存。

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。

备注

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_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_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_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

允许将请求传递到 gRPC 服务器。

备注

此模块需要*HTTP2* 模块。

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

使到 gRPC 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` 取消从上一配置级别继承的 *grpc_bind* 指令的效果，允许系统自动分配本地 IP 地址和端口。

`transparent` 参数允许到 gRPC 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：

```
grpc_bind $remote_addr transparent;
```

为使此参数生效，通常需要以超级用户 权限运行 Angie 工作进程。在 Linux 上不需要这样做，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。

备注

需要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 服务器接收的响应的第一部分的缓冲区大小。响应一旦接收到就会同步传递给客户端。

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

启用在代理服务器从组中移除或被重新解析 进程或*API* `DELETE` 命令标记为永久不可用后，终止到该服务器的所有连接。

当为客户端或代理服务器处理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，连接立即断开。

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 服务器响应中的头字段 `Date`、`Server` 和 `X-Accel-...` 传递给客户端。`grpc_hide_header` 指令设置不会被传递的附加字段。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

禁用对来自 gRPC 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` 和 `X-Accel-Charset`。

如果未禁用，处理这些头字段将产生以下效果:

- `X-Accel-Redirect` 执行到指定 URI 的内部重定向;
- `X-Accel-Charset` 设置响应所需的字符集。

grpc_intercept_errors

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

确定代码大于或等于 300 的 gRPC 服务器响应应该传递给客户端，还是被拦截并重定向到 Angie 使用*error_page* 指令进行处理。

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

指定在哪些情况下应将请求传递给*upstream* 组中的下一个服务器:

error	与服务器建立连接、向其传递请求或读取响应头时发生错误;
timeout	与服务器建立连接、向其传递请求或读取响应头时发生超时;
invalid_header	服务器返回空响应或无效响应;
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 方法 (POST、LOCK、PATCH) 的请求如果已发送到上游服务器，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

应该记住，只有在尚未向客户端发送任何内容时，才可能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	仅在指令中指定时才被视为不成功尝试
http_403、http_404	从不被视为不成功尝试

将请求传递给下一个服务器可以受尝试次数 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

限制可以将请求传递给下一个服务器 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

限制将请求传递到下一个 服务器的可能尝试次数。

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;
```

要使用基于 SSL 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, 则所有地址都将以轮询方式使用。此外, 地址可以指定为服务器组。

参数值可以包含变量。在这种情况下, 如果地址指定为域名, 则在所描述的服务器组中搜索该名称, 如果未找到, 则使用解析器 确定。

备注

如果在带有前缀尾部斜杠的 `location` 中指定 `grpc_pass` (例如,:samp:*location* */name/*), 并且*auto_redirect* 指令设置为 `default`, 则不带尾部斜杠的请求将被重定向 (`/name -> /name/`)。

grpc_pass_header

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

允许将原本禁用的 头字段从 gRPC 服务器传递到客户端。

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时间。超时仅在两次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时间。超时仅在两次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传递 给 gRPC 服务器的请求头。值可以包含文本、变量及其组合。当且仅当当前级别未定义 *grpc_set_header* 指令时，这些指令才从上一级配置继承。

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

配置到 gRPC 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`	为套接字启用 *SO_KEEPALIVE* 套接字选项。

grpc_ssl_certificate

语法	`grpc_ssl_certificate` *file*;
默认值	—
上下文	http, server, location

指定一个 PEM 格式的证书文件，用于向 gRPC SSL 服务器进行身份验证。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

定义一个缓存，用于存储使用变量指定的*SSL* 证书 和密钥。

该指令支持以下参数:

- `max` —设置缓存中的最大元素数。当缓存溢出时，将删除最近最少使用 (LRU) 的元素。
- `inactive` —定义元素在此时间内未被访问即会被删除的时长，即使其仍处于 `valid` 周期内；每次访问都会重置此计时器。默认为 10 秒。
- `valid` —定义缓存元素被视为有效并可重用的时间。默认为 60 秒。在此期间之后，证书将被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

指定一个 PEM 格式的密钥文件，用于向 gRPC SSL 服务器进行身份验证。

可以指定值 `engine:`*name*`:id` 来代替文件，这将从 OpenSSL 引擎 *name* 中加载具有指定 *id* 的密钥。

可以指定值 `store:scheme:id` 来代替文件，用于从 OpenSSL 提供程序注册的 URI scheme‘(如 ‘pkcs11) 中加载具有指定 *id* 的密钥。

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 服务器发送请求时启用的密码套件。密码套件以 OpenSSL 库理解的格式指定。

密码套件列表取决于安装的 OpenSSL 版本。可以使用 `openssl ciphers` 命令查看完整列表。

警告

使用 OpenSSL 时,:samp:*grpc_ssl_ciphers* 指令 不配置 TLS 1.3 的密码套件。要使用 OpenSSL 调整 TLS 1.3 密码套件，请使用*grpc_ssl_conf_command* 指令，该指令是为支持高级 SSL 配置而添加的。

- 在 LibreSSL 中,TLS 1.3 密码套件 可以使用 `grpc_ssl_ciphers` 配置。

- 在 BoringSSL 中,TLS 1.3 密码套件完全无法配置。

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 OpenSSL 配置 命令。

备注

使用 OpenSSL 1.0.2 或更高版本时支持该指令。要使用 OpenSSL 配置 TLS 1.3 密码套件，请使用 `ciphersuites` 命令。

可以在同一级别指定多个 *grpc_ssl_conf_command* 指令。当且仅当当前级别未定义 *grpc_ssl_conf_command* 指令时，这些指令才会从上一级配置继承。

警告

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

指定一个包含 PEM 格式吊销证书 (CRL) 的文件，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` `来自 grpc_pass 的主机名;‘
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，以及在与 gRPC SSL 服务器建立连接时通过 *SNI* 传递 的服务器名称。

服务器名称也可以用变量指定。

默认情况下, 使用*grpc_pass* 中的主机名。

grpc_ssl_password_file

语法	`grpc_ssl_password_file` *file*;
默认值	—
上下文	http, server, location

指定一个包含密钥 密码短语的文件, 每个密码短语单独占一行。加载密钥时会依次尝试这些密码短语。

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

为向 gRPC SSL 服务器发送的请求启用指定的协议。

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

启用或禁用在与 gRPC SSL 服务器建立连接时, 通过 服务器名称指示 TLS 扩展 (SNI,‘RFC 6066 <https://datatracker.ietf.org/doc/html/rfc6066.html>‘_) 传递由*grpc_ssl_name* 指令设置的服务器名称。

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。

grpc_ssl_trusted_certificate

语法	`grpc_ssl_trusted_certificate` *file*;
默认值	—
上下文	http, server, location

指定一个包含 PEM 格式受信任 CA 证书的文件，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:samp:*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，可以将传输数据的大小减少 2 倍或更多。

警告

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

达到的压缩比率，计算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",:samp:*Cache-Control* 设置为"no-cache"。
max	将 Expires 设置为"Thu, 31 Dec 2037 23:55:55 GMT",:samp:*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、WebP、HEIC 和 AVIF 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。

备注

此模块利用 libgd 库。建议使用库的最新可用版本。

要转换 WebP、HEIC 或 AVIF 格式的图像,:samp:*libgd* 库必须编译时启用对这些格式的支持。

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

在 1.11.0 版本发生变更.

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*; • `image_filter` convert *type*;
默认	`image_filter off;`
上下文	location

设置要对图像执行的转换类型:

`off`	在周围位置关闭模块处理。
`test`	确保响应为 JPEG、GIF、PNG、WebP、HEIC 或 AVIF 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。
`convert` *type*	将图像转换为指定的输出格式。有效值为 `jpeg`、`gif`、`png`、`webp`、`heic` 和 `avif`。参数值可以包含变量。

image_filter_avif_quality

语法	`image_filter_avif_quality` *quality* [*speed*];
默认	`image_filter_avif_quality 80 6;`
上下文	http, server, location

设置转换后的 AVIF 图像的期望质量。可选的 `speed` 参数控制编码器速度; 两个值都必须为正数。参数值可以包含变量。

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_heic_quality

语法	`image_filter_heic_quality` *quality*;
默认	`image_filter_heic_quality 80;`
上下文	http, server, location

设置转换后的 HEIC 图像的期望质量。可接受的值为正数。参数值可以包含变量。

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。

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。键可以包含变量。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。键可以包含变量。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。

日志在请求处理结束的 `location` 上下文中写入。如果在请求处理过程中发生了内部重定向, 这可能是与原始位置不同的 `location`。

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;`
默认值	`access_log logs/access.log combined;` (路径取决于构建参数 `--http-log-path`)
上下文	http, server, location, if in location, limit_except

设置日志的路径、格式和缓冲写入设置。可以在同一配置级别使用多个日志。通过在第一个参数中指定 *"syslog:"* 前缀来配置记录到*syslog*。特殊值 *off* 取消当前级别的所有 *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 `buffer` 参数指定了缓冲区大小或指定了 `gzip` 参数，则写入将被缓冲。

警告

缓冲区大小不得超过对磁盘文件的原子写入大小。对于 FreeBSD，此大小是无限的。

启用缓冲时，数据在以下情况下写入文件：

- 如果下一条日志行无法放入缓冲区；
- 如果缓冲区中的数据存在时间超过 *flush* 参数指定的时间间隔；
- 当重新打开日志文件 或终止工作进程时。

如果指定了 `gzip` 参数，缓冲区将在写入文件之前被压缩。压缩级别可以设置在 1(更快，但压缩效果较差)到 9(较慢，但压缩效果更好) 的范围内。默认情况下，使用 *64K* 字节的缓冲区大小和压缩级别 *1*。数据以原子块的形式压缩，并且可以随时使用 `zcat` 实用程序解压缩或读取日志文件。

示例：

```
access_log /path/to/log.gz combined gzip flush=5m;
```

备注

要支持日志的 gzip 压缩,Angie 必须使用 *zlib* 库构建。

文件路径中可以使用变量，但此类日志有一些限制：

- 工作进程运行所使用的凭据的用户 必须具有在包含此类日志的目录中创建文件的权限；
- 缓冲不起作用；
- 每次日志写入时打开文件，写入后立即关闭。但是，由于常用文件的描述符可以存储在缓存中，因此在*open_log_file_cache* 指令的 *valid* 参数指定的时间内进行日志轮换时，可能会继续写入旧文件。

- 对于每次日志写入，都会检查请求的根目录是否存在——如果该目录不存在，则不会创建日志。因此*root* 和*access_log* 应该在同一配置级别描述：

```
server {
    root       /spool/vhost/data/$host;
    access_log /spool/vhost/logs/$host;
    ...
```

`if` 参数启用条件日志记录。如果条件评估结果为 `"0"` 或空字符串，则不会记录请求。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`=:samp:*default* | `json` | `none`] *string* ...;
默认值	`log_format combined "...";`
上下文	http

指定日志格式。

`escape` 参数允许将变量中的字符转义设置为 `json` 或 `default`；默认使用 `default`。`none` 值禁用字符转义。

使用 `default` 时，字符”””、”\” 以及值小于 32 或大于 126 的字符将被转义为”\xXX”。如果未找到变量值，则将连字符”-” 作为值写入日志。

使用 `json` 时，所有 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”。

发送到客户端的标头行以前缀 `sent_http_` 开头，例如 `$sent_http_content_range`。

预定义格式 `combined` 始终存在于配置中：

```
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;`
默认值	`open_log_file_cache off;`
上下文	http, server, location

定义一个缓存, 用于存储使用变量指定名称的常用日志的文件描述符。参数:

`max`	设置缓存中描述符的最大数量; 当缓存溢出时, 最近最少使用 (LRU) 的描述符将被关闭。
`inactive`	设置在此时间内未访问缓存描述符后关闭它的时间。默认为 10 秒。
`min_uses`	设置在 `inactive` 参数指定的时间内文件使用的最小次数, 之后文件描述符将在缓存中保持打开状态。默认为 1。
`valid`	指定在多长时间后检查文件是否仍以相同名称存在。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。

结果值可以包含变量。

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 Angie 外部的方式预先放入 memcached。

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 服务器的出站连接源自指定的本地 IP 地址和可选端口。参数值可以包含变量。特殊值 `off` 取消从上一配置级别继承的 *memcached_bind* 指令的效果，允许系统自动分配本地 IP 地址和端口。

`transparent` 参数允许到 memcached 服务器的出站连接源自非本地 IP 地址，例如来自客户端的真实 IP 地址：

```
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 上不需要这样做，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。

备注

需要配置内核路由表以拦截来自 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 服务器接收的响应的第一部分的缓冲区大小。响应在接收后立即同步传递给客户端。

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

启用对 memcached 服务器响应中标志存在的测试, 如果设置了该标志, 则将 `Content-Encoding` 响应头字段设置为"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

指定在哪些情况下应将请求传递给上游池 中的下一个服务器:

`error`	与服务器建立连接、向其传递请求或读取响应头时发生错误;
`timeout`	与服务器建立连接、向其传递请求或读取响应头时发生超时;
`invalid_response`	服务器返回空响应或无效响应;
`not_found`	在服务器上未找到响应;
`off`	禁用将请求传递给下一个服务器。

备注

应该记住, 只有在尚未向客户端发送任何内容时, 才可能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。

该指令还定义了什么被视为与服务器通信的失败尝试。

`error`、`timeout`、`invalid_response`	始终被视为失败尝试, 即使未在指令中指定
`not_found`	从不被视为失败尝试

将请求传递给下一个服务器可以受到尝试次数 和时间 的限制。

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

限制可以将请求传递给下一个 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

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;
```

或作为 UNIX 域套接字路径:

```
memcached_pass unix:/tmp/memcached.socket;
```

如果域名解析为多个地址, 则所有地址都将以轮询方式使用。此外, 地址可以指定为服务器组。

备注

If `memcached_pass` is placed in a `location` whose prefix ends with a slash (for example, `location /name/`), and the *auto_redirect* directive is set to `default`, requests without a trailing slash will be redirected (`/name -> /name/`).

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。超时仅在两次连续的读取操作之间设置, 而不是用于整个响应的传输。如果 memcached 服务器在此时间内未传输任何内容, 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。超时仅在两次连续的写入操作之间设置, 而不是用于整个请求的传输。如果 memcached 服务器在此时间内未接收任何内容, 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	为套接字启用 *SO_KEEPALIVE* 套接字选项。

内置变量

$memcached_key

定义用于从 memcached 服务器获取响应的键。

Metric

Added in version 1.11.0.

ngx_http_metric_module 模块允许创建任意的实时计算指标。这些指标值存储在共享内存中，并在 /status/http/metric_zones/ API 分支中实时显示。支持多种数据聚合类型（计数器、直方图、移动平均等），并可按任意键进行分组。

配置示例

统计 API 请求：

```
http {
    metric_zone api_requests:1m count;

    server {
        listen 80;

        location /api/ {
            allow 127.0.0.1;
            deny all;
            api /status/;

            metric api_requests $http_user_agent on=request;
        }
    }
}
```

如果使用此配置向 /api/ 发起请求：

```
$ curl 127.0.0.1/api/ --user-agent "Firefox"
```

api_requests 指标会实时更新：

```
{
    "http": {
       "metric_zones": {
           "api_requests": {
               "discarded": 0,
```

```
            "metrics": {
                "Firefox": 1
            }
        }
    }
}
}
```

指令

metric

语法	metric *name key=value* [on=request| response| end];
默认值	—
上下文	http, server, location

计算指定共享内存区域 *name* 的指标值。

参数：

- ***key*** **—任意字符串（通常是变量），用于对值进行分组。**
  最大长度为 255 字节。如果键更长，将被截断为 255 字节并附加省略号 ...；
- ***value*** **—由所选模式处理的数字（可以是变量）。**
  如果省略，默认为 0。如果参数无法转换为数字，则默认为 1；
- on —可选参数，指定何时计算指标：
  - 如果为 on=request，在接收到请求时计算；
  - 如果为 on=response，在准备响应期间计算；
  - 如果为 on=end （默认），在发送响应后计算。

备注

在内部重定向的情况下，on=request 阶段的指标在原始 location 中计算。但是，on=response 和 on=end 指标将在新的 location 中计算。

使用示例：

```
metric requests $http_user_agent=$request_time;
```

备注

具有空键或无效 key=value 对的指标将被忽略。省略的 *value* 被视为 0：

```
metric foo $bar;  # 等同于 $bar=0
```

这对于 count 模式很有用，该模式忽略数值，只对指标被更新这一事实做出反应。

备注

请记住，变量在不同阶段进行求值。例如，无法在 on=request（接收请求时）使用 $bytes_sent（发送给客户端的字节数）。

metric_complex_zone

语法	metric_complex_zone *name:size* [expire=on| off] [discard_key=name] { ... }
默认值	—
上下文	http

定义一个 复合指标—一组具有独立模式的指标。块体中的每一行定义一个 子指标名称、一个 模式以及可选的模式 参数。

使用示例：

```
metric_complex_zone requests:1m expire=on discard_key="old" {
    # 子指标名称       模式          参数
    min_time           min;
    avg_time           average exp   factor=60;
    max_time           max;
    total              count;
}
```

在 API 树中，这样的复合指标模板如下所示：

```
{
    "discarded": 3,
    "metrics": {
        "key1": {
            "min_time": 20,
            "avg_time": 50,
            "max_time": 80,
            "total": 2
        },
        "old": {
             "min_time": 3,
             "avg_time": 40,
             "max_time": 152,
             "total": 80
```

```
        }
    }
}
```

`discarded`	数字；共享内存区域中被丢弃的指标数量
`metrics`	对象；其成员是具有设定键的复合指标。它们是包含一组具有计算值的子指标的对象

metric_zone

语法	`metric_zone` *name:size* [`expire=on`| `off`] [`discard_key=name`] *mode* [*parameters*];
默认值	—
上下文	http

创建一个指定 *size* 大小和给定 *name* 名称的共享内存区域来存储指标。该区域名称作为 `/status/http/metric_zones/` 分支中的节点。

参数：

- `expire=<on|off>` —区域满时的行为：
  - 如果为 `on`，则丢弃最旧的指标（按更新时间）以释放内存供新指标使用；
  - 如果为 `off` （默认）—丢弃新传入的指标，保留现有条目。
- `discard_key=<name>` —定义一个键为 *name* 的指标，用于累积被丢弃指标的值。默认情况下不创建此类指标。保留键不能手动更新。
- *mode* —数据处理算法（参见操作模式 部分）；
- *parameters* —所选模式的附加设置（例如，`average exp` 的 `factor`）。

使用示例：

```
metric_zone request_time:1m max;
```

在 API 树中，共享内存区域模板如下所示：

```
{
    "discarded": 0,
    "metrics": {
        "key1": 123,
        "key2": 10.5,
    }
}
```

discarded	数字；共享内存区域中被丢弃的指标数量
metrics	对象；其成员是具有定义键和计算值的指标

备注

在 1 MB 区域中，键大小为 39 字节且单一指标模式时，大约可以存储 8,000 个唯一键条目。

操作模式

可用指标操作模式列表：

- count —计数器；
- gauge —仪表（增加/减少）；
- last —最后接收到的值；
- min —最小值；
- max —最大值；
- average exp —指数移动平均（EMA）（参数 factor）；
- average mean —窗口内的平均值（参数 window 和 count）；
- histogram —跨” 桶” 的分布（阈值列表）。

count

计数器在每次指标更新时将其值增加 1。

默认值—0。

备注

任何指标更新（使用任何值）都会单调地将计数器增加 1。

示例：

```
metric_zone count:1m count;

# 作为复合指标的一部分：
#
# metric_complex_zone count:1m {
#     some_metric_name  count;
# }

server {
```

```
    listen 80;

    location /metric/ {
        metric count KEY;
    }

    location ~ ^/metric/set/(.+)$ {
        metric count KEY=$1;
    }

    location /api/ {
        api /status/http/metric_zones/count/metrics/;
    }
}
```

更新指标：

```
$ curl 127.0.0.1/metric/
$ curl 127.0.0.1/metric/set/1
$ curl 127.0.0.1/metric/set/23
$ curl 127.0.0.1/metric/set/-32
```

API 中的预期指标值：

```
{
    "KEY": 4
}
```

gauge

gauge 根据传入数字的符号增加或减少其值。正值增加计数器，负值减少计数器。值为 0 时不改变计数器。

默认值—0。

示例：

```
metric_zone gauge:1m gauge;

# 作为复杂指标的一部分：
#
# metric_complex_zone gauge:1m {
#     some_metric_name  gauge;
# }

server {
    listen 80;

    location /metric/ {
        metric gauge KEY;
```

```
    }

    location ~ ^/metric/set/(.+)$ {
        metric gauge KEY=$1;
    }

    location /api/ {
        api /status/http/metric_zones/gauge/metrics/;
    }
}
```

更新指标:

```
$ curl 127.0.0.1/metric/
```

API 中的预期指标值:

```
{
    "KEY": 0
}
```

进一步更新:

```
$ curl 127.0.0.1/metric/set/5
$ curl 127.0.0.1/metric/set/-5
$ curl 127.0.0.1/metric/set/8
```

API 中的预期指标值:

```
{
    "KEY": 8
}
```

last

存储最后接收到的值, 不进行任何聚合。如果省略 *value*, 则使用 0。

示例:

```
metric_zone last:1m last;

# 作为复杂指标的一部分:
#
# metric_complex_zone last:1m {
#     some_metric_name  last;
# }

server {
```

```
    listen 80;

    location /metric/ {
        metric last KEY;
    }

    location ~ ^/metric/set/(.+)$ {
        metric last KEY=$1;
    }

    location /api/ {
        api /status/http/metric_zones/last/metrics/;
    }
}
```

更新指标:

```
$ curl 127.0.0.1/metric/
```

API 中的预期指标值:

```
{
    "KEY": 0
}
```

进一步更新:

```
$ curl 127.0.0.1/metric/set/8000
$ curl 127.0.0.1/metric/set/37
$ curl 127.0.0.1/metric/set/-3.5
```

API 中的预期指标值:

```
{
   "KEY": -3.5
}
```

min

保存两个值中的最小值—当前存储的值和新值。

示例:

```
metric_zone min:1m min;

# 作为复杂指标的一部分:
#
# metric_complex_zone min:1m {
```

```
#     some_metric_name  min;
# }

server {
    listen 80;

    location /metric/ {
        metric min KEY;
    }

    location ~ ^/metric/set/(.+)$ {
        metric min KEY=$1;
    }

    location /api/ {
        api /status/http/metric_zones/min/metrics/;
    }
}
```

更新指标:

```
$ curl 127.0.0.1/metric/set/42.999
$ curl 127.0.0.1/metric/set/-512
$ curl 127.0.0.1/metric/set/1
$ curl 127.0.0.1/metric/
```

API 中的预期指标值:

```
{
    "KEY": -512
}
```

max

保存两个值中的最大值—当前存储的值和新值。

示例:

```
metric_zone max:1m max;

# 作为复杂指标的一部分:
#
# metric_complex_zone max:1m {
#     some_metric_name  max;
# }

server {
    listen 80;
```

```
    location /metric/ {
        metric max KEY;
    }

    location ~ ^/metric/set/(.+)$ {
        metric max KEY=$1;
    }

    location /api/ {
        api /status/http/metric_zones/max/metrics/;
    }
}
```

更新指标:

```
$ curl 127.0.0.1/metric/set/42.999
$ curl 127.0.0.1/metric/set/-512
$ curl 127.0.0.1/metric/set/1
$ curl 127.0.0.1/metric/
```

API 中的预期指标值:

```
{
    "KEY": 42.999
}
```

average exp

使用 指数平滑 算法计算平均值。

接受可选参数 `factor=<number>` —决定新值对平均值影响程度的系数。允许的整数值范围为 0 到 99。默认值为 90。

系数越高, 新值的权重越大。如果指定 90, 结果将是新值的 90% 加上先前平均值的 10%。

示例:

```
metric_zone avg_exp:1m average exp factor=60;

# 作为复杂指标的一部分:
#
# metric_complex_zone avg_exp:1m {
#     some_metric_name  average exp  factor=60;
# }

server {
    listen 80;
```

```
    location ~ ^/metric/set/(.+)$ {
        metric avg_exp KEY=$1;
    }

    location /api/ {
        api /status/http/metric_zones/avg_exp/metrics/;
    }
}
```

更新指标:

```
$ curl 127.0.0.1/metric/set/100
$ curl 127.0.0.1/metric/set/200
$ curl 127.0.0.1/metric/set/0
$ curl 127.0.0.1/metric/set/8
$ curl 127.0.0.1/metric/set/30
```

API 中的预期指标值:

```
{
    "KEY": 30.16
}
```

average mean

计算算术平均值。接受可选参数 `window=<off|time>` 和 `count=<number>`, 分别定义用于平均的时间间隔和样本大小。默认值:`window=off`(使用整个样本) 和:samp:`count=10`。

备注

例如,:samp:*window=5s* 将仅考虑最近 5 秒内的事件。`window` 参数不能为 0。`count=number` 参数控制样本大小 (缓存值), 以实现更平滑的平均值计算。

示例:

```
metric_zone avg_mean:1m average mean window=5s count=8;

# 作为复杂指标的一部分:
#
# metric_complex_zone avg_mean:1m {
#     some_metric_name  average mean  window=5s count=8;
# }

server {
    listen 80;

    location ~ ^/metric/set/(.+)$ {
```

```
        metric avg_mean KEY=$1;
    }

    location /api/ {
        api /status/http/metric_zones/avg_mean/metrics/;
    }
}
```

更新指标:

```
$ curl 127.0.0.1/metric/set/0.1
$ curl 127.0.0.1/metric/set/0.1
$ curl 127.0.0.1/metric/set/0.4
$ curl 127.0.0.1/metric/set/10
$ curl 127.0.0.1/metric/set/1
$ curl 127.0.0.1/metric/set/1
```

API 中的预期指标值:

```
{
    "KEY": 2.1
}
```

如果从最后一次更新等待 5 秒, 预期值将为:

```
{
    "KEY": 0
}
```

histogram

创建一组" 桶", 如果新值不超过桶的阈值, 则递增相关计数器。参数以数值阈值列表的形式提供。对于分析分布 (如响应时间) 很有用。

必需参数是 *numbers* —桶的阈值, 按升序列出。

备注

桶值 `inf` 或 `+Inf` 可用于捕获所有超过最高指定桶的值。

示例:

```
metric_zone hist:1m histogram 0.1 0.2 0.5 1 2 inf;

# 作为复杂指标的一部分:
#
# metric_complex_zone hist:1m {
```

```
#     some_metric_name  histogram  0.1 0.2 0.5 1 2 inf;
# }

server {
    listen 80;

    location ~ ^/metric/set/(.+)$ {
        metric histogram KEY=$1;
    }

    location /api/ {
        api /status/http/metric_zones/hist/metrics/;
    }
}
```

更新指标:

```
$ curl 127.0.0.1/metric/set/0.25
```

API 中的预期指标值:

```
{
    "KEY": {
        "0.1": 0,
        "0.2": 0,
        "0.5": 1,
        "1": 1,
        "2": 1,
        "inf": 1
    }
}
```

进一步更新:

```
$ curl 127.0.0.1/metric/set/2
```

API 中的预期指标值:

```
{
    "KEY": {
        "0.1": 0,
        "0.2": 0,
        "0.5": 1,
        "1": 1,
        "2": 2,
        "inf": 2
    }
}
```

进一步更新:

```
$ curl 127.0.0.1/metric/set/1000
```

API 中的预期指标值:

```
{
    "KEY": {
       "0.1": 0,
       "0.2": 0,
       "0.5": 1,
       "1": 1,
       "2": 2,
       "inf": 3
    }
}
```

内置变量

为每个指标创建以下变量:

- $metric_<name>
- $metric_<name>_key
- $metric_<name>_value

对于复杂指标, 会添加额外的变量:

- $metric_<name>_value_<metric>

$metric_<name>

与*metric* 指令类似，$metric_<name> 变量设置器可用于更新指标。计算发生在 Rewrite 阶段，允许从 njs 模块处理指标。

用于设置变量的值必须遵循 key=value 结构。键和值都可以由文本、变量及其组合构成。键是用于分组值的任意字符串。值是由所选模式处理的数字。如果省略，则默认为 0。如果参数无法转换为数字，则默认为 1。

使用示例:

```
http {
    metric_zone counter:1m count;

    # 此时添加了 $metric_counter 变量

    server {
        listen 80;

        location /metric/ {
```

```
            set $metric_counter $http_user_agent;  # 等同于 $http_user_agent=0
        }

        location /api/ {
            allow 127.0.0.1;
            deny all;
            api /status/http/metric_zones/counter/;
        }
    }
}
```

使用 njs 模块计算指标：

```
http {
    js_import metrics.js;

    resolver 127.0.0.53;

    metric_complex_zone requests:1m {
        min_time        min;
        max_time        max;
        total           count;
    }

    location /metric/ {
        js_content metrics.js_request;
        js_fetch_trusted_certificate /path/to/ISRG_Root_X1.pem;
    }

    location /api/ {
        allow 127.0.0.1;
        deny all;
        api /static/http/metric_zones/requests/;
    }
}
```

文件 metrics.js：

```
async function js_request(r) {
    let start_time = Date.now();

    let results = await Promise.all([ngx.fetch('https://google.com/'),
                                     ngx.fetch('https://google.ru/')]);

    // 使用 $metric_requests 变量设置器
    r.variables.metric_requests = `google={Date.now() - start_time}`;
}
```

```
export default {js_request};
```

在对 location /metric/ 发起多次请求后，值可能如下所示：

```
{
    "discarded": 0,
    "metrics": {
        "google": {
            "min_time": 70,
            "max_time": 432,
            "total": 6
        }
    }
}
```

备注

设置变量后，可以获取其值；它将等于指定的 key=value 对。

此外，存储在 $metric_<name>_key 变量中的值将更改为指定的键。

$metric_<name>_key 和 $metric_<name>_value

$metric_<name>_key 和 $metric_<name>_value 变量分别定义键和值。当设置 $metric_<name>_value 时会发生指标更新，前提是 $metric_<name>_key 中的键已经定义。

备注

对于复杂指标，$metric_<name>_value 变量中的子指标值使用 ", " 分隔符连接。

使用示例：

```
http {
    metric_zone gauge:1m gauge;

    # 此处添加了变量 $metric_gauge、$metric_gauge_key 和 $metric_gauge_value。

    metric_complex_zone complex:1m {
        hist histogram 1 2 3;
        avg  average exp;
    }

    # 此处添加了 $metric_complex、$metric_complex_key 和 $metric_complex_value。

    server {
        listen 80;
```

```
        location /gauge/ {
            set $metric_gauge_key "foo";
            set $metric_gauge_value 1;

            # 或者：set $metric_gauge foo=1;

            return 200 "Updated with '$metric_gauge'\nValue='$metric_gauge_value'\n";
        }

        location /complex/ {
            set $metric_complex_key "foo";
            set $metric_complex_value 3;

            # 或者：set $metric_complex foo=3;

            return 200 "Updated with '$metric_complex'\nValue='$metric_complex_value'\n";
        }
    }
}
```

使用此配置，对 /gauge/ 的请求产生：

```
$ curl 127.0.0.1/gauge/
Updated with 'foo=1'
Value='1'
```

对于 /complex/：

```
$ curl 127.0.0.1/complex/
Updated with 'foo=3'
Value='0 0 1, 3'
```

备注

如果将空字符串赋值给 $metric_<name>_value，该值将被识别为 0。如果字符串由无法转换为数字的字符组成，则被识别为 1。

仅在 $metric_<name>_key 和 $metric_<name>_value 都已设置后才会进行计算。

在这种情况下，存储在 $metric_<name> 中的值将等于新的 key=value 对。

$metric_<name>_key 中的值表示通过变量指定的最后一个键。

$metric_<name>_value 中的值表示为 $metric_<name>_key 中设置的键计算的最后一个值。

`$metric_<name>_value_<metric>`

对于复杂指标，可以使用 `$metric_<name>_value_<metric>` 变量获取特定子指标的值，其中 *<metric>* 是子指标的名称。

使用示例：

```
http {
    metric_complex_zone foo:1m {
        count count;
        min   min;
        avg   average exp;
    }

    # 添加 $metric_foo、$metric_foo_key、$metric_foo_value，
    # 以及 $metric_foo_value_count、$metric_foo_value_min、$metric_foo_value_avg。

    server {
        listen 80;

        location /foo/ {
            set $metric_foo_key   bar;
            set $metric_foo_value 9;

            # 或者：set $metric_foo bar=9;

            return 200 "Updated with '$metric_foo'\nValues='$metric_foo_value'\nCount='$metric_
↪foo_value_count'\n";
        }
    }
}
```

使用此配置，对 `/foo/` 的请求产生：

```
$ curl 127.0.0.1/foo/
Updated with 'bar=9'
Values='1, 9, 9'
Count='1'
```

其他示例

监控 HTTP 方法

```
metric_zone http_methods:1m count;

server {
    listen 80;

    location / {
```

```
        metric http_methods $request_method;
    }

    location /metrics/ {
        allow 127.0.0.1;
        deny all;
        api /status/http/metric_zones/http_methods/metrics/;
    }
}
```

响应：

```
{
    "GET": 65,
    "POST": 20,
    "PUT": 10,
    "DELETE": 5
}
```

上游响应时间分布

```
metric_zone upstream_time:10m expire=on histogram
    0.05 0.1 0.3 0.5 1 2 5 10 inf;

server {
    listen 80;

    location /backend/ {
        proxy_pass http://backend;
        metric upstream_time $upstream_addr=$upstream_response_time on=end;
    }

    location /metrics/ {
        allow 127.0.0.1;
        deny all;
        api /status/http/metric_zones/upstream_time/;
    }
}
```

响应：

```
{
    "discarded": 0,
    "metrics": {
        "backend1:8080": {
            "0.05": 12,
            "0.1": 28,
            "0.3": 56,
```

```
            "0.5": 78,
            "1": 92,
            "2": 97,
            "5": 99,
            "10": 100,
            "inf": 100
        }
    }
}
```

活动连接数

```
metric_zone active_connections:2m gauge;

server {
    listen 80;
    server_name site1.com;

    location / {
        # 连接时增加
        metric active_connections site1=1 on=request;

        # 结束时减少
        metric active_connections site1=-1 on=end;
    }
}

server {
    listen 80;
    server_name site2.com;

    location / {
        metric active_connections site2=1 on=request;
        metric active_connections site2=-1 on=end;
    }
}

server {
    listen 8080;

    location /connections/ {
        allow 127.0.0.1;
        deny all;
        api /status/http/metric_zones/active_connections/metrics;
    }
}
```

响应：

```
{
    "site1": 42,
    "site2": 17
}
```

Prometheus 支持

Angie 包含一个内置模块，用于以 Prometheus 格式 显示指标，该模块支持自定义指标。

作为集成示例，请考虑以下配置：

```
http {
    # 创建 "upload" 指标
    metric_complex_zone upload:1m discard_key="other" {
        stats    histogram 64 256 1024 4096 16384 +Inf;
        sum      gauge;
        count    count;
        avg_size average exp;
    }

    # 描述 "upload" 指标的 Prometheus 模板
    prometheus_template upload_metric {
        'stats{le="$1"}' $p8s_value
                         path=~^/http/metric_zones/upload/metrics/angie/stats/(.+)$
                         type=histogram;

        'stats_sum'      $p8s_value
                         path=/http/metric_zones/upload/metrics/angie/sum;
        'stats_count'    $p8s_value
                         path=/http/metric_zones/upload/metrics/angie/count;

        'avg_size'       $p8s_value
                         path=/http/metric_zones/upload/metrics/angie/avg_size;
    }

    server {
        listen 80;

        # 更新指标
        location ~ ^/upload/(.*)$ {
            api /status/http/metric_zones/upload/metrics/angie/;
            metric upload angie=$1 on=request;
        }

        # 指标抓取目标
        location /prometheus/upload_metric/ {
            prometheus upload_metric;
        }
```

```
    }
}
```

在对 /upload/... 发起多次请求后：

```
$ curl 127.0.0.1/upload/16384
$ curl 127.0.0.1/upload/64448
$ curl 127.0.0.1/upload/64
$ curl 127.0.0.1/upload/1028
$ curl 127.0.0.1/upload/1028
```

指标值将为：

```
{
    "stats": {
        "64": 1,
        "256": 1,
        "1024": 1,
        "4096": 3,
        "16384": 4,
        "+Inf": 5
    },

    "sum": 82952,
    "count": 5,
    "avg_size": 1077.9376
}
```

以 Prometheus 格式，该指标可在 /prometheus/upload_metric/ 获取：

```
# Angie Prometheus template "upload_metric"
# TYPE stats histogram
stats{le="64"} 1
stats{le="256"} 1
stats{le="1024"} 1
stats{le="4096"} 3
stats{le="16384"} 4
stats{le="+Inf"} 5
stats_sum 82952
stats_count 5
avg_size 1077.9376
```

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。可以在同一配置级别指定多个镜像。

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。

警告

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_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 文件向客户端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* 和相关指令。

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，达到该数据量后触发由*mp4_limit_rate* 设置的速率限制。

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_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 start 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

该模块允许在 Perl 中编写位置和变量处理器，以及将 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；可以使用*load_module* 指令加载。

备注

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，因此任何情况都可能发生。

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

为指定变量设置一个 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 统计信息，基于配置中定义的模板，并以 Prometheus 格式 返回从这些模板生成的指标。

警告

要收集统计信息，请在相应的上下文中使用以下指令启用共享内存区：

- *http_upstream* 或*stream_upstream* 中的 `zone` 指令；
- *status_zone* 指令；
- *resolver* 指令中的 `status_zone` 参数。

配置示例

三个用于收集服务器共享内存区请求统计信息的指标，组合到 `custom` 模板中并在 `/p8s` 路径发布：

```
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 包含一个辅助文件 `prometheus_all.conf`，其中包含一组常用指标，组合到 `all` 模板中：

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,␣
↪4 - recovering, 5 - unhealthy, 6 - checking, 7 - draining, or 8 - busy.';


'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_http_acme_clients_state{client="$1"}' $p8st_acme_cert_state
    path=~^/http/acme_clients/([^/]+)/state$
    type=gauge
    'help=The current state of an ACME client: 1 - ready, 2 - requesting, 3 - disabled, or 4 -␣
↪failed.';

'angie_http_acme_certs_state{client="$1"}' $p8st_acme_cli_state
    path=~^/http/acme_clients/([^/]+)/certificate$
    type=gauge
    'help=The current state of an ACME client certificate: 1 - valid, 2 - mismatch, 3 -␣
↪expired, 4 - missing, or 5 - error.';
```

```
'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,
↪ 4 - recovering, 7 - draining, or 8 - busy.';


'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;
}


map $p8s_value $p8st_acme_cli_state {
    volatile;
    "ready"         1;
    "requesting"    2;
    "disabled"      3;
    "failed"        4;
}


map $p8s_value $p8st_acme_cert_state {
    volatile;
    "valid"         1;
    "mismatch"      2;
    "expired"       3;
    "missing"       4;
    "error"         5;
}
```

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, 7 - draining, 8 - busy, or 9 - idle.';


'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_peers_balance_feedback{upstream="$1",peer="$2"}' $p8s_value
```

```
    path=~^/http/upstreams/([^/]+)/peers/([^/]+)/feedback$
    type=gauge
    'help=The feedback balancer indicator, average value.';

'angie_http_upstreams_peers_balance_bandwidth{upstream="$1",peer="$2"}' $p8s_value
    path=~^/http/upstreams/([^/]+)/peers/([^/]+)/bandwidth$
    type=gauge
    'help=The least_bandwidth balancer indicator, average value.';


'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_http_acme_clients_state{client="$1"}' $p8st_acme_cert_state
    path=~^/http/acme_clients/([^/]+)/state$
    type=gauge
    'help=The current state of an ACME client: 1 - ready, 2 - requesting, 3 - disabled, or 4 -␣
↪failed.';

'angie_http_acme_certs_state{client="$1"}' $p8st_acme_cli_state
    path=~^/http/acme_clients/([^/]+)/certificate$
    type=gauge
    'help=The current state of an ACME client certificate: 1 - valid, 2 - mismatch, 3 -␣
↪expired, 4 - missing, or 5 - error.';


'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, 7 - draining, 8 - busy, or 9 - idle.';


'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_peers_balance_feedback{upstream="$1",peer="$2"}' $p8s_value
    path=~^/stream/upstreams/([^/]+)/peers/([^/]+)/feedback$
    type=gauge
    'help=The feedback balancer indicator, average value.';

'angie_stream_upstreams_peers_balance_bandwidth{upstream="$1",peer="$2"}' $p8s_value
    path=~^/stream/upstreams/([^/]+)/peers/([^/]+)/bandwidth$
    type=gauge
    'help=The least_bandwidth balancer indicator, average value.';

'angie_stream_upstreams_peers_balance_packets{upstream="$1",peer="$2"}' $p8s_value
    path=~^/stream/upstreams/([^/]+)/peers/([^/]+)/packets$
    type=gauge
    'help=The least_packets balancer indicator, average value.';


'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;
    "idle"          9;
    default         0;
}

map $p8s_value $p8st_all_ups_backup {
    volatile;
    "false"         0;
    "true"          1;
    default         $p8s_value;
}


map $p8s_value $p8st_acme_cli_state {
    volatile;
    "ready"         1;
    "requesting"    2;
    "disabled"      3;
    "failed"        4;
}

map $p8s_value $p8st_acme_cert_state {
    volatile;
    "valid"         1;
    "mismatch"      2;
    "expired"       3;
    "missing"       4;
    "error"         5;
}
```

用法:

```
http {

    include prometheus_all.conf;

    # ...

    server {
```

```
        listen 80;

        location =/p8s {
            prometheus all;
        }

        # ...

    }
}
```

```
$ curl localhost/p8s

    # Angie Prometheus template "all"
    ...
```

指令

prometheus

语法	prometheus *template_name*;
默认值	—
上下文	location

为 location 上下文指定一个模板处理器，由*prometheus_template* 指令定义。当被请求时，此 location 计算并以 Prometheus 格式返回模板指标。

```
location =/p8s {
    prometheus custom;
}
```

```
$ curl localhost/p8s

    # Angie Prometheus template "custom"
    ...
```

prometheus_template

语法	prometheus_template *template_name* { ... }
默认值	—
上下文	http

定义一个由 Angie 收集和导出的指标的命名模板，供*prometheus* 指令使用。

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，每个定义具有以下结构：*<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，您还可以使用该表达式中定义的捕获组。这些变量和组在获取指标值时进行求值，指标值由 *variable* 设置。
variable	设置将被求值并作为指标值添加到响应中的变量名称。如果变量不存在或求值结果为空（""），则不添加该指标。

指标使用 *variable* 设置的值进行计算；成功求值后，指标将添加到响应中，例如：

```
'angie_time{version="$angie_version"}' $msec;
```

```
$ curl localhost/p8s

   angie_time{version="1.12.0"} 1695119820.562
```

path=match_strin	与 Angie */status* API 子树中指标的所有端点路径进行匹配，允许一次将指标的多个实例添加到响应中。

在匹配期间，路径带有前导斜杠但不带尾部斜杠，例如 /angie/generation；匹配不区分大小写。有两种匹配方法：

path=exact_match	通过逐字符比较进行检查。
path=~regular_ex	使用 PCRE 库进行检查；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，该路径处的 Angie 指标值将存储在*$p8s_value* 变量中，该变量可在指定 path= 时在 *variable* 字段中使用。

如果 *match_string* 以尾部斜杠结尾，指标值为相应列表或对象中的项目数。例如：

```
'angie_http_server_zones_count' $p8s_value
   path=/http/server_zones/;
```

对于正则表达式，可能有多个匹配路径；指标将针对 * 每个 * 匹配添加到响应中。结合捕获组，这允许获取一系列具有相同名称和不同标签的指标，例如：

```
'angie_slabs_slots_free{zone="$1",size="$2"}' $p8s_value
    path=~^/slabs/([^/]+)/slots/([^/]+)/free$;
```

此定义为配置中当前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），则不添加该指标。

备注

`path=` 参数仅在使用*API* 模块构建 Angie 时可用。

`type=type`、`help=help`	分别以 Prometheus 格式 设置指标的类型和帮助字符串，它们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，当将 Angie API */status* 部分的指标路径与*prometheus_template* 指令定义的指标的 *match_string* 参数匹配时，该变量接收其值。

`$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，您可以使用*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。

代理

允许将请求传递到另一个 (被代理的) 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;

    proxy_cache       cache_zone;
    proxy_cache_valid 200 302 10m;
    proxy_cache_valid 404      1m;
}
```

指令

proxy_bind

在 1.12.0 版本发生变更: PRO

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

使到被代理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` 取消从上一级配置继承的 *proxy_bind* 指令的效果，允许系统自动分配本地 IP 地址和端口。

默认情况下，该值在请求初始化时求值一次，即在选择被代理的服务器之前。在 Angie PRO 中，每次选择被代理的服务器之后 (包括重试) 都会进行求值，因此绑定地址可以取决于所选服务器；例如，通过 `$upstream_current_addr` 变量。如果求得的值为空，则不进行绑定。

`transparent` 参数允许到被代理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：

```
proxy_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 上不需要这样做，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。

备注

需要配置内核路由表以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 4k|8k;`
上下文	http, server, location

设置用于读取从被代理服务器接收的响应的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。根据平台不同, 这可能是 4K 或 8K。但可以设置得更小。

proxy_buffering

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

启用或禁用来自被代理服务器的响应的缓冲。

`on`	Angie 尽快从被代理服务器接收响应, 将其保存到由*proxy_buffer_size* 和*proxy_buffers* 指令设置的缓冲区中。向客户端发送会并行进行: 已填满的缓冲区会被交给发送, 但总量不超过*proxy_busy_buffers_size*。如果缓冲区未被完全填满, 则不会被送去发送, 除非这是响应的最后一部分。因此, 当需要即时传输每隔几个字节时, 缓冲读取模式并不适合。如果整个响应无法放入内存, 则可以将其一部分保存到磁盘上的临时文件 中。写入临时文件由*proxy_max_temp_file_size* 和*proxy_temp_file_write_size* 指令控制。
`off`	响应在接收时立即传递给客户端。Angie 以" 读取 —发送" 的循环工作, 不会等待缓冲区被填满: 例如从 4K 缓冲区读取到 10 字节时就会立即发送。同时, 如果整个响应可以放入缓冲区,Angie 也可能一次读完整。Angie 一次可以从服务器接收的数据的最大大小由*proxy_buffer_size* 指令设置。使用 `off` 时,Angie 不会缓存响应, 并且*proxy_limit_rate* 不生效。

也可以通过在 `X-Accel-Buffering` 响应头字段中传递" yes" 或" no" 来启用或禁用缓冲。可以使用*proxy_ignore_headers* 指令禁用此功能。

proxy_buffers

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

设置用于从被代理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。根据平台不同, 这可能是 4K 或 8K。

proxy_busy_buffers_size

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

当启用来自被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向客户端发送响应的缓冲区的总大小。同时, 其余的缓冲区可用于读取响应, 如果需要, 还可以将部分响应缓冲到临时文件中。

默认情况下, 大小受*proxy_buffer_size* 和*proxy_buffers* 指令设置的两个缓冲区大小的限制。

proxy_cache

语法	`proxy_cache` *zone* | `off` [`path=`*path*];
默认值	`proxy_cache off;`
上下文	http, server, location

定义用于缓存的共享内存区域。同一个区域可以在多个地方使用。参数值可以包含变量。

`off`	禁用从上一级配置继承的缓存。

在 Angie PRO 中, 您可以指定多个使用相同 `keys_zone` 值的*proxy_cache_path* 指令来启用 '缓存分片 '。这样做时, 您应该设置使用此 `keys_zone` 值的*proxy_cache* 指令的 `path` 参数:

`path=`*path*	该值在从后端 * 缓存 * 响应时进行评估, 预期使用变量, 包括那些包含来自响应的信息的变量。 如果响应从缓存中获取, 则不会重新评估 *path*; 因此, 缓存的响应保留其原始 *path*, 直到它从缓存中删除。

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所需的缓存路径。`Content-Type` 的示例:

```
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;
        proxy_cache_valid 200 10m;
    }
}
```

这里有两个缓存路径和一个用于区分它们的变量映射。如果 `Content-Type` 以 `text/` 开头, 将选择第一个路径, 否则选择第二个。

备注

使用*proxy_cache* 时, 通常还需要设置*proxy_cache_valid* 指令来明确指定响应的缓存时间。如果未设置,Angie 不使用默认值, 而是根据来自服务器的 HTTP 响应头按以下优先级顺序确定响应缓存时间:

1. `X-Accel-Expires` 头 (最高优先级)。
2. 带有 `max-age` 或 `s-maxage` 参数的 `Cache-Control` 头。
3. `Expires` 头。

如果这些头都不包含有效值或根本不存在, 响应将不会被缓存, 因为无法确定其过期时间。

proxy_cache_background_update

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

允许启动后台子请求来更新过期的缓存项, 同时将陈旧的缓存响应返回给客户端。

警告

在更新陈旧缓存响应时使用它必须被允许。

proxy_cache_bypass

语法	`proxy_cache_bypass ...;`
默认值	—
上下文	http, server, location

定义不从缓存中获取响应的条件。如果字符串参数中至少有一个值不为空且不等于”0”, 则不会从缓存中获取响应:

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

参数值可以包含变量。

可以与*proxy_no_cache* 指令一起使用。

proxy_cache_convert_head

语法	proxy_cache_convert_head on | off;
默认值	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。如果禁用转换,:ref: 缓存键 *<proxy_cache_key>* 必须包含*$request_method*。

proxy_cache_key

语法	proxy_cache_key *string*;
默认值	proxy_cache_key $scheme$proxy_host$request_uri;
上下文	http, server, location

定义缓存的键。键可以包含变量。例如:

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

默认情况下, 该指令的值接近于以下字符串

```
proxy_cache_key $scheme$proxy_host$uri$is_args$args;
```

proxy_cache_lock

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

启用后, 一次只允许一个请求通过向被代理服务器传递请求来填充根据*proxy_cache_key* 指令标识的新缓存元素。对于相同缓存元素的其他请求将等待响应出现在缓存中, 或者等待该元素的缓存锁被释放, 最长等待时间由*proxy_cache_lock_timeout* 指令设置。

proxy_cache_lock_age

语法	`proxy_cache_lock_age` *time*;
默认值	`proxy_cache_lock_age 5s;`
上下文	http, server, location

如果传递给被代理服务器以填充新缓存元素的最后一个请求在指定时间内未完成，则可以再向被代理服务器传递一个请求。

proxy_cache_lock_timeout

语法	`proxy_cache_lock_timeout` *time*;
默认值	`proxy_cache_lock_timeout 5s;`
上下文	http, server, location

设置*proxy_cache_lock* 的超时时间。当超时时间到期时，请求将被传递给被代理服务器，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偏移量（以字节为单位）。如果范围超出指定的偏移量，范围请求将被传递到代理服务器，并且响应将不会被缓存。

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

如果客户端请求方法在此指令中列出，则响应将被缓存。”GET” 和”HEAD” 方法始终会添加到列表中，但建议显式指定它们。另请参阅*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

设置在多少次请求后响应将被缓存。

警告

缓存元数据存储在共享内存中。手动删除缓存文件不会重置计数器，可能导致不可预测的行为。要完全重置，请停止服务器，删除缓存目录，然后重新启动。

备注

第三方缓存清除模块（例如 external-cache-purge）仅删除文件，但不会重置 *proxy_cache_min_uses* 计数器。该指令旨在保护缓存免受不频繁请求的污染，在清除时重置计数器可能会对性能产生负面影响。

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 *path* 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`	定义缓存的层次结构级别：从 1 到 3，每个级别接受值 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
```

缓存的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。但是，请注意，在这种情况下，文件将在两个文件系统之间复制，而不是执行廉价的重命名操作。因此，建议对于任何给定位置，将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	确定用于临时文件的目录
on	如果未指定该参数或设置为 on，将使用给定 *location* 的*proxy_temp_path* 指令指定的目录
off	临时文件将直接放置在缓存目录中
keys_zone	设置用于存储所有活动键和数据信息的共享内存区域的名称和大小。缓存元数据存储在共享内存中。 1 兆字节的区域足以存储约 8000 个键。 当使用 keys_zone 指定可选的 *file* 时，Angie 会在主进程终止时将该区域的内容转储到磁盘，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它，以确保更可靠的数据持久性并减少缓存加载时间。 如果由于区域大小更改、二进制版本不兼容或其他原因而无法恢复该区域，Angie 将记录警告（failed to restore zone at address）并且不会使用区域恢复机制。相反，不兼容的文件将被重命名为 .old；您可以删除它，或恢复其名称并将 Angie 还原到最初创建此文件的配置和版本。 **警告** 确保正确指定 *file* 的路径，并具有 Angie 使用所需的访问权限，并受到保护以防止未经授权的访问；相对路径基于 prefix。
inactive	如果在此参数指定的时间内未访问缓存数据，则无论其新鲜度如何，数据都将被删除；每次访问都会重置此计时器。这与下文所述的、由 ** 缓存管理器 ** 执行的基于大小的清除不同：即使缓存未达到 max_size，基于不活动的删除也会进行，并且会忽略由*proxy_cache_valid* 配置的有效期。 默认情况下，inactive 为 10 分钟。

备注

在 Angie PRO 中，可以使用相同的 keys_zone 值指定多个 *proxy_cache_path* 指令。共享内存区域大小只能在第一个指令中指定。将根据相应*proxy_cache* 指令的 path 参数在指令之间进行选择。

一个特殊的 ** 缓存管理器 ** 进程会监控最大缓存大小和缓存所在文件系统的最小可用空间，当超过最大缓存大小或可用空间不足时，会删除最近最少使用的数据。数据删除以迭代方式进行。

max_size	缓存大小的最大阈值
min_free	缓存所在文件系统可用空间的最小阈值
manager_files	一次迭代中要删除的最大缓存项数 默认值：100。
manager_threshol	限制一次迭代的持续时间 默认值：200 毫秒。
manager_sleep	配置迭代之间的暂停时间 默认值：50 毫秒。

Angie 启动一分钟后，会激活一个特殊的 ** 缓存加载器 ** 进程。它扫描文件系统中先前缓存的数据，并将这些信息加载到缓存区域中。此进程以迭代方式工作；每次迭代处理由 `loader_files` 参数指定的有限数量的项，确保不超过 `loader_threshold`，然后进行由 `loader_sleep` 定义的短暂暂停，再继续处理下一批。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`	一次迭代中要加载的最大缓存项数 默认值：`100`
`loader_threshold`	限制一次迭代的持续时间 默认值：`200` 毫秒
`loader_sleep`	配置迭代之间的暂停时间 默认值：`50` 毫秒

备注

为 `keys_zone` 参数指定 *file* 不会影响缓存加载器的操作。

proxy_cache_revalidate

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

启用使用带有 `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。

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

确定在哪些情况下可以使用过期的缓存响应。该指令的参数与*proxy_next_upstream* 指令的参数相匹配。

`error`	如果无法选择代理服务器来处理请求，则允许使用过期的缓存响应。
`updating`	附加参数，如果缓存响应当前正在更新，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代理服务器的访问次数。

也可以直接在响应头中启用使用过期的缓存响应，指定响应过期后的秒数：

- `Cache-Control` 头字段的 stale-while-revalidate 扩展允许在缓存响应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 扩展允许在发生错误时使用过期的缓存响应。

备注

这比使用指令参数的优先级更低。

要在填充新缓存元素时最小化对代理服务器的访问次数，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;
```

为代码 200 和 302 的响应设置 10 分钟的缓存时间，为代码 404 的响应设置 1 分钟的缓存时间。

如果只指定了缓存时间，

```
proxy_cache_valid 5m;
```

则只缓存 200、301 和 302 响应。

此外，可以指定 any 参数来缓存任何响应：

```
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。这比使用指令设置缓存时间具有更高的优先级。

- X-Accel-Expires 头字段以秒为单位设置响应的缓存时间。零值会禁用响应的缓存。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），直到该时间响应可以被缓存。
- 如果头中不包含 X-Accel-Expires 字段，则可以在 Expires 或 Cache-Control 头字段中设置缓存参数。
- 如果头中包含 Set-Cookie 字段，则不会缓存此类响应。
- 如果头中包含值为"*" 的 Vary 字段，则不会缓存此类响应。如果头中包含其他值的 Vary 字段，则会考虑相应的请求头字段来缓存此类响应。

可以使用*proxy_ignore_headers* 指令禁用对这些响应头字段中一个或多个的处理。

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，此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

启用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，终止到该服务器的所有连接。

当为客户端或被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；设置为 `on` 时，连接将立即断开。

proxy_cookie_domain

语法	`proxy_cookie_domain off`; `proxy_cookie_domain` *domain replacement*;
默认值	`proxy_cookie_domain off;`
上下文	http, server, location

设置应在被代理服务器响应的 `Set-Cookie` 头字段的 domain 属性中更改的文本。假设被代理服务器返回的 `Set-Cookie` 头字段带有属性"domain=localhost"。指令

```
proxy_cookie_domain localhost example.org;
```

将把此属性重写为"domain=example.org"。

domain 和 *replacement* 字符串以及 *domain* 属性中的前导点将被忽略。该值不区分大小写。

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捕获和位置捕获，*replacement* 可以引用它们：

```
proxy_cookie_domain ~\.(?P<sl_domain>[-0-9a-z]+\.[a-z]+)$ $sl_domain;
```

可以在同一级别指定多个 *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，将选择第一个。

`off` 参数取消从上一级配置继承的 *proxy_cookie_domain* 指令的效果。

proxy_cookie_flags

语法	`proxy_cookie_flags off | cookie [flag ...];`
默认值	`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` 参数删除相应的标志。

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，将选择第一个匹配的指令。在示例中，为 cookie `one` 添加 `httponly` 标志，对于所有其他 cookie 添加 `samesite=strict` 标志并删除 `secure` 标志。

`off` 参数取消从上一级配置继承的 *proxy_cookie_flags* 指令的效果。

proxy_cookie_path

语法	`proxy_cookie_path off;` `proxy_cookie_path` *path replacement*;
默认值	`proxy_cookie_path off;`
上下文	http, server, location

设置应在被代理服务器响应的 `Set-Cookie` 头字段的 path 属性中更改的文本。假设被代理服务器返回的 `Set-Cookie` 头字段带有属性"path=/two/some/uri/"。指令

```
proxy_cookie_path /two/ /;
```

将把此属性重写为"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的匹配，或以"~*" 符号开头进行不区分大小写的匹配。正则表达式可以包含命名捕获和位置捕获，*replacement* 可以引用它们：

```
proxy_cookie_path ~*^/user/([^/]+) /u/$1;
```

可以在同一级别指定多个 *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，将选择第一个匹配的指令。

`off` 参数取消从上一级配置继承的 *proxy_cookie_path* 指令的效果。

proxy_force_ranges

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

无论被代理服务器响应中的 `Accept-Ranges` 字段如何，都为来自被代理服务器的缓存和非缓存响应启用字节范围支持。

proxy_headers_hash_bucket_size

语法	`proxy_headers_hash_bucket_size` *size*;
默认值	`proxy_headers_hash_bucket_size 64;`
上下文	http, server, location

设置*proxy_hide_header* 和*proxy_set_header* 指令使用的哈希表的桶大小。哈希表设置的详细信息在单独的文档 中提供。

proxy_headers_hash_max_size

语法	`proxy_headers_hash_max_size` *size*;
默认值	`proxy_headers_hash_max_size 512;`
上下文	http, server, location

设置*proxy_hide_header* 和*proxy_set_header* 指令使用的哈希表的最大大小。哈希表设置的详细信息在单独的文档 中提供。

proxy_hide_header

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

默认情况下，Angie 不会将被代理服务器响应中的 `Date`、`Server`、`X-Pad` 和 `X-Accel-...` 头字段传递给客户端。*proxy_hide_header* 指令设置不会被传递的附加字段。相反，如果需要允许传递字段，可以使用*proxy_pass_header* 指令。

proxy_http3_hq

语法	`proxy_http3_hq on | off;`
默认值	`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 Angie 依赖的*QUIC 互操作性测试*。

警告

仅在运行明确需要此模式的专门测试时启用此模式。

proxy_http3_max_concurrent_streams

语法	`proxy_http3_max_concurrent_streams` *number*;
默认值	`proxy_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。需要启用*keepalive* 连接。

proxy_http3_max_table_capacity

语法	`proxy_http3_max_table_capacity` *number*;
默认值	`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

类似的指令*http3_max_table_capacity* 为服务器连接设置此值。为避免错误，在代理模式下启用缓存时会禁用动态表的使用。

proxy_http3_stream_buffer_size

语法	`proxy_http3_stream_buffer_size` *size*;
默认值	`proxy_http3_stream_buffer_size 64k;`
上下文	http, server

设置用于读取和写入*QUIC 流* 的缓冲区的大小。

proxy_http_version

语法	`proxy_http_version 1.0 | 1.1 | 2 | 3;`
默认值	`proxy_http_version 1.1;`
上下文	http, server, location, if in location, limit_except

设置代理的 HTTP 协议版本。建议使用 1.1 或更高版本与*keepalive 连接* 一起使用。

proxy_ignore_client_abort

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

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

proxy_ignore_headers

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

禁用对来自代理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，处理这些头字段将产生以下效果：

- X-Accel-Expires、Expires、Cache-Control、Set-Cookie 和 Vary 设置响应缓存 的参数；
- X-Accel-Redirect 执行到指定 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 设置响应所需的字符集。

proxy_intercept_errors

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

确定状态码大于或等于 300 的代理响应应该传递给客户端，还是被拦截并重定向到 Angie 以使用*error_page* 指令进行处理。

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* 以每秒字节数指定；允许使用变量。

0	禁用速率限制

备注

该限制是针对每个请求设置的，因此如果 Angie 同时打开两个到代理服务器的连接，总速率将是指定限制的两倍。该限制仅在启用来自代理服务器的响应缓冲 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缓冲 时, 如果整个响应无法放入*proxy_buffer_size* 和*proxy_buffers* 指令设置的缓冲区中, 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*proxy_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。

proxy_method

语法	`proxy_method` *method*;
默认值	—
上下文	http, server, location

指定在转发到代理服务器的请求中使用的 HTTP 方法, 而不是来自客户端请求的方法。参数值可以包含变量。

proxy_next_upstream

在 1.11.0 版本发生变更: 如果上游组中的所有服务器均不可用，或者对该指令（及其他协议对应指令）认定为错误的状态码作出响应， Angie 现在始终返回自己的错误页面， 而不是最后一台服务器的响应。

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

指定在哪些情况下应将请求传递给*upstream* 组中的下一个服务器：

error	与服务器建立连接、向其传递请求或读取响应头时发生错误；
timeout	与服务器建立连接、向其传递请求或读取响应头时发生超时；
invalid_header	服务器返回空响应或无效响应；
http_500	服务器返回状态码为 500 的响应；
http_502	服务器返回状态码为 502 的响应；
http_503	服务器返回状态码为 503 的响应；
http_504	服务器返回状态码为 504 的响应；
http_403	服务器返回状态码为 403 的响应；
http_404	服务器返回状态码为 404 的响应；
http_429	服务器返回状态码为 429 的响应；
non_idempotent	通常，如果请求已发送到上游服务器，则不会将使用 非幂等 方法（POST、LOCK、PATCH）的请求传递给下一个服务器；启用此选项将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

应该理解，只有在尚未向客户端发送任何内容时，才可能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	仅在指令中指定时才被视为不成功尝试
http_403 http_404	永远不会被视为不成功尝试

将请求传递给下一个服务器可以通过尝试次数 和时间 进行限制。

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

限制可以将请求传递给下一个服务器 的时间。

0	禁用此限制

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	禁用此限制

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。如果字符串参数中至少有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;
```

参数值可以包含变量。

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, 以及一个可选的 URI, 用于映射 location。协议可以指定为 `http` 或 `https`。地址可以指定为域名或 IP 地址, 以及一个可选的端口：

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

或者指定为 UNIX 域套接字路径, 在 `unix` 关键字之后, 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, 所有地址将以轮询方式使用。此外, 地址还可以指定为服务器组。

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

备注

proxy_pass 中不包含变量且与已定义的服务器组 不匹配的域名, 会隐式创建一个仅含单个服务器的服务器组, 等价于:

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

这些服务器组的名称在整个 http 配置中共享, 与 proxy_pass 所在的 server 块无关。由于隐式创建的 server 没有*resolve* 参数, 该名称仅在配置加载时解析一次, 所获得的地址会一直使用到下次重载为止, 即使配置了*resolver* 也是如此。

因此, 当某个 proxy_pass 的变量求值为同一名称时, 它会复用这个已有的服务器组及其加载时的地址, 而不会通过*resolver* 在每次请求时解析该名称。要在运行时重新解析该名称并跟随 DNS 变化, 请显式定义该服务器组, 并为其 server 添加*resolve* 参数。

请求 URI 按以下方式传递给服务器:

- 如果 proxy_pass 指令 ** 带有 URI** 指定, 那么当请求传递给服务器时,:ref: 规范化 *<location>* 请求 URI 中与 location 匹配的部分将被指令中指定的 URI 替换:

```
location /name/ {
    proxy_pass http://127.0.0.1/remote/;
}
```

- 如果 proxy_pass **不带 URI** 指定, 则在处理原始请求时, 请求 URI 以客户端发送的相同形式传递给服务器, 或者在处理更改后的 URI 时传递完整的规范化请求 URI:

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

在某些情况下, 无法确定要替换的请求 URI 部分:

- 当 location 使用正则表达式指定时, 以及在命名 location 内部。

在这些情况下,:samp:*proxy_pass* 应该不带 URI 指定。

- 当在代理的 location 内部使用*rewrite* 指令更改 URI 时, 并且将使用此相同配置来处理请求 (*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, 完整的更改后的请求 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, 它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。

备注

如果 proxy_pass 放置在前缀带有尾部斜杠的 location 中 (例如,:samp:*location /name/*), 并且*auto_redirect* 指令设置为 default, 则不带尾部斜杠的请求将被重定向 (/name -> /name/)。

proxy_pass_header

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

允许将代理服务器的原本禁用 的头字段传递给客户端。

proxy_pass_request_body

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

指示是否将原始请求正文传递给代理服务器。

```
location /x-accel-redirect-here/ {
    proxy_method GET;
    proxy_pass_request_body off;
    proxy_set_header Content-Length "";
```

```
    proxy_pass ...;
}
```

另请参阅*proxy_set_header* 和*proxy_pass_request_headers* 指令。

proxy_pass_request_headers

语法	proxy_pass_request_headers on | off;
默认值	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代理服务器。

```
location /x-accel-redirect-here/ {
    proxy_method GET;
    proxy_pass_request_headers off;
    proxy_pass_request_body off;

    proxy_pass ...;
}
```

另请参阅*proxy_set_header* 和*proxy_pass_request_body* 指令。

proxy_pass_trailers

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

允许将代理服务器的尾部字段传递给客户端。

HTTP/1.1 中的尾部部分需要 显式启用。

```
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*;
默认值	`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动 连接 ID 数量。

proxy_quic_gso

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

切换使用 通用分段卸载 以*QUIC* 优化批处理模式发送数据。

proxy_quic_host_key

语法	`proxy_quic_host_key` *file*;
默认值	—
上下文	http, server

设置一个包含密钥的 *file*, 该密钥与*QUIC* 一起使用来加密 无状态重置 和 地址验证 令牌。默认情况下, 每次重启时都会生成一个随机密钥。使用旧密钥生成的令牌不被接受。

proxy_read_timeout

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

定义从代理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, 而不是用于整个响应的传输。如果代理服务器在此时间内未传输任何内容, 则连接将关闭。

proxy_redirect

语法	proxy_redirect default; proxy_redirect off; proxy_redirect *redirect replacement*;
默认值	proxy_redirect default;
上下文	http, server, location

设置应在代理服务器响应的”Location” 和”Refresh” 头字段中更改的文本。

假设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/;
```

将把此字符串重写为:

```
Location: http://frontend/one/some/uri/
```

replacement 字符串中可以省略服务器名称:

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

然后将插入主服务器的名称和端口 (如果不是 80)。

由 `default` 参数指定的默认替换使用*location* 和*proxy_pass* 指令的参数。因此, 以下两个配置是等效的:

```
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/;
}
```

警告

如果*proxy_pass* 使用变量指定, 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”~” 符号开头表示区分大小写的匹配,或者以”~*” 符号开头表示不区分大小写的匹配。正则表达式可以包含命名捕获和位置捕获,‘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/ /;
```

如果多个指令可以应用于代理服务器响应的头字段, 将选择第一个匹配的指令。

`off` 参数取消从上一级配置继承的 *proxy_redirect* 指令的效果。

使用此指令, 还可以向代理服务器发出的相对重定向添加主机名:

```
proxy_redirect / /;
```

proxy_request_buffering

语法	`proxy_request_buffering on | off;`
默认值	`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`	在将请求发送到代理服务器之前, 从客户端读取 整个请求体。
`off`	请求体在接收时立即发送到代理服务器。在这种情况下, 如果 Angie 已经开始发送请求体, 则无法将请求传递到下一个服务器。

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, 除非为代理启用 HTTP/1.1、HTTP/2 或 HTTP/3。

proxy_send_lowat

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

如果该指令设置为非零值,Angie 将尝试通过使用*kqueue* 方法的 *NOTE_LOWAT* 标志或 *SO_SNDLOWAT* 套接字选项, 以指定的大小最小化到代理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。

proxy_send_timeout

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

设置向代理服务器传输请求的超时时间。超时仅在两次连续的写操作之间设置, 而不是用于整个请求的传输。如果代理服务器在此时间内没有接收到任何内容, 连接将被关闭。

proxy_set_body

语法	`proxy_set_body` *value*;
默认值	—
上下文	http, server, location

允许重新定义传递给代理服务器的请求体。该值可以包含文本、变量及其组合。

proxy_set_header

语法	`proxy_set_header` *field value*;
默认值	`proxy_set_header Host $proxy_host;`
上下文	http, server, location

允许重新定义或追加字段到传递 给代理服务器的请求头。*value* 可以包含文本、变量及其组合。当且仅当当前级别没有定义 *proxy_set_header* 指令时, 这些指令才从上一级配置继承。默认情况下, 只重新定义两个字段:

```
proxy_set_header Host       $proxy_host;
proxy_set_header Connection close;
```

如果启用了缓存，原始请求中的头字段 If-Modified-Since、If-Unmodified-Since、If-None-Match、If-Match、Range 和 If-Range 不会传递给代理服务器。

可以像这样传递未更改的"Host" 请求头字段:

```
proxy_set_header Host       $http_host;
```

但是，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 - 它的值等于"Host" 请求头字段中的服务器名称，或者如果此字段不存在则为主服务器名称:

```
proxy_set_header Host       $host;
```

此外，服务器名称可以与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，则此字段不会传递给代理服务器:

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

proxy_socket_keepalive

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

配置到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	为套接字打开 *SO_KEEPALIVE* 套接字选项。

proxy_ssl_certificate

语法	proxy_ssl_certificate *file* [*file*];
默认值	—
上下文	http, server, location

指定一个包含 PEM 格式证书的文件，用于向代理 HTTPS 服务器进行身份验证。可以在文件名中使用变量。

当启用*proxy_ssl_ntls* 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

定义一个缓存, 用于存储使用变量指定的*SSL* 证书 和私钥。

该指令支持以下参数:

- `max` —设置缓存中的最大元素数量。当缓存溢出时, 将删除最近最少使用 (LRU) 的元素。
- `inactive` —定义元素在此时间内未被访问即会被删除的时长，即使其仍处于 `valid` 周期内；每次访问都会重置此计时器。默认为 10 秒。
- `valid` —定义缓存元素被视为有效并可以重用的时间。默认为 60 秒。在此期间之后, 证书将被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*];
默认值	—
上下文	http, server, location

指定一个包含 PEM 格式私钥的文件, 用于向代理的 HTTPS 服务器进行身份验证。

可以指定值 `engine:`name`:id` 来代替文件, 这将从名为 name 的 OpenSSL 引擎中加载具有指定 id 的私钥。

可以指定值 `store:scheme:id` 来代替文件, 用于通过已在 OpenSSL provider 中注册的 URI scheme 加载具有指定 id 的私钥, 例如 pkcs11。

文件名中可以使用变量。

当启用*proxy_ssl_ntls* 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*;
默认值	`proxy_ssl_ciphers DEFAULT;`
上下文	http, server, location

指定向代理的 HTTPS 服务器发送请求时启用的加密套件。加密套件以 OpenSSL 库能够理解的格式指定。

加密套件列表取决于安装的 OpenSSL 版本。可以使用 `openssl ciphers` 命令查看完整列表。

警告

当使用 OpenSSL 时,:samp:*proxy_ssl_ciphers* 指令 * 不 * 配置 TLS 1.3 的加密套件。要使用 OpenSSL 调整 TLS 1.3 加密套件, 请使用*proxy_ssl_conf_command* 指令, 该指令是为支持高级 SSL 配置而添加的。

- 在 LibreSSL 中,TLS 1.3 加密套件 * 可以 * 使用 `proxy_ssl_ciphers` 配置。
- 在 BoringSSL 中,TLS 1.3 加密套件完全无法配置。

proxy_ssl_conf_command

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

在与代理的 HTTPS 服务器建立连接时设置任意 OpenSSL 配置 命令。

备注

当使用 OpenSSL 1.0.2 或更高版本时支持该指令。要使用 OpenSSL 配置 TLS 1.3 加密套件，请使用 `ciphersuites` 命令。

可以在同一级别指定多个 *proxy_ssl_conf_command* 指令。当且仅当当前级别没有定义 *proxy_ssl_conf_command* 指令时，这些指令才会从上一级配置继承。

警告

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	http, server, location

指定一个包含 PEM 格式吊销证书 (CRL) 的文件，用于验证 代理的 HTTPS 服务器的证书。

proxy_ssl_keylog_file

Added in version 1.12.0.

语法	`proxy_ssl_keylog_file` *file*;
默认值	—
上下文	http, server, location

指定一个文件，用于以 SSLKEYLOGFILE 格式记录该上下文中到被代理的 HTTPS 服务器的连接的 SSL/TLS 密钥，该格式被 Wireshark 等工具用于解密捕获的 TLS 流量。

该指令不应与 LibreSSL 一起使用。

警告

该文件包含敏感的 TLS 密钥材料，可用于解密相应连接所捕获的流量。仅将此指令用于调试或协议分析，限制对该文件的访问，并在不再需要时将其删除。

proxy_ssl_name

语法	`proxy_ssl_name` *name*;
默认值	`proxy_ssl_name $proxy_host;`
上下文	http, server, location

允许覆盖用于验证 代理的 HTTPS 服务器证书的服务器名称，以及在与代理的 HTTPS 服务器建立连接时通过 *SNI* 传递 的服务器名称。

服务器名称也可以用变量指定。

默认情况下，使用*proxy_pass* URL 的主机部分。

proxy_ssl_ntls

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

使用 TongSuo TLS 库启用客户端对 NTLS 的支持。

```
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;
}
```

备注

Angie 必须使用 `--with-ntls` 配置参数以及相应的支持 NTLS 的 SSL 库进行构建

```
./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	http, server, location

指定一个包含私钥 密码短语的文件，每个密码短语单独占一行。加载密钥时会依次尝试这些密码短语。

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

启用向代理的 HTTPS 服务器发送请求时使用的指定协议。

proxy_ssl_server_name

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

启用或禁用在与代理的 HTTPS 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* 指令设置的服务器名称。

proxy_ssl_session_reuse

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

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*;
默认值	—
上下文	http, server, location

指定一个包含 PEM 格式受信任 CA 证书的文件，用于验证 代理的 HTTPS 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	http, server, location

启用或禁用对代理的 HTTPS 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	http, server, location

设置代理的 HTTPS 服务器证书链中的验证深度。

proxy_store

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

启用将文件保存到磁盘。

`on`	根据*alias* 或*root* 指令中指定的路径保存文件
`off`	禁用文件保存

文件名可以使用带变量的 `string` 显式设置：

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

文件的修改时间根据响应中接收到的 `Last-Modified` 头字段设置。响应首先写入临时文件，然后重命名该文件。临时文件和响应的持久存储可以位于不同的文件系统上。但是，请注意，在这种情况下，文件将从一个文件系统复制到另一个文件系统，而不是在一个文件系统内进行廉价的重命名操作。因此建议对于任何给定的位置，保存的文件和由*proxy_temp_path* 指令设置的临时文件目录都放在同一个文件系统上。

此指令可用于创建静态不可变文件的本地副本，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` *users:permissions ...*;
默认值	`proxy_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：

```
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 `user` 权限：

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

proxy_temp_file_write_size

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

当启用将代理服务器的响应缓冲到临时文件时，限制一次写入临时文件的数据大小。默认情况下，大小受*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*]]]‘;
默认值	`proxy_temp_path proxy_temp;` (路径取决于 构建选项 `--http-proxy-temp-path`)
上下文	http, server, location

定义用于存储从代理服务器接收的数据的临时文件的目录。在指定目录下可以使用最多三级子目录层次结构。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
```

另请参阅*proxy_cache_path* 指令的 `use_temp_path` 参数。

内置变量

http_proxy 模块支持内置变量，可用于使用*proxy_set_header* 指令组合头部:

$proxy_add_x_forwarded_for

客户端请求头字段 `X-Forwarded-For` 附加*$remote_addr* 变量后的值，以逗号分隔。如果客户端请求头中不存在 `X-Forwarded-For` 字段，则*$proxy_add_x_forwarded_for* 变量等于*$remote_addr* 变量。

$proxy_host

proxy_pass 指令中指定的被代理服务器的名称和端口;

$proxy_port

proxy_pass 指令中指定的被代理服务器的端口，或协议的默认端口;

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

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。

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。

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 `Referer` 头字段值无效的请求访问站点。需要注意的是，伪造一个带有适当 `Referer` 字段值的请求是相当容易的，因此该模块的预期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。还应考虑到，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

设置有效引用来源哈希表的桶大小。设置哈希表的详细信息在单独的文档 中提供。

referer_hash_max_size

语法	`referer_hash_max_size` *size*;
默认值	`referer_hash_max_size 2048;`
上下文	server, location

设置有效引用来源哈希表的最大大小。设置哈希表的详细信息在单独的文档 中提供。

valid_referers

语法	`valid_referers none` | `blocked` | `server_names` | *string* ...;
默认值	—
上下文	server, location

指定将导致内置变量*$invalid_referer* 被设置为空字符串的 `Referer` 请求头字段值。否则，该变量将被设置为”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；此类值是不以 http:// 或 https:// 开头的字符串；
server_names	Referer 请求头字段包含其中一个服务器名称；
任意字符串	定义服务器名称和可选的 URI 前缀。服务器名称的开头或结尾可以有一个”*”。在检查期间，Referer 字段中的服务器端口将被忽略；
正则表达式	第一个符号应该是”~”。需要注意的是，表达式将与 http:// 或 https:// 之后开始的文本进行匹配。

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，以及有条件地选择配置。

break、*goto*、*if*、*return*、*rewrite* 和*set* 指令按以下顺序处理：

- 在 *server* 级别指定的该模块指令按顺序执行；
- 重复执行：
  - 根据请求 URI 搜索 *location*；
  - 在找到的 location 内指定的该模块指令按顺序执行；
  - 如果请求 URI 被重写，则重复循环，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* 模块指令集。

如果指令在 *location* 内指定，则请求的进一步处理将在此 *location* 中继续。

示例:

```
if ($slow) {
    limit_rate 10k;
    break;
}
```

goto

Added in version 1.12.0.

语法	goto *@name*;
默认值	—
上下文	server、location、if

停止处理当前的 `http_rewrite` 模块指令集, 并向 `name` 参数给定的命名 *location*执行无条件内部重定向。与*rewrite* 不同, 它不会更改*$uri*。该参数必须是以 `@` 为前缀的名称; 否则, 配置将被拒绝。

向命名 location 的重定向会计入与重写请求 URI 时相同的 10 次 URI 更改上限。

示例:

```
location / {
    if ($http_x_maintenance) {
        goto @maintenance;
    }

    proxy_pass http://backend;
}

location @maintenance {
    return 503 "Service temporarily unavailable";
}
```

if

语法	if (*condition*) { ... }
默认值	—
上下文	server、location

对指定的条件进行求值。如果为真, 则执行大括号内指定的该模块指令, 并将 *if* 指令内的配置分配给请求。*if* 指令内的配置从上一级配置继承。

警告

虽然可以在 *if* 块内使用其他模块的指令, 但不建议这样做, 因为这可能导致意外行为。

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, 这些捕获可在 $1..$9 变量中供后续重用。还可以使用否定运算符”!~” 和”!~*”。如果正则表达式包含”}” 或”;” 字符, 则整个表达式应该用单引号或双引号括起来。
- 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

停止处理并向客户端返回指定的 `code`。非标准代码 444 会在不发送响应头的情况下关闭连接。

可以指定重定向 URL(用于代码 301、302、303、307 和 308) 或响应正文文本 (用于其他代码)。响应正文文本和重定向 URL 可以包含变量。作为特殊情况, 重定向 URL 可以指定为此服务器的本地 URI, 在这种情况下, 完整的重定向 URL 将根据请求方案 (*$scheme*) 以及*server_name_in_redirect* 和*port_in_redirect* 指令形成。

此外，代码为 302 的临时重定向 URL 可以指定为唯一参数。此类参数应以 `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与请求 URI 匹配, 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出现的顺序依次执行。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, 则处理停止并将重定向返回给客户端。

`replacement` 字符串可以包含变量，包括位置捕获 `$1` 到 `$9`。

可选的 `flag` 参数可以是以下之一:

`last`	停止处理当前的 *http_rewrite* 模块指令集, 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* 模块指令集, 与 *break* 指令相同;
`redirect`	返回代码为 302 的临时重定向; 当 `replacement` 字符串不以 `http://`、`https://` 或"*$scheme*" 开头时使用;
`permanent`	返回代码为 301 的永久重定向。

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;
#    ...
}
```

但如果这些指令放在"/download/" location 内, 则应将 `last` 标志替换为 `break`, 否则 Angie 将进行 10 次循环并返回 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它们之后。如果不希望这样, 在替换字符串末尾放置一个问号可以避免附加它们, 例如:

```
rewrite ^/users/(.*)$ /show?user=$1? last;
```

如果正则表达式包含"}" 或";" 字符, 则整个表达式应该用单引号或双引号括起来。

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* 模块指令处理结果以 *notice* 级别记录到*error_log* 中。

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

控制是否记录有关未初始化变量的警告。

内部实现

http_rewrite 模块指令在配置阶段被编译为内部指令, 这些指令在请求处理期间被解释执行。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;
}
```

将被转换为这些指令:

```
variable $forbidden
check for zero
    return 403
    end of code
variable $slow
check for zero
match of regular expression
copy "/"
copy $1
copy "/mp3/"
copy $2
copy ".mp3"
end of regular expression
end of code
```

请注意, 没有*limit_rate* 指令的指令, 因为它与 `http_rewrite` 模块无关。为 `if` 块创建了单独的配置。如果条件为真, 请求将获得此配置, 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移到括号内，可以减少一条指令：

```
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`;
默认值	—
上下文	http, server, location

使到 SCGI 服务器的出站连接源自指定的本地 IP 地址和可选端口。参数值可以包含变量。特殊值 `off` 取消从上一配置级别继承的 *scgi_bind* 指令的效果，允许系统自动分配本地 IP 地址和端口。

`transparent` 参数允许到 SCGI 服务器的出站连接源自非本地 IP 地址，例如来自客户端的真实 IP 地址：

```
scgi_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 上不需要这样做，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。

备注

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*;
默认值	`scgi_buffer_size 4k|8k;`
上下文	http, server, location

设置用于读取从 SCGI 服务器接收的响应的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。根据平台不同，这可能是 4K 或 8K。但可以设置得更小。

scgi_buffering

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

启用或禁用来自 SCGI 服务器的响应的缓冲。

`on`	Angie 尽快从 SCGI 服务器接收响应，将其保存到由*scgi_buffer_size* 和*scgi_buffers* 指令设置的缓冲区中。向客户端发送会并行进行：已填满的缓冲区会被传递用于发送，但总量不超过*scgi_busy_buffers_size*。如果缓冲区未被完全填满，则不会被传递用于发送，除非这是响应的最后一部分。因此，当需要即时传输每隔几个字节时，缓冲读取模式并不适合。如果整个响应无法放入内存，可以将其中一部分保存到磁盘上的临时文件 中。写入临时文件由*scgi_max_temp_file_size* 和*scgi_temp_file_write_size* 指令控制。
`off`	响应在接收到时立即传递给客户端。Angie 以” 读取—发送” 的循环工作，不会等待缓冲区被填满：例如从 4K 缓冲区读取到 10 字节时就会立即发送。同时，如果整个响应可以放入缓冲区,Angie 也可能一次读完。Angie 一次可以从服务器接收的最大数据量由*scgi_buffer_size* 指令设置。使用 `off` 时,:ref:*scgi_limit_rate* 不生效。

也可以通过在 `X-Accel-Buffering` 响应头字段中传递” yes” 或” no” 来启用或禁用缓冲。可以使用*scgi_ignore_headers* 指令禁用此功能。

scgi_buffers

语法	`scgi_buffers` *number size*;
默认值	`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。根据平台不同, 这可能是 4K 或 8K。

scgi_busy_buffers_size

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

当启用来自 SCGI 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向客户端发送响应的缓冲区的总大小。同时, 其余缓冲区可用于读取响应, 如果需要, 还可以将部分响应缓冲到临时文件中。

默认情况下, 大小受*scgi_buffer_size* 和*scgi_buffers* 指令设置的两个缓冲区大小的限制。

scgi_cache

语法	`scgi_cache` *zone*	`off`;
默认值	`scgi_cache off;`
上下文	http, server, location

定义用于缓存的共享内存区域。同一区域可以在多个地方使用。参数值可以包含变量。

`off`	禁用从上一配置级别继承的缓存。

备注

使用*scgi_cache* 时, 通常还需要设置*scgi_cache_valid* 指令来明确指定响应的缓存时间。如果未设置,Angie 不使用默认值, 而是根据来自服务器的 HTTP 响应头按以下优先级顺序确定响应缓存时间:

1. `X-Accel-Expires` 头 (最高优先级)。
2. 带有 `max-age` 或 `s-maxage` 参数的 `Cache-Control` 头。
3. `Expires` 头。

如果这些头都不包含有效值或根本不存在, 响应将不会被缓存, 因为无法确定其过期时间。

scgi_cache_background_update

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

允许启动后台子请求来更新过期的缓存项, 同时将陈旧的缓存响应返回给客户端。

警告

注意, 必须允许 在更新时使用陈旧的缓存响应。

scgi_cache_bypass

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

定义不从缓存中获取响应的条件。如果字符串参数中至少有一个值不为空且不等于”0”, 则不会从缓存中获取响应:

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

参数值可以包含变量。

可以与*scgi_no_cache* 指令一起使用。

scgi_cache_key

语法	scgi_cache_key *string*;
默认值	—
上下文	http, server, location

定义缓存的键。键可以包含变量。例如

```
scgi_cache_key localhost:9000$request_uri;
```

scgi_cache_lock

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

启用后，一次只允许一个请求通过将请求传递给 SCGI 服务器来填充根据*scgi_cache_key* 指令标识的新缓存元素。同一缓存元素的其他请求将等待响应出现在缓存中或此元素的缓存锁被释放，最长等待时间由*scgi_cache_lock_timeout* 指令设置。

scgi_cache_lock_age

语法	scgi_cache_lock_age *time*;
默认值	scgi_cache_lock_age 5s;
上下文	http, server, location

如果传递给 SCGI 服务器以填充新缓存元素的最后一个请求在指定时间内未完成，则可以再传递一个请求给 SCGI 服务器。

scgi_cache_lock_timeout

语法	scgi_cache_lock_timeout *time*;
默认值	scgi_cache_lock_timeout 5s;
上下文	http, server, location

设置*scgi_cache_lock* 的超时时间。当时间到期时，请求将被传递给 SCGI 服务器，但响应不会被缓存。

scgi_cache_max_range_offset

语法	scgi_cache_max_range_offset *number*;
默认值	—
上下文	http, server, location

为字节范围请求设置偏移量 (以字节为单位)。如果范围超出偏移量，范围请求将被传递给 SCGI 服务器，并且响应不会被缓存。

scgi_cache_methods

语法	scgi_cache_methods GET | HEAD | POST ...;
默认值	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" 方法始终会被添加到列表中，但建议显式指定它们。另请参阅*scgi_no_cache* 指令。

scgi_cache_min_uses

语法	`scgi_cache_min_uses` *number*;
默认值	`scgi_cache_min_uses 1;`
上下文	http, server, location

设置在多少次请求后响应将被缓存。

警告

缓存元数据存储在共享内存中。手动删除缓存文件不会重置计数器，可能导致不可预测的行为。要完全重置缓存，请停止服务器，删除缓存目录，然后重新启动。

备注

第三方缓存清除模块（例如 external-cache-purge）仅删除文件，但不会重置 *scgi_cache_min_uses* 计数器。该指令旨在保护缓存免受不频繁请求的污染，在清除期间重置计数器可能会对性能产生负面影响。

scgi_cache_path

语法	`scgi_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`	定义缓存的层次结构级别：从 1 到 3，每个级别接受值 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
```

缓存的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。但是，请注意，在这种情况下，文件将在两个文件系统之间复制，而不是廉价的重命名操作。因此，建议对于任何给定位置，缓存和保存临时文件的目录都放在同一文件系统上。

`use_temp_path=on | off`	确定用于临时文件的目录
`on`	如果未指定该参数或设置为 `on`，将使用给定 *location* 的*scgi_temp_path* 指令指定的目录
`off`	临时文件将直接放置在缓存目录中
`keys_zone`	设置用于存储所有活动键和数据信息的共享内存区域的名称和大小。缓存元数据存储在共享内存中。 1 兆字节的区域足以存储约 8000 个键。 当使用 `keys_zone` 指定可选的 *file* 时，Angie 会在主进程终止时将该区域的内容转储到磁盘，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它，以确保更可靠的数据持久性并减少缓存加载时间。 如果由于区域大小更改、二进制版本不兼容或其他原因而无法恢复该区域，Angie 将记录警告（`failed to restore zone at address`）并且不会使用区域恢复机制。相反，不兼容的文件将被重命名为 `.old`；您可以删除它，或恢复其名称并将 Angie 还原到最初创建此文件的配置和版本。 **警告** 确保正确指定 *file* 的路径，并具有 Angie 使用所需的访问权限，并受到保护以防止未经授权的访问；相对路径基于 prefix。
`inactive`	如果在此参数指定的时间内未访问缓存数据，则无论其新鲜度如何，数据都将被删除；每次访问都会重置此计时器。这与下文所述的、由 ** 缓存管理器 ** 执行的基于大小的清除不同：即使缓存未达到 `max_size`，基于不活动的删除也会进行，并且会忽略由*scgi_cache_valid* 配置的有效期。 默认情况下，`inactive` 为 10 分钟。

特殊的 ** 缓存管理器 ** 进程监控最大缓存大小和缓存所在文件系统的最小可用空间量，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。数据以迭代方式删除。

`max_size`	最大缓存大小
`min_free`	缓存所在文件系统的最小可用空间量
`manager_files`	限制一次迭代期间要删除的项目数 默认情况下，`100`
`manager_threshol`	限制一次迭代的持续时间 默认情况下，`200` 毫秒
`manager_sleep`	配置迭代之间的暂停 默认情况下，`50` 毫秒

Angie 启动一分钟后，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。加载也以迭代方式完成。

`loader_files`	限制一次迭代期间要加载的项目数 默认情况下，100
`loader_threshold`	限制一次迭代的持续时间 默认情况下，200 毫秒
`loader_sleep`	配置迭代之间的暂停 默认情况下，50 毫秒

scgi_cache_revalidate

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

启用使用带有 `If-Modified-Since` 和 `If-None-Match` 头字段的条件请求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_503 | http_403 | http_404 | http_429 | off ...;`
默认值	`scgi_cache_use_stale off;`
上下文	http, server, location

确定在哪些情况下可以使用过期的缓存响应。该指令的参数与*scgi_next_upstream* 指令的参数匹配。

`error`	如果无法选择 SCGI 服务器来处理请求，则允许使用过期的缓存响应。
`updating`	附加参数，如果当前正在更新过期的缓存响应，则允许使用它。这可以最大限度地减少更新缓存数据时对 SCGI 服务器的访问次数。

也可以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：

- `Cache-Control` 头字段的 stale-while-revalidate 扩展允许在当前正在更新过期的缓存响应时使用它。
- `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

此方法的优先级低于使用指令设置参数。

要在填充新缓存元素时最大限度地减少对 SCGI 服务器的访问次数，可以使用*scgi_cache_lock* 指令。

scgi_cache_valid

语法	`scgi_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

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

为代码 200 和 302 的响应设置 10 分钟的缓存，为代码 404 的响应设置 1 分钟的缓存。

如果仅指定缓存时间，

```
scgi_cache_valid 5m;
```

则仅缓存 200、301 和 302 响应。

此外，可以使用 `any` 参数指定缓存任何响应：

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

备注

缓存参数也可以直接在响应头中设置。这比使用指令设置缓存时间具有更高的优先级。

- `X-Accel-Expires` 头字段以秒为单位设置响应的缓存时间。零值将禁用响应的缓存。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），响应可以缓存到该时间。
- 如果头不包含 `X-Accel-Expires` 字段，则可以在头字段 `Expires` 或 `Cache-Control` 中设置缓存参数。
- 如果头包含 `Set-Cookie` 字段，则此类响应将不会被缓存。
- 如果头包含特殊值"*" 的 `Vary` 字段，则此类响应将不会被缓存。如果头包含其他值的 `Vary` 字段，则此类响应将根据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中的一个或多个的处理。

scgi_connect_timeout

语法	`scgi_connect_timeout` *time*;
默认值	`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`;
默认值	`scgi_connection_drop off;`
上下文	http, server, location

启用在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，终止到该服务器的所有连接。

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；设置为 `on` 时，连接将立即断开。

scgi_force_ranges

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

无论 SCGI 服务器响应中的 `Accept-Ranges` 字段如何，都为来自 SCGI 服务器的缓存和非缓存响应启用字节范围支持。

scgi_hide_header

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

默认情况下，Angie 不会将 SCGI 服务器响应中的头字段 `Status` 和 `X-Accel-...` 传递给客户端。`scgi_hide_header` 指令设置不会被传递的其他字段。相反，如果需要允许传递字段，可以使用*scgi_pass_header* 指令。

scgi_ignore_client_abort

语法	`scgi_ignore_client_abort on	off;`
默认值	`scgi_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 SCGI 服务器的连接。

scgi_ignore_headers

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

禁用对来自 SCGI 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，处理这些头字段将产生以下效果:

- `X-Accel-Expires`、`Expires`、`Cache-Control`、`Set-Cookie` 和 `Vary` 设置响应缓存 的参数;
- `X-Accel-Redirect` 执行到指定 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` 设置响应所需的字符集。

scgi_intercept_errors

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

确定代码大于或等于 300 的 SCGI 服务器响应应传递给客户端，还是被拦截并重定向到 Angie 以使用*error_page* 指令进行处理。

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* 以每秒字节数指定; 可以使用变量。

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 同时打开两个到 SCGI 服务器的连接, 则总速率将是指定限制的两倍。该限制仅在启用来自 SCGI 服务器的响应缓冲 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

当启用来自 SCGI 服务器的响应缓冲 时, 如果整个响应不适合*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。

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

指定在哪些情况下应将请求传递到下一个服务器:

error	与服务器建立连接、向其传递请求或读取响应头时发生错误;
timeout	与服务器建立连接、向其传递请求或读取响应头时发生超时;
invalid_header	服务器返回空响应或无效响应;
http_500	服务器返回代码为 500 的响应;
http_503	服务器返回代码为 503 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 方法 (POST、LOCK、PATCH) 的请求如果已经发送到上游服务器, 则不会传递到下一个服务器; 启用此选项将明确允许重试此类请求;
off	禁用将请求传递到下一个服务器。

备注

需要注意的是, 只有在尚未向客户端发送任何内容时, 才可能将请求传递到下一个服务器。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	始终被视为不成功尝试, 即使未在指令中指定
http_500 http_503 http_429	仅在指令中指定时才被视为不成功尝试
http_403 http_404	永远不会被视为不成功尝试

将请求传递到下一个服务器可以通过尝试次数 和时间 进行限制。

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

限制可以将请求传递到下一个 服务器的时间。

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

限制将请求传递到下一个 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

定义响应不会保存到缓存的条件。如果字符串参数中至少有一个值不为空且不等于”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;
```

参数值可以包含变量。

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。当且仅当当前级别上没有定义 scgi_param 指令时, 这些指令才会从上一级配置继承。

标准 CGI 环境变量 应作为 SCGI 头提供, 请参阅发行版中提供的 scgi_params 文件:

```
location / {
    include scgi_params;
#    ...
}
```

在标准 `scgi_params` 文件中,:samp:*REQUEST_METHOD* 设置为 `$upstream_request_method`。

如果指令使用 `if_not_empty` 指定, 则仅当参数值不为空时才会将其传递给服务器:

```
scgi_param HTTPS $https if_not_empty;
```

scgi_pass

语法	`scgi_pass` *uri*;
默认值	—
上下文	location, if in location

设置 SCGI 服务器的地址。地址可以指定为域名或 IP 地址, 以及可选的端口:

```
scgi_pass localhost:9000;
```

或作为 UNIX 域套接字路径:

```
scgi_pass unix:/tmp/scgi.socket;
```

如果域名解析为多个地址, 则所有地址都将以轮询方式使用。此外, 地址可以指定为服务器组。

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。

备注

如果在前缀带有尾部斜杠的 `location` 中指定了 `scgi_pass` (例如 `location /name/`), 并且*auto_redirect* 指令设置为 `default`, 则不带尾部斜杠的请求将被重定向 (`/name -> /name/`)。

scgi_pass_header

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

允许将原本被禁用的 头字段从 SCGI 服务器传递给客户端。

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。另请参阅*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。另请参阅*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。超时仅在两次连续的读操作之间设置，而不是用于整个响应的传输。如果 SCGI 服务器在此时间内未传输任何内容，则连接将被关闭。

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，从客户端完整读取 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，则无法将请求传递到下一个服务器。

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。

scgi_send_timeout

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

设置向 SCGI 服务器传输请求的超时时间。超时仅在两次连续的写操作之间设置，而不是用于整个请求的传输。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;
默认值	scgi_socket_keepalive off;
上下文	http, server, location

配置到 SCGI 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	为套接字启用 *SO_KEEPALIVE* 套接字选项。

scgi_store

语法	scgi_store on | off | *string*;
默认值	scgi_store off;
上下文	http, server, location

启用将文件保存到磁盘。

on	使用与*alias* 或*root* 指令对应的路径保存文件
off	禁用保存文件

可以使用带变量的 *string* 显式设置文件名：

```
scgi_store /data/www$original_uri;
```

文件的修改时间根据接收到的 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。但是请注意, 在这种情况下, 文件将跨两个文件系统复制, 而不是进行廉价的重命名操作。因此建议对于任何给定位置, 保存的文件和由*scgi_temp_path* 指令设置的临时文件目录都放在同一文件系统上。

此指令可用于创建静态不变文件的本地副本, 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 *users:permissions ...*;
默认值	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, 则可以省略用户权限:

```
scgi_store_access group:rw all:r;
```

scgi_temp_file_write_size

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

当启用将来自 SCGI 服务器的响应缓冲到临时文件时，限制一次写入临时文件的数据大小。默认情况下，大小受*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 *path* [*level1* [*level2* [*level3*]]]‘;
默认值	scgi_temp_path scgi_temp; (路径取决于 --http-scgi-temp-path 构建选项)
上下文	http, server, location

定义用于存储从 SCGI 服务器接收的数据的临时文件的目录。在指定目录下最多可以使用三级子目录层次结构。例如, 在以下配置中

```
scgi_temp_path /spool/angie/scgi_temp 1 2;
```

临时文件可能如下所示：

```
/spool/angie/scgi_temp/7/45/00000123457
```

另请参阅*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。表达式可以包含变量。

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*$slice_range* 变量作为:samp:*Range*‘请求头字段传递给代理服务器。如果启用了*:ref:*‘缓存 *<proxy_cache>*，应将*$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:samp:‘ (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_gmt`

当前时间（GMT）。格式由 *config* 命令中的 *timefmt* 参数设置。

`$date_local`

当前时间（本地时区）。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, 此模块默认不会被构建; 应使用 `--with-http_ssl_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。

备注

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 数量设置为等于处理器数量,
- 启用*keep-alive* 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 (默认为 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*;
默认值	ssl_buffer_size 16k;
上下文	http, server

设置用于发送数据的缓冲区大小。

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。为了最小化首字节时间, 使用较小的值可能是有益的, 例如:

```
ssl_buffer_size 4k;
```

ssl_certificate

语法	ssl_certificate *file*;
默认值	—
上下文	http, server

为给定的虚拟服务器指定 PEM 格式的证书文件。如果除了主证书之外还应指定中间证书, 则应按以下顺序在同一文件中指定它们: 首先是主证书, 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              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 或更高版本支持为不同证书使用单独的证书链。对于较旧的版本，只能使用一个证书链。

备注

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，使用变量意味着将为每次 SSL 握手加载证书，这可能会对性能产生负面影响。

可以指定值 `data:$variable` 来代替 *file*，这将从变量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，例如将私钥数据写入错误日志。

备注

应该记住，由于 HTTPS 协议的限制，为了最大的互操作性，虚拟服务器应该监听不同的 *IP* 地址。

如果启用了 *ssl_ntls*，该指令可以接受两个参数 (证书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_key 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

定义一个缓存，用于存储使用变量指定的 *SSL* 证书 和私钥。

该指令支持以下参数：

- `max` —设置缓存中的最大元素数量。当缓存溢出时，最近最少使用 (LRU) 的元素将被移除。

- `inactive` —定义元素在此时间内未被访问即会被移除的时长，即使其仍处于 `valid` 周期内；每次访问都会重置此计时器。默认为 10 秒。
- `valid` —定义缓存元素被视为有效并可以重用的时间。默认为 60 秒。在此期间之后，证书将被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_compression

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

启用 TLS 1.3 服务器证书 压缩。

备注

在使用 OpenSSL 3.2 或更高版本时支持该指令; 支持的压缩算法列表由库提供。

备注

在使用 BoringSSL 时支持该指令; 支持的压缩算法列表包括 `zlib`。

如果启用了*ssl_stapling*, 则禁用证书压缩。

ssl_certificate_key

语法	`ssl_certificate_key` *file*;
默认值	—
上下文	http, server

为给定的虚拟服务器指定 PEM 格式的私钥文件。

备注

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*, 这将从 OpenSSL 引擎 *name* 加载具有指定 *id* 的私钥。

可以指定值 `store:scheme:id` 来代替 *file*, 用于从具有指定 *id* 和 OpenSSL 提供程序注册的 URI *scheme* 加载私钥, 例如 pkcs11。

可以指定值 `data:$variable` 来代替 *file*, 这将从变量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带来安全隐患, 例如将私钥数据写入错误日志。

如果启用了*ssl_ntls*, 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_key rsa.key;
```

ssl_ciphers

语法	`ssl_ciphers` *ciphers*;
默认值	`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;
```

密码套件列表取决于安装的 OpenSSL 版本。可以使用 `openssl ciphers` 命令查看完整列表。

警告

使用 OpenSSL 时,:samp:*ssl_ciphers* 指令 不配置 TLS 1.3 的密码套件。要使用 OpenSSL 配置 TLS 1.3 密码套件, 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,TLS 1.3 密码套件 可以使用 `ssl_ciphers` 配置。
- 在 BoringSSL 中,TLS 1.3 密码套件根本无法配置。

ssl_client_certificate

语法	`ssl_client_certificate` *file*;
默认值	—
上下文	http, server

指定 PEM 格式的受信任 CA 证书文件, 用于验证 客户端证书, 以及在启用*ssl_stapling* 时验证 OCSP 响应。

证书列表将发送给客户端。如果不希望这样做, 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	http, server

设置任意 OpenSSL 配置命令。

备注

在使用 OpenSSL 1.0.2 或更高版本时支持该指令。要使用 OpenSSL 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。

警告

直接配置 OpenSSL 可能导致意外行为。

ssl_crl

语法	`ssl_crl` *file*;
默认值	—
上下文	http, server

指定用于验证 客户端证书的 PEM 格式吊销证书 (CRL) 文件。

ssl_dhparam

语法	`ssl_dhparam` *file*;
默认值	—
上下文	http, server

指定一个包含 DHE 密码套件的 DH 参数的文件。

警告

默认情况下不设置参数, 因此不会使用 DHE 密码套件。

ssl_early_data

语法	`ssl_early_data on | off;`
默认值	`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;
```

备注

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*;
默认值	`ssl_ecdh_curve auto;`
上下文	http, server

指定 ECDHE 密码套件的曲线。

备注

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` 对应于 OpenSSL 1.0.2 或更高版本中 OpenSSL 库内置的曲线列表，或者在较旧版本中对应于 *prime256v1*。

备注

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_encrypted_hello_key

Added in version 1.11.0.

语法	`ssl_encrypted_hello_key` *file*;
默认值	—
上下文	http, server

指定一个包含 PEM 格式的 ECH 私钥和 ECHConfigList 的文件。该指令可以指定多次。需要支持加密客户端问候 (ECH) 的 OpenSSL 或 BoringSSL 构建；否则不支持。

ssl_keylog_file

Added in version 1.12.0.

语法	`ssl_keylog_file` *file*;
默认值	—
上下文	http, server

指定一个文件，用于记录由该上下文处理的连接的 SSL/TLS 密钥，采用 SSLKEYLOGFILE 格式，Wireshark 等工具使用该格式来解密捕获的 TLS 流量。

该指令不应与 LibreSSL 一起使用。

警告

该文件包含敏感的 TLS 密钥材料，可用于解密相应连接的已捕获流量。请仅将该指令用于调试或协议分析，限制对该文件的访问，并在不再需要时将其删除。

ssl_ntls

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

在使用 TongSuo TLS 库时启用服务器端对 NTLS 的支持。

```
listen ... ssl;
ssl_ntls  on;
```

备注

必须使用 --with-ntls 配置参数构建 Angie, 并使用相应的支持 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;
默认值	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 验证正常工作,:ref:*ssl_verify_client* 指令应设置为 on 或 optional。

要解析 OCSP 响应者主机名, 也应指定*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*];
默认值	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*;
默认值	—
上下文	http, server

覆盖在 "Authority Information Access" 证书扩展中指定的 OCSP 响应者的 URI, 用于验证 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*;
默认值	—
上下文	http, server

指定一个包含私钥 密码短语的文件, 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;
默认值	ssl_prefer_server_ciphers off;
上下文	http, server

指定在使用 SSLv3 和 TLS 协议时, 服务器密码套件应优于客户端密码套件。

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

启用指定的协议。

备注

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;
默认值	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, 与服务器名称不是 *example.com* 的 SSL 握手将被拒绝:

```
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*];
默认值	`ssl_session_cache none;`
上下文	http, server

设置用于存储会话参数的缓存类型和大小。缓存可以是以下任意类型：

`off`	严格禁止使用会话缓存：Angie 明确告知客户端会话不可重用。
`none`	温和地禁止使用会话缓存：Angie 告知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 内置缓存；仅由一个工作进程使用。缓存大小以会话数指定。如果未指定大小，则等于 20480 个会话。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；1 兆字节可以存储约 4000 个会话。每个共享缓存应有一个任意名称。具有相同名称的缓存可以在多个虚拟服务器中使用。除非使用*ssl_session_ticket_key* 指令显式配置，否则它还用于自动生成、存储和定期轮换 TLS 会话票据密钥。

两种缓存类型可以同时使用，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*;
默认值	—
上下文	http, server

设置包含用于加密和解密 TLS 会话票据的密钥的文件。如果需要在多个服务器之间共享相同的密钥，则此指令是必需的。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，将使用 AES256（用于 80 字节密钥）或 AES128（用于 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	http, server

启用或禁用通过 TLS 会话票据 进行会话恢复。

ssl_session_timeout

语法	`ssl_session_timeout` *time*;
默认值	`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;`
默认值	`ssl_stapling off;`
上下文	http, server

启用或禁用服务器 装订 OCSP 响应。示例：

```
ssl_stapling on;
resolver 127.0.0.53;
```

要使 OCSP 装订正常工作，应该知道服务器证书颁发者的证书。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* 指令指定的文件中。

警告

为了解析 OCSP 响应者主机名，还应指定*resolver* 指令。

ssl_stapling_file

语法	`ssl_stapling_file` *file*;
默认值	—
上下文	http, server

设置后，装订的 OCSP 响应将从指定的文件中获取，而不是查询服务器证书中指定的 OCSP 响应者。

该文件应为 DER 格式，由 `openssl ocsp` 命令生成。

ssl_stapling_responder

语法	`ssl_stapling_responder uri;`
默认值	—
上下文	http, server

覆盖证书扩展 "Authority Information Access" 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;`
默认值	`ssl_stapling_verify off;`
上下文	http, server

启用或禁用服务器对 OCSP 响应的验证。

要使验证正常工作，应使用*ssl_trusted_certificate* 指令将服务器证书颁发者的证书、根证书和所有中间证书配置为受信任。

ssl_trusted_certificate

语法	`ssl_trusted_certificate` *file*;
默认值	—
上下文	http, server

指定包含 PEM 格式受信任 CA 证书的文件，用于验证 客户端证书，以及在启用*ssl_stapling* 时验证 OCSP 响应。

与*ssl_client_certificate* 设置的证书集不同，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

启用客户端证书验证。验证结果存储在*$ssl_client_verify* 变量中。

`optional`	请求客户端证书，如果证书存在则进行验证。
`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。这适用于由 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` *number*;
默认值	`ssl_verify_depth 1;`
上下文	http, server

设置客户端证书链中的验证深度。

错误处理

`http_ssl` 模块支持几个非标准错误代码，可以使用*error_page* 指令进行重定向：

495	客户端证书验证过程中发生错误；
496	客户端未提供所需的证书；
497	常规请求已发送到 HTTPS 端口。

重定向发生在请求完全解析之后，此时变量（如*$request_uri*、*$uri*、*$args* 等）已可用。

内置变量

http_ssl 模块支持内置变量：

$ssl_alpn_protocol

返回在 SSL 握手期间通过 ALPN 选择的协议，否则返回空字符串。

$ssl_cipher

返回已建立的 SSL 连接所使用的密码套件名称。

$ssl_ciphers

返回客户端支持的密码套件列表。已知的密码套件按名称列出，未知的以十六进制显示，例如：

```
AES128-SHA:AES256-SHA:0x00ff
```

备注

仅在使用 OpenSSL 1.0.2 或更高版本时才完全支持该变量。对于较旧版本，该变量仅适用于新会话，并且仅列出已知的密码套件。

`$ssl_client_escaped_cert`

返回已建立的 SSL 连接的 PEM 格式客户端证书（URL 编码）。

`$ssl_client_fingerprint`

返回已建立的 SSL 连接的客户端证书的 SHA1 指纹。

`$ssl_client_i_dn`

根据 RFC 2253 返回已建立的 SSL 连接的客户端证书的" 颁发者 DN" 字符串。

`$ssl_client_i_dn_legacy`

返回已建立的 SSL 连接的客户端证书的" 颁发者 DN" 字符串。

`$ssl_client_raw_cert`

返回已建立的 SSL 连接的 PEM 格式客户端证书。

`$ssl_client_s_dn`

根据 RFC 2253 返回已建立的 SSL 连接的客户端证书的" 主题 DN" 字符串。

`$ssl_client_s_dn_legacy`

返回已建立的 SSL 连接的客户端证书的" 主题 DN" 字符串。

`$ssl_client_serial`

返回已建立的 SSL 连接的客户端证书的序列号。

`$ssl_client_sigalg`

返回已建立的 SSL 连接的客户端证书的 '签名算法 <https://www.iana.org/assignments/tls-parameters/tls-parameters.xhtml#tls-parameters-16>'_。

备注

仅在使用 OpenSSL 3.5 或更高版本时才支持该变量。对于较旧版本，该变量值将为空字符串。

备注

该变量仅适用于新会话。

`$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剩余天数。

`$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`，如果未提供证书则返回 `NONE`。

`$ssl_curve`

返回在 SSL 握手期间用于密钥交换的协商曲线。已知的曲线按名称列出，未知的以十六进制显示，例如：

```
prime256v1
```

备注

仅在使用 OpenSSL 3.0 或更高版本时才支持该变量。对于较旧版本，该变量值将为空字符串。

`$ssl_curves`

返回客户端支持的曲线列表。已知的曲线按名称列出，未知的以十六进制显示，例如：

```
0x001d:prime256v1:secp521r1:secp384r1
```

备注

仅在使用 OpenSSL 1.0.2 或更高版本时才支持该变量。对于较旧版本，该变量值将为空字符串。

该变量仅适用于新会话。

`$ssl_early_data`

如果使用 TLS 1.3 早期数据 且握手未完成，则返回”1”，否则返回””。

`$ssl_encrypted_hello`

Added in version 1.11.0.

如果使用加密客户端问候（ECH），则返回”1”，否则返回””。

`$ssl_protocol`

返回已建立的 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`

返回已建立的 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_sigalg`

返回已建立的 SSL 连接的服务器证书的 ‘签名算法 <https://www.iana.org/assignments/tls-parameters/tls-parameters.xhtml#tls-parameters-16>‘_。

备注

仅在使用 OpenSSL 3.5 或更高版本时才支持该变量。对于较旧版本，该变量值将为空字符串。

备注

该变量仅适用于新会话。

`$ssl_sigalgs`

返回客户端在 SSL 握手期间提供的 签名算法 列表，以冒号分隔。

备注

对于 OpenSSL 4.0 或更高版本，算法以名称形式返回；没有名称的算法回退为其原始十六进制代码。暴露旧版签名算法 API 的 SSL 库将所有算法作为原始十六进制代码返回。不支持签名算法 API 的构建返回空值。

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_waiting`

与 *Waiting* 的值相同。

`$connections_writing`

与 *Writing* 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。

隧道

Added in version 1.12.0.

隧道模块（ngx_http_tunnel_module）处理 HTTP/1.1 CONNECT 请求，并在客户端与后端服务器之间建立端到端的虚拟连接，使 Angie 能够充当转发代理。

对隧道的访问由标准的访问控制指令进行控制：*auth_basic*、*satisfy* 和*auth_delay*。

配置示例

```
http {

    map $host $allow_host {
        hostnames;

        example.org    1;
        *.example.org  1;
    }

    server {
        listen 8000;

        resolver 127.0.0.53;

        if ($allow_host != 1) {
            return 502;
        }

        tunnel_pass;
    }
}
```

指令

tunnel_bind

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

使到后端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 会取消从上一级配置继承的 tunnel_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 参数允许到后端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：

```
tunnel_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 上则不需要这样做，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。

tunnel_buffer_size

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

设置用于从后端服务器读取数据的缓冲区的 *size*。它同时也设置用于从客户端读取数据的缓冲区的 *size*。

tunnel_connect_timeout

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

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

tunnel_next_upstream

语法	`tunnel_next_upstream error	timeout	off` ...;
默认值	`tunnel_next_upstream error timeout;`
上下文	http, server, location

指定在哪些情况下应将请求传递到下一个服务器：

`error`	与服务器建立连接或从其读取数据时发生错误；
`timeout`	与服务器建立连接、向其传递请求或从其读取数据时发生超时；
`off`	禁用将请求传递到下一个服务器。

只有在尚未向客户端发送任何内容时，才能将请求传递到下一个服务器。如果在传输响应的过程中发生错误或超时，则无法对其进行修复。

该指令还定义了何种情况被视为与服务器通信的失败尝试。其中，`error` 和 `timeout` 的情况始终被视为失败尝试，即使它们未在指令中指定。

将请求传递到下一个服务器的操作可以通过尝试次数 和时间 加以限制。

tunnel_next_upstream_timeout

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

限制可以将请求传递到下一个服务器 的时间。值 `0` 将关闭此限制。

tunnel_next_upstream_tries

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

限制将请求传递到下一个服务器 的可能尝试次数。值 `0` 将关闭此限制。

tunnel_pass

语法	`tunnel_pass` [*address*];
默认值	—
上下文	server, location, if in location

启用对 CONNECT 请求的处理，并设置后端服务器的地址。默认情况下，*address* 为 `$host:$request_port`，取自客户端请求。在大多数情况下，`tunnel_pass` 不需要配置任何参数。

地址可以指定为域名或 IP 地址，以及端口：

```
tunnel_pass localhost:9000;
```

或指定为 UNIX 域套接字路径：

```
tunnel_pass unix:/tmp/backend.socket;
```

如果一个域名解析到多个地址，则将以轮询 方式使用所有这些地址。此外，地址还可以指定为服务器组。

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

tunnel_read_timeout

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

设置客户端或后端服务器连接上两次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任何数据，则连接将被关闭。

tunnel_send_lowat

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

如果该指令设置为非零值，Angie 将尝试尽量减少到后端服务器的出站连接上的发送操作次数，方法是使用 `kqueue` 方法的 `NOTE_LOWAT` 标志，或使用带有指定 *size* 的 `SO_SNDLOWAT` 套接字选项。

此指令在 Linux、Solaris 和 Windows 上会被忽略。

tunnel_send_timeout

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

设置向后端服务器传输数据的超时时间。该超时仅设置在两次连续写操作之间，而非整个传输过程。如果后端服务器在此时间内未收到任何数据，则连接将被关闭。

tunnel_socket_keepalive

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

为到后端服务器的出站连接配置”TCP keepalive” 行为。默认情况下，套接字采用操作系统的设置。如果该指令设置为 `on` 值，则会为套接字启用 `SO_KEEPALIVE` 套接字选项。

Upstream

提供一个上下文，用于描述可在*proxy_pass*、*fastcgi_pass*、*uwsgi_pass*、*scgi_pass*、*memcached_pass* 和*grpc_pass* 指令中使用的服务器组。

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)

语法	backup_switch permanent[=*time*];
默认值	—
上下文	upstream

该指令启用从 活动组 (即上次成功找到服务器的组) 而不是主组开始服务器选择的能力。如果在活动组中无法为下一个请求找到服务器, 并且搜索移动到备份组, 则该组将成为活动组, 后续请求将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, 则该组在选择后保持活动状态, 并且不会自动重新检查较低级别的组。如果指定了 *time*, 则该组的活动状态在指定的时间间隔后过期, 负载均衡器将再次检查较低级别的组, 如果服务器正常工作则返回到这些组。

示例:

```
upstream my_backend {
    zone my_backend 1m;
    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份组, 所有后续请求将在 2 分钟内由该备份组处理。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, 则再次使其成为活动组。

bind_conn (PRO)

语法	bind_conn *value*;
默认值	—
上下文	upstream

当指定为变量字符串的 *value* 不等于 "" 和 "0" 时，允许将服务器连接绑定到客户端连接。

警告

bind_conn 指令必须在所有设置负载均衡方法的指令之后使用，否则将不起作用。如果与*sticky* 指令一起使用，则 bind_conn 必须在 sticky 之后。

警告

使用该指令时,:ref:*Proxy <http_proxy>* 模块设置必须允许使用持久连接，例如:

```
proxy_http_version 1.1;
proxy_set_header Connection "";
```

该指令的典型用例是代理使用 NTLM 身份验证的连接，其中需要在协商开始时确保客户端到服务器的绑定:

```
map $http_authorization $ntlm {
    ~*^NTLM       1;
    ~*^Negotiate  1;
}

upstream ntlm_backend {
    zone ntlm_backend 1m;
    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)

在 1.12.0 版本发生变更: PRO

语法	`feedback` *variable* [`inverse`] [`factor=`*number*] [`account=`*condition_variable*] [`last_byte`];
默认值	—
上下文	upstream

在 `upstream` 中设置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均衡决策, 该值会根据 *variable* 的值随时间变化, 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。它应该表示性能或健康指标; 假定服务器在响应头或其他方式中提供该值。 该值在服务器的每个响应中进行评估, 并根据 `inverse` 和 `factor` 设置纳入移动平均值。
`inverse`	如果设置了该参数, 则反馈值将被反向解释: 较低的值表示更好的性能。
`factor`	计算平均值时考虑反馈值的因子。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; 如果指定为 90, 则将采用 90% 的先前值和仅 10% 的新值。
`account`	指定一个条件变量, 用于控制在计算中考虑哪些响应。仅当该响应的条件变量不等于 `""` 或 `"0"` 时, 才会使用该响应的反馈值更新平均值。 **备注** 默认情况下,:ref: 主动检查 *<u_upstream_probe>* 期间的响应不包含在计算中; 将*$upstream_probe* 变量与 `account` 结合使用可以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代理服务器的数据, 而不仅仅是响应头。

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";
}
```

此配置根据响应头字段中的特定分数按反馈级别对服务器响应进行分类，并在*$upstream_probe* 上添加条件，仅考虑来自 `high_priority` 主动检查的响应或对常规客户端请求的响应。

hash

语法	`hash` *key* [`consistent`];
默认值	—
上下文	upstream

为服务器组指定负载均衡方法，其中客户端到服务器的映射使用哈希键值确定。键可以包含变量。请注意，从组中添加或删除服务器可能会导致大多数键重新映射到不同的服务器。该方法与 Cache::Memcached Perl 库兼容。

```
hash $remote_addr;
```

当使用解析为多个 IP 地址的域名时（例如，使用 `resolve` 参数），服务器不会对接收到的地址进行排序，因此它们的顺序可能在不同服务器之间有所不同，这会影响客户端分布。为确保一致的分布，请使用 `consistent` 参数。

如果指定了 `consistent` 参数，将使用 ketama 一致性哈希方法。该方法确保当从组中添加或删除服务器时，只有少数键会重新映射到不同的服务器。这有助于为缓存服务器实现更高的缓存命中率。该方法与 Cache::Memcached::Fast Perl 库兼容，其中 `ketama_points` 参数设置为 160。

ip_hash

语法	`ip_hash`;
默认值	—
上下文	upstream

为服务器组指定负载均衡方法，根据客户端 IP 地址在服务器之间分配请求。使用客户端 IPv4 地址的前三个八位组或完整的 IPv6 地址作为哈希键。该方法确保来自同一客户端的请求始终传递到同一服务器，除非该服务器不可用。在这种情况下，客户端请求将传递到另一台服务器。最有可能的是，这也将是同一台服务器。

如果需要临时移除某台服务器, 应使用 `down` 参数标记它, 以保持客户端 IP 地址的当前哈希:

```
upstream backend {
    zone backend 1m;
    ip_hash;

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

keepalive

语法	`keepalive` *connections*;
默认值	—
上下文	upstream

激活到上游服务器的连接缓存。

`connections` 参数设置每个工作进程缓存中保留的到上游服务器的空闲 keepalive 连接的最大数量。当超过此数量时, 最近最少使用的连接将被关闭。

备注

需要特别注意的是,:samp:*keepalive* 指令不限制 Angie 工作进程可以打开的到上游服务器的连接总数。`connections` 参数应设置得足够低, 以便让上游服务器也能处理新的传入连接。

警告

`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。

使用 keepalive 连接的 memcached 上游配置示例:

```
upstream memcached_backend {
    zone memcached_backend 1m;
    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, 应将*proxy_http_version* 指令设置为"1.1", 并清除 Connection 头字段:

```
upstream http_backend {
    zone http_backend 1m;
    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

或者, 可以通过向上游服务器传递"Connection: Keep-Alive" 头字段来使用 HTTP/1.0 持久连接, 但不推荐使用此方法。

对于 FastCGI 服务器, 需要设置*fastcgi_keep_conn* 才能使 keepalive 连接工作:

```
upstream fastcgi_backend {
    zone fastcgi_backend 1m;
    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 keepalive 连接的概念。

keepalive_requests

语法	`keepalive_requests` *number*;
默认值	`keepalive_requests 1000;`
上下文	upstream

设置可以通过一个 keepalive 连接处理的最大请求数。达到最大请求数后, 连接将被关闭。

定期关闭连接对于释放每个连接的内存分配是必要的。因此, 使用过高的最大请求数可能导致内存使用过多, 不建议这样做。

keepalive_time

语法	`keepalive_time` *time*;
默认值	`keepalive_time 1h;`
上下文	upstream

限制可以通过一个 keepalive 连接处理请求的最长时间。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*;
默认值	`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

指定服务器组应使用负载均衡方法, 将请求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, 则使用加权轮询均衡方法依次尝试它们。

least_time

语法	`least_time header | last_byte [factor=`*number*`] [account=`*condition_variable*`];`
默认值	—
上下文	upstream

指定服务器组应使用负载均衡方法, 其中活动服务器接收请求的机会与其平均响应时间成反比; 响应时间越短, 服务器获得的请求越多。

`header`	该指令计算接收响应头的平均时间。
`last_byte`	该指令使用接收整个响应的平均时间。

`factor`	与*response_time_factor* 的作用相同, 如果设置了该参数, 则会覆盖它。
`account`	指定一个条件变量, 用于控制哪些响应包含在计算中。仅当响应的条件变量不是 `""` 或 `"0"` 时, 才会更新平均值。 **备注** 默认情况下, 探测 *(PRO)* 的响应不包含在计算中; 将*$upstream_probe* 变量与 `account` 结合使用, 可以包含这些响应, 甚至排除其他所有响应。

当前值在 API 的上游指标 中以服务器的 `health` 对象中的 `header_time` (仅头部) 和 `response_time` (整个响应) 形式呈现。

queue (PRO)

语法	`queue` *number* `[timeout=`*time*`];`
默认值	—
上下文	upstream

如果在第一次尝试时无法为请求分配代理服务器 (例如, 在短暂的服务中断期间或负载激增达到*max_conns* 限制时), 请求不会被拒绝; 相反,Angie 会尝试将其排队等待处理。

该指令的 number 参数设置工作进程 队列中的最大请求数。如果队列已满, 则向客户端返回 `502 (Bad Gateway)` 错误。

备注

proxy_next_upstream 指令的逻辑也适用于排队的请求。具体来说，如果为请求选择了服务器但无法将其交给该服务器，则请求可能会返回队列。

如果在 `timeout` 设置的时间 内 (默认为 60 秒) 未选择服务器来处理排队的请求，则向客户端返回 `502 (Bad Gateway)` 错误。过早关闭连接的客户端的请求也会从队列中删除；在*API* 中有通过队列的请求状态的计数器。

警告

`queue` 指令必须在所有设置负载均衡方法的指令之后使用；否则它将不起作用。

random

语法	`random [two];`
默认值	—
上下文	upstream

为服务器组指定负载均衡方法，将请求传递给随机选择的服务器，同时考虑服务器权重。

如果指定了可选的 `two` 参数,Angie 会随机选择两个服务器，然后使用*least_conn* 方法从中选择一个，即将请求传递给活动连接数最少的服务器。

response_time_factor

语法	`response_time_factor` *number*;
默认值	`response_time_factor 90;`
上下文	upstream

设置在使用 指数加权移动平均 公式计算*least_time* 负载均衡方法的平均响应时间时,** 先前 ** 值的平滑因子。

指定的 *number* 越高，新值对平均值的影响越小；如果指定为 `90`，则取先前值的 90% 和新值的 10%。有效值范围为 0 到 99(含)。

当前计算结果在 API 的上游指标 中以服务器 `health` 对象中的 `header_time` (仅头部) 和 `response_time` (完整响应) 形式呈现。

备注

计算中仅包含成功的响应；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*

指令确定。此外,:samp:*header_time* 值仅在接收并处理所有头部时重新计算, 而 `response_time` 仅在接收完整响应时重新计算。

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, 可带可选端口, 或指定为 UNIX 域套接字路径 (在 `unix:` 前缀之后)。如果未指定端口, 则使用端口 80。解析为多个 IP 地址的域名会一次性定义多个服务器。

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区 中, 则该限制按工作进程生效。

备注

启用空闲保持连接、多个工作进程 和共享内存区 时, 到代理服务器的活动和空闲连接总数可能超过 `max_conns` 值。

`max_fails=`*number* —设置在指定的*fail_timeout* 时间段内与服务器通信的失败尝试次数，达到该次数后服务器将被视为不可用；之后，将在相同的时间段后再次检查。

失败尝试的定义由*proxy_next_upstream*、*fastcgi_next_upstream*、*uwsgi_next_upstream*、*scgi_next_upstream*、*memcached_next_upstream* 和*grpc_next_upstream* 指令确定。

当超过 `max_fails` 时，从*upstream_probe (PRO)* 的角度来看，服务器也被视为不可用；客户端请求将不会被定向到该服务器，直到检查确定其可用为止。

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。

如果在解析所有 `server` 指令后，上游中仅剩一个服务器，则 `max_fails` 设置不起作用并将被忽略。

`max_fails=1`	默认尝试次数;
`max_fails=0`	禁用尝试计数。

`fail_timeout=`*time* —设置时间段，在此期间必须发生指定次数的与服务器通信失败尝试（*max_fails*），服务器才会被视为不可用。然后服务器在相同的时间段内保持不可用状态，之后再次被检查。

默认值为 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。

如果在解析所有 `server` 指令后，上游中仅剩一个服务器，则 `fail_timeout` 设置不起作用并将被忽略。

`backup`	将服务器标记为备份服务器。当主服务器不可用时，它将接收请求。 如果指定了*backup_switch (PRO)* 指令，其主动备份逻辑也将适用。
`down`	将服务器标记为永久不可用。
`drain`	将服务器标记为排空状态；这意味着它仅接收来自先前通过*sticky* 绑定的会话的请求。否则，行为与 `down` 模式相同。

警告

`backup` 参数不能与*hash*、*ip_hash* 和*random* 负载均衡方法一起使用。

`down` 和 `drain` 参数互斥。

`resolve`	允许监控与域名对应的 IP 地址列表的变化，并在不重新加载配置的情况下更新它。该组必须位于共享内存 中；还必须定义*resolver*。
`service=`*name*	启用 DNS SRV 记录的解析并设置服务名称。要使该参数生效，必须为服务器指定 `resolve` 参数，且主机名中不指定端口。 如果服务名称不包含点，则根据 RFC 标准形成名称：服务名称前加 `_` 前缀，然后在点后附加 `_tcp`。因此，服务名称 `http` 将变为 `_http._tcp`。 Angie 通过组合规范化的服务名称和主机名来解析 SRV 记录，并通过 DNS 获取结果组合的服务器列表，以及它们的优先级和权重。 • 具有最高优先级的 SRV 记录（优先级值最低的记录）被解析为主服务器，而其他记录成为备份服务器。如果 `server` 设置了 `backup`，则具有最高优先级的 SRV 记录被解析为备份服务器，其他记录被忽略。 • 权重类似于 `server` 指令的 `weight` 参数。如果在指令本身和 SRV 记录中都指定了权重，则使用指令中设置的权重。

在此示例中，对记录 `_http._tcp.backend.example.com` 执行查找：

```
server backend.example.com service=http resolve;
```

`sid=`*id*	设置组中的服务器 ID。如果未指定该参数，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。

`slow_start=`*time*	设置返回服务器权重逐步恢复的时间，当使用轮询 或*least_conn* 方法进行负载均衡时。 如果设置了该参数，并且从*max_fails* 和*upstream_probe (PRO)* 的角度来看，服务器在故障后再次被认为可运行，则该服务器会在给定的时间段内逐步增加到其指定的权重。 如果未设置该参数，在类似情况下，服务器会立即以其指定的权重开始运行。

备注

如果上游中仅指定了一个 `server`，`slow_start` 将不起作用并会被忽略。

state (PRO)

语法	`state` *file*;
默认值	—
上下文	upstream

指定一个 *file*，用于持久化存储上游服务器列表。当从 我们的软件包安装时，会专门创建目录 `/var/lib/angie/state/`（FreeBSD 上为 `/var/db/angie/state/`）来存储此类文件，并具有适当的访问权限，在配置中您只需添加文件名：

```
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 `server`。每当通过配置 API 在*/config/http/upstreams/* 部分更改服务器时，文件内容就会被修改。该文件在 Angie 启动或重新加载配置时读取。

警告

要在 `upstream` 块中使用 `state` 指令，其中不能有 `server` 指令，但需要共享内存区（*zone*）。

sticky

在 1.10.0 版本发生变更: PRO

在 1.11.0 版本发生变更.

在 1.11.0 版本发生变更: PRO

语法	`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*]; (PRO)
默认值	—
上下文	upstream

配置会话亲和性，以将客户端会话绑定到代理服务器，模式由第一个参数指定；要排空配置了 `sticky` 指令的服务器，可以在*server* 块中使用 `drain` 选项。

警告

`sticky` 指令必须在所有指定特定负载均衡方法的指令之后使用，否则将不起作用。如果与*bind_conn (PRO)* 指令一起使用，则 `bind_conn` 必须在 `sticky` 之后。

`cookie` 模式

此模式使用 cookie 来管理会话。适用于已经使用 cookie 进行会话跟踪的场景。

在应用任何粘性之前，第一个客户端请求会根据配置的负载均衡方法发送到后端服务器。然后 Angie 会设置一个标识所选服务器的 cookie。

cookie 名称（`name`）由 `sticky` 指令定义，其值对应于*sid* （来自*server* 指令）。如果设置了*sticky_secret*，此值会进一步进行哈希处理。

后续携带此 cookie 的请求会被路由到由其*sid* 指定的服务器。如果该服务器不可用或无法处理请求，则通过配置的负载均衡方法选择另一个服务器。

您可以在指令中分配 cookie 属性；默认情况下，仅设置 `path=/`。属性值可以包含变量。要删除某个属性，请将其设置为空值：`attr=`。例如，`sticky cookie path=` 会省略 `path` 属性。

此示例设置一个名为 `srv_id` 的 cookie，有效期为 1 小时，使用来自变量的域名：

```
upstream backend {
    zone backend 1m;
    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，这些标识符可能来自 URL、cookie 或请求参数。灵活性较低，但适用于此类标识符已经存在的情况。

后端服务器可能返回一个客户端和服务器都知道的路由 ID。此值必须与*sid* 匹配。

后续请求应携带路由 ID，例如通过*cookie* 或查询参数。

该指令接受一个变量列表来提取路由 ID。第一个非空值会与*sid* 进行匹配。

在此示例中，Angie 首先检查 route cookie，然后检查 route 查询参数：

```
upstream backend {
    zone backend 1m;
    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

此模式使用动态生成的密钥将客户端分配给后端。它很灵活，支持在共享内存中存储会话以及各种标识符来源。

会话从后端服务器的响应中创建。create 和 lookup 定义如何生成和定位会话，两者都接受多个变量。

会话 ID 是 create 中第一个非空变量。例如，它可能来自响应 cookie。

会话存储在共享内存中，通过 zone name:size 定义。如果在 timeout 时长（默认：1 小时）内未使用，会话将过期。

默认情况下，Angie 在每次使用时刷新会话。要禁用此功能，请使用 norefresh。

客户端请求中的会话 ID 通过 lookup 提取，使用列出的第一个非空变量。如果找不到，则为新请求。

使用 header 可在接收到响应头时创建会话，而不是在完整响应处理后。

示例：使用 examplecookie 创建会话：

```
upstream backend {
    zone backend 1m;
    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）

使用 `remote_action` 和 `remote_result` 通过外部会话存储管理会话 ID。共享内存区域充当缓存；外部存储具有权威性。`create` 与 `remote_action` 不兼容。

会话在 `timeout` （默认：1 小时）后过期，与 `remote_action` 无关。

默认情况下，Angie 在每次使用时刷新会话 TTL。使用 `norefresh` 可禁用此功能。

`zone` 在使用 `remote_action` 时是可选的。如果没有它，Angie 总是查询外部存储。

基本流程：

- 从第一个非空的 `lookup` 变量中提取会话 ID。如果没有，则回退到标准负载均衡。
- 如果设置了 `zone` 且会话存在，则使用它并停止。
- 如果没有会话或没有 zone，选择一个服务器并向 `remote_action` 端点发起 HTTP 子请求，包含：
  - 会话 ID（*$sticky_sessid*）；
  - 来自 `sid=` 或*$sticky_sid* 的服务器 ID。

  通过 HTTP 头（通过*proxy_set_header*）发送这些信息。
- 远程存储响应：
  - 200/201/204 确认会话；如果设置了 `zone` 则缓存它。
  - 409 表示冲突（如果设置了 `zone`）—会话链接到另一个服务器。使用 `remote_result` 提取更正后的服务器 ID。
  - 其他状态码或缺少服务器 ID —回退到原始服务器。

`remote_result` 使用 `upstream_http_*` 变量从远程存储的响应中读取头部。

示例：会话 ID 来自 `$cookie_bar`，通过 `$upstream_http_x_sticky_sid` 确认：

```
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;
        }
    }
}
```

以下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=web-server-01;
        server 10.0.0.2 sid=web-server-02;

        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/;
        }

        # Request to the remote session store
        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";
        }
    }
}
```

响应示例：

```
HTTP/1.1 200 OK
...
X-Sid: web-server-01
X-Session-Backend: backend-pool-1
```

生成的 Angie 变量：

- `$upstream_http_x_sid` → `web-server-01`
- `$upstream_http_x_session_backend` → `backend-pool-1`

`remote_result` 将使用 `web-server-01` 来选择匹配的 `sid`。

`sticky` 指令遵循上游服务器状态：

- 标记为 `down` 或失败的服务器会被排除。
- 超过 `max_conns` 限制的服务器会被跳过。
- `drain` 服务器在 `sticky` 模式下当标识符匹配时仍可能被选中用于新会话。
- 恢复的服务器会自动重新使用。

您可以使用*sticky_secret* 和*sticky_strict* 进一步调整行为。如果粘性失败且 `sticky_strict` 关闭，则使用回退负载均衡；如果开启，则拒绝请求。

`sticky` 中使用的每个 `zone` 必须专属于单个 `upstream`。区域不能在多个 `upstream` 块之间共享。

sticky_secret

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 `cookie` 和 `route` 模式下的*sticky* 指令。*string* 可以包含变量，例如 `$remote_addr`：

```
upstream backend {
    zone backend 1m;
    server backend1.example.com:8080;
    server backend2.example.com:8080;

    sticky cookie cookie_name;
    sticky_secret my_secret.$remote_addr;
}
```

盐值会附加到被哈希的值后面；要独立验证哈希机制：

```
$ echo -n "<VALUE><SALT>" | md5sum
```

sticky_strict

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，会导致 Angie 向客户端返回 HTTP 502 错误，而不是像上游中没有可用服务器时那样使用任何其他可用服务器。

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，可以混合使用监听 TCP 和 UNIX 域套接字的服务器。

示例：

```
upstream backend {
    zone backend 1m;
    server backend1.example.com weight=5;
    server 127.0.0.1:8080       max_fails=3 fail_timeout=30s;
    server backup1.example.com  backup;
}
```

默认情况下，请求使用加权轮询均衡方法在服务器之间分配。在上面的示例中，每 7 个请求将按如下方式分配:5 个请求发送到 backend1.example.com，第二个和第三个服务器各接收一个请求。该分配是平滑的:权重较高的服务器的请求会分散到整个轮询周期中，而不是集中成一批连续发送。

如果在与服务器通信期间发生错误，请求将被传递到下一个服务器，依此类推，直到尝试所有正常运行的服务器。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通信的结果。

备注

默认情况下，偶尔出现失败尝试但未达到*max_fails* 阈值的服务器会暂时收到较少的请求份额，并在后续请求中逐渐恢复到完整份额。这与*slow_start* 不同：后者仅在服务器被标记为不可用并随后恢复后，才使其逐步回到完整权重。

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。

备注

只有在配置的 `size` 不变时，重载才会保留区域内容。任何大小变更——增大或减小——都会导致区域被重新创建为空。

备注

仅在配置了此区域时才会收集上游指标。如未配置，该组将不会出现在*/status/http/upstreams/<upstream>*、「*HTTP Upstreams*」小部件 及*Prometheus* 输出中，且不会输出任何警告信息。参见配置示例。

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` 获取的初始会话 ID。

`$sticky_sid`

与*sticky* 中的 `remote_action` 一起使用；存储先前与会话关联的服务器 ID。

$upstream_addr

存储上游服务器的 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, 该变量保存服务器组 的 名称。

$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_key

Added in version 1.11.0.

包含用于请求的缓存键。

$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`: 在缓存中未找到响应, 请求被传递到上游服务器。
- `BYPASS`: 绕过缓存, 请求直接传递到上游服务器。
- `EXPIRED`: 缓存的响应已过期, 向上游服务器传递新请求以更新内容。
- `STALE`: 缓存的响应已过期, 但仍然提供给客户端, 直到最终从上游服务器更新内容。
- `UPDATING`: 缓存的响应已过期, 但仍然提供给客户端, 同时正在进行从上游服务器的更新。
- `REVALIDATED`: 缓存的响应已过期, 但已成功重新验证, 无需从上游服务器更新。
- `HIT`: 响应从缓存中获取。

如果请求绕过缓存而未访问它, 则不设置该变量。

`$upstream_connect_time`

保存与上游服务器建立连接所花费的时间; 时间以秒为单位保存, 具有毫秒分辨率。在 SSL 的情况下, 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>`

上游服务器在 `Set-Cookie` 响应头字段中发送的指定名称的 cookie。仅保存最后一个服务器响应中的 cookie。

`$upstream_current_addr` **(PRO)**

Added in version 1.12.0: PRO

为当前连接尝试所选择的被代理服务器的地址；与*$upstream_addr* 不同，后者会累积每次重试所尝试的每个服务器的地址,而该变量始终只保存单个地址,并且在服务器被选择后立即可用,包括在对*proxy_bind* 变量求值期间。

`$upstream_header_time`

保存从上游服务器接收响应头所花费的时间; 时间以秒为单位保存, 具有毫秒分辨率。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,:samp:*Server* 响应头字段可通过 `$upstream_http_server` 变量获取。将头字段名称转换为变量名称的规则与以 `$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`

保存请求在下一次服务器选择之前在队列 中花费的时间; 时间以秒为单位保存, 具有毫秒分辨率。多次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_request_method`

Added in version 1.11.0.

用于上游请求的请求方法。当缓存将 `HEAD` 转换为 `GET` 或设置了*proxy_method* 时, 它可能与客户端请求方法不同。

`$upstream_response_length`

保存从上游服务器获取的响应的长度; 长度以字节为单位保存。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

保存从上游服务器接收响应所花费的时间; 时间以秒为单位保存, 具有毫秒分辨率。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 Angie 为每次上游尝试记录的状态码。对于 DoH 请求，被截断的 UDP 响应记录为 426，已接受的 DNS 响应记录为 200。多次尝试的状态码用逗号和冒号分隔, 类似于*$upstream_addr* 变量中的地址。

$upstream_sticky_status

粘性请求的状态。

""	请求发送到未启用粘性的上游。
NEW	请求不包含粘性信息。
HIT	带有粘性信息的请求发送到所需的服务器。
MISS	带有粘性信息的请求发送到由负载均衡算法选择的服务器。

来自多个连接的状态用逗号和冒号分隔, 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_transport

Added in version 1.12.0.

保存每次上游连接尝试所使用的传输协议 TCP 或 UDP。多次尝试的值用逗号和冒号分隔，类似于*$upstream_addr* 变量中的地址。

Upstream Probe

该模块为*Upstream* 实现主动健康探测。

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)

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`];
默认值	—
上下文	location

为*upstream* 组中的服务器定义主动健康探测，这些上游组在与 `upstream_probe` 指令相同的 `location` 上下文中通过*proxy_pass*、*uwsgi_pass* 和类似指令指定。Angie 根据指定的参数定期向上游组中的每个服务器执行请求。

如果对服务器的请求成功，考虑到 `upstream_probe` 指令的所有参数设置以及控制其定义所在的 `location` 上下文如何使用上游的所有参数，则服务器通过探测。这包括*proxy_next_upstream* 和*uwsgi_next_upstream* 指令等，以及*proxy_set_header* 等。

要使用探测，上游必须具有共享内存区域（*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	要附加到*proxy_pass*、*uwsgi_pass* 等参数的请求 URI。默认为 /。
port	探测请求的备用端口号。
interval	探测之间的间隔。默认为 5s。
method	探测请求的 HTTP 方法。默认为 GET。
test	在请求期间要检查的条件；定义为包含变量的字符串。如果变量替换产生 "" 或 "0"，则探测失败。
essential	如果设置，服务器的初始状态需要验证，在通过探测之前不会将客户端请求转发给它。
persistent	设置此参数需要首先启用 essential；在配置重新加载 之前正常工作的 persistent 服务器无需首先通过此探测即可开始接收请求。
fails	使服务器变为不健康状态的连续失败请求数。默认为 1。
passes	使服务器变为健康状态的连续成功请求数。默认为 1。
max_body	响应正文的最大内存量。默认为 256k。
mode	探测模式，取决于服务器的健康状态： • always —无论服务器状态如何都进行探测； • idle —探测影响不健康的服务器以及自上次客户端请求以来已经过 interval 的服务器。 • onfail —仅探测不健康的服务器。 默认为 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，服务器在通过为其配置的 * 所有 * `essential` 探测之前不会接收客户端请求（如果配置已重新加载且服务器在此之前被认为是健康的，则跳过 `persistent` 探测）。如果没有此类探测，则认为服务器是健康的。
- 如果为服务器配置的 * 任何 * 探测达到其 `fails` 阈值，或者服务器本身达到*max_fails* 阈值，则认为服务器不健康且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，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* 期间接收的服务器响应正文；其大小受 `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:samp:*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:samp:*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:samp:*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` *address* [`transparent`] | `off`;
默认值	—
上下文	http, server, location

使到 uWSGI 服务器的出站连接源自指定的本地 IP 地址和可选端口。参数值可以包含变量。特殊值 `off` 取消从上一级配置继承的 *uwsgi_bind* 指令的效果，允许系统自动分配本地 IP 地址和端口。

transparent 参数允许到 uWSGI 服务器的出站连接源自非本地 IP 地址, 例如, 源自客户端的真实 IP 地址:

```
uwsgi_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 上不需要这样做, 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。

备注

需要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 *size*;
默认值	uwsgi_buffer_size 4k|8k;
上下文	http, server, location

设置用于读取从 uWSGI 服务器接收的响应的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。根据平台不同, 这可能是 4K 或 8K。但可以设置得更小。

uwsgi_buffering

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

启用或禁用来自 uWSGI 服务器的响应的缓冲。

on	Angie 尽快从 uWSGI 服务器接收响应, 将其保存到由*uwsgi_buffer_size* 和*uwsgi_buffers* 指令设置的缓冲区中。向客户端发送会并行进行: 已填满的缓冲区会被传递用于发送, 但其总大小受*uwsgi_busy_buffers_size* 限制。如果缓冲区未被完全填满, 则不会被传递用于发送, 除非它包含响应的最后一部分。因此, 当需要立即传递每几个字节时, 缓冲读取并不适合。如果整个响应无法放入内存, 则可以将其一部分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	响应在接收到时立即传递给客户端。Angie 以" 读取—发送" 的循环工作, 不会等待缓冲区完全填满: 例如, 从 4K 缓冲区读取的 10 字节会立即发送。同时, 如果整个响应可以放入缓冲区,Angie 可以完整读取它。Angie 一次可以从服务器接收的数据的最大大小由*uwsgi_buffer_size* 指令设置。使用 off 时,:ref:*uwsgi_limit_rate* 不生效。

也可以通过在 `X-Accel-Buffering` 响应头字段中传递" yes" 或" no" 来启用或禁用缓冲。可以使用*uwsgi_ignore_headers* 指令禁用此功能。

uwsgi_buffers

语法	`uwsgi_buffers` *number size*;
默认值	`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。根据平台不同, 这可能是 4K 或 8K。

uwsgi_busy_buffers_size

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

当启用来自 uWSGI 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向客户端发送响应的缓冲区的总大小。同时, 其余的缓冲区可用于读取响应, 如果需要, 还可以将部分响应缓冲到临时文件中。

默认情况下, 大小受*uwsgi_buffer_size* 和*uwsgi_buffers* 指令设置的两个缓冲区大小的限制。

uwsgi_cache

语法	`uwsgi_cache` *zone*	`off`;
默认值	`uwsgi_cache off;`
上下文	http, server, location

定义用于缓存的共享内存区域。同一区域可以在多个地方使用。参数值可以包含变量。

`off`	禁用从上一级配置继承的缓存。

备注

使用*uwsgi_cache* 时, 通常还需要设置*uwsgi_cache_valid* 指令来明确指定响应的缓存时间。如果未设置,Angie 不使用默认值, 而是根据来自服务器的 HTTP 响应头按以下优先级顺序确定响应缓存时间:

1. `X-Accel-Expires` 头 (最高优先级)。
2. 带有 `max-age` 或 `s-maxage` 参数的 `Cache-Control` 头。

3. Expires 头。

如果这些头都不包含有效值或根本不存在, 响应将不会被缓存, 因为无法确定其过期时间。

uwsgi_cache_background_update

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

允许启动后台子请求来更新过期的缓存项, 同时将过期的缓存响应返回给客户端。

警告

注意, 必须允许 在更新过期缓存响应时使用它。

uwsgi_cache_bypass

语法	uwsgi_cache_bypass ...;
默认值	—
上下文	http, server, location

定义不从缓存中获取响应的条件。如果字符串参数中至少有一个值不为空且不等于”0”, 则不会从缓存中获取响应:

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

参数值可以包含变量。

可以与*uwsgi_no_cache* 指令一起使用。

uwsgi_cache_key

语法	uwsgi_cache_key *string*;
默认值	—
上下文	http, server, location

定义缓存的键。键可以包含变量。例如

```
uwsgi_cache_key localhost:9000$request_uri;
```

uwsgi_cache_lock

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

启用后，一次只允许一个请求通过将请求传递给 uWSGI 服务器来填充根据*uwsgi_cache_key* 指令标识的新缓存元素。同一缓存元素的其他请求将等待响应出现在缓存中或此元素的缓存锁被释放，最长等待时间由*uwsgi_cache_lock_timeout* 指令设置。

uwsgi_cache_lock_age

语法	uwsgi_cache_lock_age *time*;
默认值	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一个请求在指定时间内未完成，则可以再传递一个请求给 uWSGI 服务器。

uwsgi_cache_lock_timeout

语法	uwsgi_cache_lock_timeout *time*;
默认值	uwsgi_cache_lock_timeout 5s;
上下文	http, server, location

设置*uwsgi_cache_lock* 的超时时间。当时间到期时，请求将被传递给 uWSGI 服务器，但是响应不会被缓存。

uwsgi_cache_max_range_offset

语法	uwsgi_cache_max_range_offset *number*;
默认值	—
上下文	http, server, location

为字节范围请求设置偏移量 (以字节为单位)。如果范围超出偏移量，范围请求将被传递到 uWSGI 服务器，并且响应不会被缓存。

uwsgi_cache_methods

语法	`uwsgi_cache_methods GET | HEAD | POST ...;`
默认值	`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" 方法始终会添加到列表中，但建议显式指定它们。另请参阅*uwsgi_no_cache* 指令。

uwsgi_cache_min_uses

语法	`uwsgi_cache_min_uses` *number*;
默认值	`uwsgi_cache_min_uses 1;`
上下文	http, server, location

设置在多少次请求后响应将被缓存。

警告

缓存元数据存储在共享内存中。手动删除缓存文件不会重置计数器，可能导致不可预测的行为。要完全重置缓存，请停止服务器，删除缓存目录，然后重新启动。

备注

第三方缓存清除模块（例如 external-cache-purge）仅删除文件，但不会重置 *uwsgi_cache_min_uses* 计数器。该指令旨在保护缓存免受不频繁请求的污染，在清除时重置计数器可能会对性能产生负面影响。

uwsgi_cache_path

语法	`uwsgi_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`	定义缓存的层次结构级别：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，然后重命名该文件。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，文件将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，建议对于任何给定位置，缓存和存放临时文件的目录都放在同一文件系统上。

`use_temp_path=on | off`	确定用于临时文件的目录
`on`	如果未指定该参数或设置为 `on`，将使用给定 *location* 的*uwsgi_temp_path* 指令指定的目录
`off`	临时文件将直接放置在缓存目录中
`keys_zone`	设置用于存储所有活动键和数据信息的共享内存区域的名称和大小。缓存元数据存储在共享内存中。 1 兆字节的区域足以存储约 8000 个键。 当使用 `keys_zone` 指定可选的 *file* 时，Angie 会在主进程终止时将该区域的内容转储到磁盘，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它，以确保更可靠的数据持久性并减少缓存加载时间。 如果由于区域大小更改、二进制版本不兼容或其他原因而无法恢复该区域，Angie 将记录警告（`failed to restore zone at address`）并且不会使用区域恢复机制。相反，不兼容的文件将被重命名为 `.old`；您可以删除它，或恢复其名称并将 Angie 还原到最初创建此文件的配置和版本。 **警告** 确保正确指定 *file* 的路径，并具有 Angie 使用所需的访问权限，并受到保护以防止未经授权的访问；相对路径基于 prefix。
`inactive`	如果在此参数指定的时间内未访问缓存数据，则无论其新鲜度如何，数据都将被删除；每次访问都会重置此计时器。这与下文所述的、由 ** 缓存管理器 ** 执行的基于大小的清除不同：即使缓存未达到 `max_size`，基于不活动的删除也会进行，并且会忽略由*uwsgi_cache_valid* 配置的有效期。 默认情况下，`inactive` 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缓存所在文件系统上的最小可用空间量，当超过大小或没有足够的可用空间时，它会删除最近最少使用的数据。数据以迭代方式删除。

max_size	缓存大小的最大阈值
min_free	缓存所在文件系统上可用空间的最小阈值
manager_files	一次迭代中删除的最大缓存项数 默认值:100
manager_threshol	限制一次迭代的时间 默认值:200 毫秒
manager_sleep	迭代之间保持暂停的时间 默认值:50 毫秒

Angie 启动一分钟后, 会激活一个特殊的 ** 缓存加载器 ** 进程, 该进程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。加载也以迭代方式进行。

loader_files	一次迭代中加载的最大缓存项数 默认值:100
loader_threshold	限制一次迭代的时间 默认值:200 毫秒
loader_sleep	迭代之间保持暂停的时间 默认值: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

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

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

确定在哪些情况下可以使用过期的缓存响应。该指令的参数与*uwsgi_next_upstream* 指令的参数匹配。

error	如果无法选择 uwsgi 服务器来处理请求, 则允许使用过期的缓存响应。
updating	附加参数, 如果当前正在更新过期的缓存响应, 则允许使用它。这可以最大限度地减少更新缓存数据时对 uwsgi 服务器的访问次数。

也可以直接在响应头中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:

- `Cache-Control` 头字段的 stale-while-revalidate 扩展允许在当前正在更新过期的缓存响应时使用它。
- `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

此方法的优先级低于设置指令参数。

为了在填充新缓存元素时最大限度地减少对 uwsgi 服务器的请求数, 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;
```

为代码 200 和 302 的响应设置 10 分钟的缓存, 为代码 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, 可以使用 `any` 参数指定缓存任何响应:

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

备注

缓存参数也可以直接在响应头中设置。此方法的优先级高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。值 *0* 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), 响应可以缓存到该时间。
- 如果头中不包含 `X-Accel-Expires` 字段, 则缓存参数由 `Expires` 或 `Cache-Control` 头字段确定。
- 带有 `Set-Cookie` 头字段的响应将不会被缓存。

- 带有特殊值"*" 的 `Vary` 头字段的响应将不会被缓存。带有其他值的 `Vary` 头字段的响应将在考虑相应请求头字段的情况下被缓存。

可以使用*uwsgi_ignore_headers* 指令禁用对这些头字段中的一个或多个的处理。

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时间。需要注意的是, 此超时时间通常不能超过 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

配置当代理服务器因重新解析 过程或*API* 命令 `DELETE` 而从组中移除或标记为永久不可用时, 是否终止到该服务器的所有连接。

当处理客户端或代理服务器的下一个读取或写入事件时, 连接将被终止。

设置 *time* 启用连接终止超时; 设置为 `on` 时, 连接将立即断开。

uwsgi_force_ranges

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

无论 uwsgi 服务器的响应中是否存在 `Accept-Ranges` 字段, 都为来自该服务器的缓存和非缓存响应启用字节范围支持。

uwsgi_hide_header

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

默认情况下,Angie 不会将 uwsgi 服务器响应中的头字段 `Date`、`Server`、`X-Pad` 和 `X-Accel-...` 传递给客户端。*uwsgi_hide_header* 指令设置不会被传递的其他字段。相反, 如果需要允许传递字段, 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

确定当客户端在未等待响应的情况下关闭连接时，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

禁用对来自 uwsgi 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，处理这些头字段将产生以下效果:

- `X-Accel-Expires`、`Expires`、`Cache-Control`、`Set-Cookie` 和 `Vary` 设置响应缓存 的参数;
- `X-Accel-Redirect` 执行到指定 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` 设置响应所需的字符集。

uwsgi_intercept_errors

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

确定状态码大于或等于 300 的 uwsgi 服务器响应应该传递给客户端，还是被拦截并重定向到 Angie 以使用*error_page* 指令进行处理。

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* 以每秒字节数指定; 可以使用变量。

0	禁用速率限制

备注

限制是按请求设置的, 因此如果 Angie 同时打开两个到 uwsgi 服务器的连接, 总速率将是指定限制的两倍。该限制仅在启用来自 uwsgi 服务器的响应缓冲 时有效。

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

当启用来自 uwsgi 服务器的响应缓冲 时, 如果整个响应不适合*uwsgi_buffer_size* 和*uwsgi_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*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_hide_header` *field*;
默认值	—
上下文	http, server, location

默认情况下,Angie 不会将 uwsgi 服务器响应中的头字段 `Date`、`Server`、`X-Pad` 和 `X-Accel-...` 传递给客户端。`uwsgi_hide_header` 指令设置不会被传递的其他字段。相反, 如果需要允许传递字段, 可以使用*uwsgi_pass_header* 指令。

uwsgi_modifier2

Syntax	`uwsgi_modifier2` *number*;
Default	`uwsgi_modifier2 0;`
Context	http, server, location

Sets the value of the modifier2 field in the uwsgi packet header.

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

指定在哪些情况下应将请求传递给*upstream* 组中的下一个服务器:

`error`	发生连接错误、请求传输错误或响应头读取错误;
`timeout`	在建立连接、传输请求或读取响应头期间发生超时;
`invalid_header`	服务器返回空响应或无效响应;
`http_500`	服务器返回代码为 500 的响应;
`http_503`	服务器返回代码为 503 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, 如果已经向上游服务器发送了请求, 则不会将使用 非幂等 方法 (`POST`、`LOCK`、`PATCH`) 的请求传递给另一个服务器; 启用此参数可显式允许重试此类请求;
`off`	禁用将响应缓冲到临时文件

备注

应该理解, 只有在尚未向客户端发送任何内容时, 才可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。

该指令还定义了什么被视为与服务器通信的不成功尝试。

error	始终被视为不成功尝试, 即使未在指令中指定
timeout	
invalid_header	
http_500	仅在指令中指定时才被视为不成功尝试
http_503	
http_429	
http_403	永远不会被视为不成功尝试
http_404	

将请求传递给下一个服务器可以通过尝试次数 和时间 进行限制。

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

限制可以将请求传递给下一个 服务器的时间。

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

定义不将响应保存到缓存的条件。如果字符串参数中至少有一个值不为空且不等于”0”, 则不会保存响应:

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

参数值可以包含变量。

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。该值可以包含文本、变量及其组合。当且仅当当前级别上没有定义 `uwsgi_param` 指令时，这些指令才从上一级配置继承。

标准 CGI 环境变量 应作为 uwsgi 头提供，请参阅发行版中提供的 `uwsgi_params` 文件：

```
location / {
    include uwsgi_params;
#    ...
}
```

如果使用 `if_not_empty` 指定指令，则仅当参数值不为空时才会将该参数传递给服务器：

```
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。作为协议，可以指定 `uwsgi` 或 `suwsgi` （安全 uwsgi，基于 SSL 的 uwsgi）。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;
```

或者指定为 UNIX 域套接字路径，在单词 `unix` 之后并用冒号括起来：

```
uwsgi_pass unix:/tmp/uwsgi.socket;
```

如果域名解析为多个地址，则所有地址都将以轮询方式使用。此外，地址可以指定为服务器组。

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。

备注

如果在带有尾部斜杠的前缀的 `location` 中指定了 `uwsgi_pass`（例如 `location /name/`），并且*auto_redirect* 指令设置为 `default`，则不带尾部斜杠的请求将被重定向（`/name -> /name/`）。

uwsgi_pass_header

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

允许将原本被禁用的 头字段从 uwsgi 服务器传递给客户端。

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。另请参阅*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

启用或禁用将原始请求的头字段传递给 uwsgi 服务器。另请参阅*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，而不是用于整个响应的传输。如果 uwsgi 服务器在此时间内未传输任何内容，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，从客户端完全读取 请求体。
`off`	请求体在接收时立即发送到 uwsgi 服务器。在这种情况下，如果 Angie 已经开始发送请求体，则请求无法传递到下一个服务器。

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。

uwsgi_send_timeout

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

设置向 uwsgi 服务器传输请求的超时时间。超时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 uwsgi 服务器在此时间内未接收任何内容，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

配置到 uwsgi 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`	为套接字启用 *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

指定一个 PEM 格式的证书文件，用于向安全的 uwsgi 服务器进行身份验证。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

定义一个缓存，用于存储使用变量指定的*SSL 证书* 和密钥。

该指令支持以下参数:

- `max` —设置缓存中的最大元素数量。当缓存溢出时，最近最少使用（LRU）的元素将被移除。
- `inactive` —定义元素在此时间内未被访问即会被移除的时长，即使其仍处于 `valid` 周期内；每次访问都会重置此计时器。默认为 10 秒。
- `valid` —定义缓存元素被视为有效并可重用的时间。默认为 60 秒。在此期间之后，证书将被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

指定一个 PEM 格式的密钥文件，用于向安全的 uwsgi 服务器进行身份验证。

可以指定值 `engine:`name`:id` 来代替文件，这将从 OpenSSL 引擎 name 中加载具有指定 id 的密钥。

可以指定值 `store:scheme:id` 来代替文件，用于从 OpenSSL 提供程序注册的 URI 方案（如 pkcs11）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

指定向安全的 uwsgi 服务器发送请求时启用的加密套件。加密套件以 OpenSSL 库理解的格式指定。

加密套件列表取决于安装的 OpenSSL 版本。可以使用 `openssl ciphers` 命令查看完整列表。

警告

使用 OpenSSL 时，`uwsgi_ssl_ciphers` 指令 不配置 TLS 1.3 的加密套件。要使用 OpenSSL 调整 TLS 1.3 加密套件，请使用*uwsgi_ssl_conf_command* 指令，该指令是为支持高级 SSL 配置而添加的。

- 在 LibreSSL 中，TLS 1.3 加密套件 可以使用 `uwsgi_ssl_ciphers` 配置。
- 在 BoringSSL 中，TLS 1.3 加密套件完全无法配置。

uwsgi_ssl_conf_command

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

在与安全的 uwsgi 服务器建立连接时设置任意 OpenSSL 配置 命令。

备注

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 加密套件，请使用 `ciphersuites` 命令。

可以在同一级别指定多个 *uwsgi_ssl_conf_command* 指令。当且仅当当前级别没有定义 *uwsgi_ssl_conf_command* 指令时，这些指令才会从上一级配置继承。

警告

请注意，直接重新配置 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

指定一个 PEM 格式的吊销证书（CRL）文件，用于验证 安全的 uwsgi 服务器的证书。

uwsgi_ssl_name

语法	`uwsgi_ssl_name` *name*;
默认值	`uwsgi_ssl_name ``uwsgi_pass 中的主机名;‘`
上下文	http, server, location

允许覆盖用于验证 安全的 uwsgi 服务器证书的服务器名称，以及在与安全的 uwsgi 服务器建立连接时通过 *SNI* 传递 的服务器名称。

服务器名称也可以用变量指定。

默认情况下，使用*uwsgi_pass* 指令中的主机名。

uwsgi_ssl_password_file

语法	`uwsgi_ssl_password_file` *file*;
默认值	—
上下文	http, server, location

指定一个包含密钥 口令的文件，每个口令单独占一行。加载密钥时依次尝试这些口令。

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

启用向安全的 uwsgi 服务器发送请求时使用的指定协议。

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）传递由*uwsgi_ssl_name* 指令设置的服务器名称。

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

确定在与安全 uwsgi 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。

uwsgi_ssl_trusted_certificate

语法	`uwsgi_ssl_trusted_certificate` *file*;
默认值	—
上下文	http, server, location

指定包含 PEM 格式受信任 CA 证书的文件，用于验证 安全的 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 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 uwsgi 服务器证书链的验证深度。

uwsgi_store

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

启用将文件保存到磁盘。

on	使用与*alias* 或*root* 指令对应的路径保存文件
off	禁用文件保存

可以使用带变量的 *string* 显式设置文件名：

```
uwsgi_store /data/www$original_uri;
```

文件的修改时间根据接收到的 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。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，而不是进行廉价的重命名操作。因此建议对于任何给定位置，保存的文件和由*uwsgi_temp_path* 指令设置的临时文件目录都放在同一文件系统上。

此指令可用于创建静态不可变文件的本地副本，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 ...*;
默认值	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*;
默认值	uwsgi_temp_file_write_size 8k|16k;
上下文	http, server, location

当启用将来自 uwsgi 服务器的响应缓冲到临时文件时，限制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*]]]‘;
默认值	uwsgi_temp_path uwsgi_temp; (路径取决于 构建参数 --http-uwsgi-temp-path)
上下文	http, server, location

定义用于存储从 uwsgi 服务器接收的数据的临时文件的目录。在指定目录下最多可以使用三级子目录层次结构。例如，在以下配置中

```
uwsgi_temp_path /spool/angie/uwsgi_temp 1 2;
```

临时文件可能如下所示:

```
/spool/angie/uwsgi_temp/7/45/00000123457
```

另请参阅*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，默认不构建此模块; 应使用 --with-http_v2_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。

配置示例

```
server {
    listen 443 ssl;

    http2 on;

    ssl_certificate server.crt;
```

```
    ssl_certificate_key server.key;
}
```

备注

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，该功能从 OpenSSL 1.0.2 版本开始提供。

如果*ssl_prefer_server_ciphers* 指令设置为值"on", 则应配置密码套件 以符合 RFC 9113 附录 A 黑名单，并且客户端需要支持。

指令

http2

语法	`http2 on	off;`
默认值	`http2 off;`
上下文	http, server

启用 HTTP/2 协议。

http2_body_preread_size

语法	`http2_body_preread_size` *size*;
默认值	—
上下文	http, server

设置每个请求的缓冲区大小, 在开始处理请求之前, 请求体可以保存在该缓冲区中。

http2_chunk_size

语法	`http2_chunk_size` *size*;
默认值	`http2_chunk_size 8k;`
上下文	http, server, location

设置响应体被切分成的块的最大大小。值过低会导致更高的开销。值过高会由于 队头阻塞 而影响优先级处理。

http2_max_concurrent_pushes

自 1.2.0 版本弃用.

语法	`http2_max_concurrent_pushes` *number*;
默认值	`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*;
默认值	`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`;
默认值	`http2_push off;`
上下文	http, server, location

抢先发送 (推送) 对指定 uri 的请求, 连同对原始请求的响应一起发送。只会处理带有绝对路径的相对 URI, 例如:

```
http2_push /static/css/main.css;
```

`uri` 值可以包含变量。

可以在同一配置级别指定多个 *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`;
默认值	`http2_push_preload off;`
上下文	http, server, location

启用将"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*;
默认值	`http2_recv_buffer_size 256k;`
上下文	http

设置每个工作进程 的输入缓冲区大小。

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`	表示基于 TLS 的 HTTP/2
`h2c`	表示基于明文 TCP 的 HTTP/2
`""`	否则为空字符串

HTTP/3

为客户端连接提供 HTTP/3 协议支持, 同时也支持与使用以下*Proxy* 模块指令配置的代理服务器的连接:

- *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, 此模块默认不会被构建; 需要使用 `--with-http_v3_module` 构建选项来启用。

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 / {
            # 用于通告 HTTP/3 的可用性
            add_header Alt-Svc 'h3=":8443"; ma=86400';
        }
    }
}
```

备注

请注意, 通过 TLS 接受 HTTP/3 连接需要 TLSv1.3 协议支持, 该协议从 OpenSSL 1.1.1 版本开始可用。

0-RTT 支持需要 OpenSSL 3.5.1 或更高版本。或者, 可以使用 BoringSSL、LibreSSL 或 QuicTLS 来构建和运行此模块。

在 1.29.1 版本之前, 无论*ssl_early_data* 指令的值如何, 都无法通过 OpenSSL 启用 0-RTT 支持。

对于 HTTP/3 请求, 如果未传递 `Host` 头, 则 `$http_host` 变量会从 `:authority` 伪头初始化。

此外,:samp:*reuseport* 选项只能在服务器上的一个 `listen ... quic` 指令中指定。所有其他 `listen ... quic` 指令必须不带此选项。

指令

http3

语法	`http3 on | off;`
默认值	`http3 on;`
上下文	http, server

启用 HTTP/3 协议协商。

http3_hq

语法	`http3_hq on | off;`
默认值	`http3_hq off;`
上下文	http, server

启用在 QUIC 互操作性测试 中使用的 HTTP/0.9 协议协商。

警告

仅在运行明确需要此模式的专门测试时才启用。

http3_max_concurrent_streams

语法	`http3_max_concurrent_streams` *number*;
默认值	`http3_max_concurrent_streams 128;`
上下文	http, server

初始化 HTTP/3 和 QUIC 设置, 并设置一个连接中并发 HTTP/3 请求流的最大数量。

http3_max_table_capacity

语法	`http3_max_table_capacity` *number*;
默认值	`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* 指令用于代理连接。为避免错误, 当启用带缓存的代理时, 动态表使用会被禁用。

http3_stream_buffer_size

语法	`http3_stream_buffer_size` *size*;
默认值	`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*;
默认值	`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;`
默认值	`quic_bpf off;`
上下文	main

启用使用 eBPF 路由 QUIC 数据包。启用后, 可以支持 QUIC 连接迁移。

备注

该指令仅在 Linux 5.7+ 上支持。

quic_gso

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

启用使用分段卸载的优化批量发送模式。

备注

优化发送仅在支持 `UDP_SEGMENT` 的 Linux 上受支持。

quic_host_key

语法	`quic_host_key` *file*;
默认值	—
上下文	http, server

设置包含用于加密无状态重置和地址验证令牌的密钥的 *file* 文件。默认情况下，每次重新加载时都会生成一个随机密钥。使用旧密钥生成的令牌不会被接受。

quic_retry

语法	`quic_retry on | off;`
默认值	`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，以及验证在 *Initial* 数据包中收到的令牌。

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。

备注

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、配置用于请求路由的位置、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标头。

本节中的其他模块扩展了此功能，允许您针对各种场景和需求灵活配置和优化 HTTP 服务器。

指令

absolute_redirect

语法	`absolute_redirect on	off;`
默认值	`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*];
默认值	`aio off;`
上下文	http, server, location

在 FreeBSD 和 Linux 上启用或禁用异步文件 I/O (AIO) 的使用:

```
location /video/ {
  aio            on;
  output_buffers 1 64k;
}
```

在 FreeBSD 上, 从 FreeBSD 4.3 开始可以使用 AIO。在 FreeBSD 11.0 之前,AIO 可以静态链接到内核中:

```
options VFS_AIO
```

或作为内核可加载模块动态加载:

```
kldload aio
```

在 Linux 上, 从内核版本 2.6.22 开始可以使用 AIO。此外, 还需要启用*directio*, 否则读取将会阻塞:

```
location /video/ {
  aio            on;
  directio       512;
  output_buffers 1 128k;
}
```

在 Linux 上,:ref:*directio* 只能用于读取按 512 字节边界对齐的块 (对于 XFS 为 4K)。文件的未对齐末尾以阻塞模式读取。对于字节范围请求和非从文件开头开始的 FLV 请求也是如此: 在文件开头和末尾读取未对齐的数据将会阻塞。

当在 Linux 上同时启用 AIO 和*sendfile* 时, 对于大于或等于*directio* 指令中指定大小的文件使用 AIO, 而对于较小的文件或禁用*directio* 时使用*sendfile*:

```
location /video/ {
  sendfile       on;
  aio            on;
  directio       8m;
}
```

最后, 可以使用多线程读取和发送 文件, 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池 的线程。如果省略池名称, 则使用名为"default" 的池。池名称也可以使用变量设置:

```
aio threads=pool$disk;
```

使用 `aio on` 需要使用 `--with-file-aio` 配置参数进行构建。使用 `aio threads` 需要使用 `--with-threads` 参数进行构建。

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 上受支持。

另请参阅*sendfile* 指令。

aio_write

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

如果启用了*aio*, 指定是否将其用于写入文件。目前, 这仅在使用 `aio threads` 时有效, 并且仅限于写入包含从代理服务器接收的数据的临时文件。

alias

语法	`alias` *path*;
默认值	—
上下文	location

为指定的位置定义替换路径。例如，使用以下配置：

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

对于 `/i/top.gif` 的请求，将发送文件 */data/w3/images/top.gif*。

path 值可以包含变量，但不包括*$document_root* 和*$realpath_root*。

如果在使用正则表达式定义的位置内使用 `alias`，则该正则表达式应包含捕获组，并且 `alias` 应引用这些捕获组，例如：

```
location ~ ^/users/(.+\.(?:gif|jpe?g|png))$ {
  alias /data/w3/images/$1;
}
```

当位置匹配指令值的最后部分时：

```
location /images/ {
  alias /data/w3/images/;
}
```

最好使用*root* 指令：

```
location /images/ {
  root /data/w3;
}
```

小心

使用 `alias` 时，匹配的 location 前缀会按其长度从 URI 中剥除，且前缀本身不再经过重新验证。location 名称末尾斜杠与 `alias` 值末尾斜杠必须保持一致。如果 `alias` 值缺少 location 所带的末尾斜杠，剩余部分会被直接拼接（例如 `/data/w3/imagestop.gif`），从而返回错误的文件；反之则只是多出一个斜杠，通常无害。对于匹配路径最后一部分的 location，建议使用*root* 指令，它会将完整 URI 追加到其值，不受此影响。

auth_delay

语法	`auth_delay` *time*;
默认值	`auth_delay 0s;`
上下文	http, server, location

延迟处理返回 401 响应代码的未授权请求，以防止在通过密码 或子请求结果 限制访问时的时序攻击。

auto_redirect

语法	auto_redirect [on | off | default];
默认值	auto_redirect default;
上下文	http, server, location

控制当前缀位置以斜杠结尾时的重定向 行为:

```
location /prefix/ {
    auto_redirect on;
}
```

在这里, 对 /prefix 的请求会导致重定向到 /prefix/。

值 on 显式启用重定向, 而 off 禁用它。当设置为 default 时, 仅当位置使用*api*、*proxy_pass*、*fastcgi_pass*、*uwsgi_pass*、*scgi_pass*、*memcached_pass* 或*grpc_pass* 处理请求时才启用重定向。

chunked_transfer_encoding

语法	chunked_transfer_encoding on | off;
默认值	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。当使用的软件尽管标准要求但不支持分块编码时, 这可能会派上用场。

client

Added in version 1.10.0.

在 1.10.1 版本发生变更.

语法	client { ... }
默认值	—
上下文	http

创建一个特殊的 client 上下文, 用于处理 Angie 自行执行的内部 HTTP 请求, 而无需外部客户端参与。

client 上下文将来自各种 Angie 模块的服务流量与用户流量隔离, 允许对其进行额外控制。在此上下文中, 只能定义命名位置 (带有 @ 前缀); 它们无法被外部 HTTP 请求访问, 只能通过内部服务器机制以编程方式调用。

client 上下文用于:

- 在*ACME* 模块中通过预定义的 location @acme 向证书颁发机构发送请求, 可以使用代理 模块的指令进行额外配置;

- 在*Docker* 模块中通过预定义的 `location @docker_events` 和 `@docker_containers` 向 Docker API 发送请求, 可以使用代理 模块的指令进行额外配置;
- 通过*upstream_probe (PRO)* 对代理服务器进行健康探测;
- 在流*Upstream* 模块中使用 `remote_action` 的*sticky learn* 模式。

支持多个 `client` 块允许在每个块内为多个 `location` 块分组通用设置, 这有助于避免配置重复。

在每个 `client` 块中指定的指令仅由在其中显式声明的 `location` 块继承。特别是, 这就是为什么它们不会影响隐式使用 `client` 块进行出站请求的其他模块的配置 (例如*ACME* 或*Docker*)。

使用多个 `client` 块并继承设置的示例:

```
client {

    proxy_set_header Host docker.example.com;
    proxy_set_header Authorization "Basic QWxhZGRpbjpvcGVuIHNlc2FtZQ==";

    location @docker_events {

    }

    location @docker_containers {

    }
}

client {

    proxy_method GET;
    proxy_set_header Host backend.example.com;
    proxy_set_header X-Real-IP $remote_addr;

    location @health_check {

        proxy_pass http://upstream-server/health;
    }
}
```

备注

这里允许使用与常规 `location` 块中相同的指令, 但实际上只有内容处理器 (如 js_content 或*autoindex*) 和变量处理器 (如*map*), 以及自身生成请求的指令 (如 `upstream_probe`) 才会生效。

在其他请求处理阶段 操作的指令 (如*limit_req*、*auth_request*、*try_files*、图像过滤器、XSLT 等) 在这里不起作用。

client_body_buffer_size

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

设置读取客户端请求体的缓冲区大小。如果请求体大于缓冲区，则将整个请求体或仅其部分写入临时文件。默认情况下，缓冲区大小等于两个内存页。在 x86、其他 32 位平台和 x86-64 上，这是 8K。在其他 64 位平台上，通常是 16K。

client_body_in_file_only

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

确定是否将整个客户端请求体保存到文件中。此指令可在调试期间使用，或在使用*$request_body_file* 变量或*Perl* 模块的*$r->request_body_file* 方法时使用。

`on`	请求处理后不删除临时文件
`clean`	允许删除请求处理后留下的临时文件

client_body_in_single_buffer

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

确定是否将整个客户端请求体保存在单个缓冲区中。建议在使用*$request_body* 变量时使用此指令，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*]]];
默认值	`client_body_temp_path client_body_temp;` (路径取决于构建选项 `--http-client-body-temp-path`)
上下文	http, server, location

定义用于存储客户端请求体临时文件的目录。在指定目录下最多可以使用三级子目录层次结构。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*;
默认值	`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, 则请求将以 408 (Request Time-out) 错误终止。

client_header_buffer_size

语法	`client_header_buffer_size` *size*;
默认值	`client_header_buffer_size 1k;`
上下文	http, server

设置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。但是, 如果请求包含长 cookie, 或来自 WAP 客户端, 则可能无法容纳在 1K 中。如果请求行或请求头字段无法容纳在此缓冲区中, 则会分配由*large_client_header_buffers* 指令配置的更大缓冲区。

如果在*server* 级别指定该指令, 则可以使用默认服务器的值。详情请参阅虚拟服务器选择 部分。

client_header_timeout

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

定义读取客户端请求头的超时时间。如果客户端在此时间内未传输整个请求头, 则请求将以 408 (Request Time-out) 错误终止。

client_max_body_size

语法	`client_max_body_size` *size*;
默认值	`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的值, 则向客户端返回 413 (Request Entity Too Large) 错误。请注意, 浏览器无法正确显示此错误。

0	禁用检查客户端请求体大小

connection_pool_size

语法	`connection_pool_size` *size*;
默认值	`connection_pool_size 256 | 512;`
上下文	http, server, location

允许精确调整每个连接的内存分配。此指令对性能的影响很小, 通常不应使用。默认情况下:

256 (字节)	在 32 位平台上
512 (字节)	在 64 位平台上

default_type

语法	`default_type` mime-*type*;
默认值	`default_type text/plain;`
上下文	http, server, location

定义响应的默认 MIME 类型。文件扩展名到 MIME 类型的映射可以使用*types* 指令设置。

directio

语法	`directio` *size* | `off`;
默认值	`directio off;`
上下文	http, server, location

在读取大于或等于指定大小的文件时, 启用 `O_DIRECT` 标志 (FreeBSD、Linux)、`F_NOCACHE` 标志 (macOS) 或 `directio()` 函数 (Solaris) 的使用。该指令会自动禁用给定请求的*sendfile* 使用。建议用于提供大文件:

```
directio 4m;
```

或在 Linux 上使用*aio* 时。

directio_alignment

语法	`directio_alignment` *size*;
默认值	`directio_alignment 512;`
上下文	http, server, location

设置*directio* 的对齐方式。在大多数情况下,512 字节对齐就足够了。但是, 在 Linux 下使用 XFS 时, 需要将其增加到 4K。

disable_symlinks

语法	`disable_symlinks off;` `disable_symlinks on | if_not_owner` [`from=`*part*];
默认值	`disable_symlinks off;`
上下文	http, server, location

确定在打开文件时应如何处理符号链接:

`off`	允许路径中的符号链接且不进行检查。这是默认行为。
`on`	如果路径的任何组成部分是符号链接, 则拒绝访问该文件。
`if_not_owner`	如果路径的任何组成部分是符号链接, 且该链接与其指向的对象具有不同的所有者, 则拒绝访问该文件。
`from=`*part*	在检查符号链接时 (参数 `on` 和 `if_not_owner`), 通常会检查所有路径组成部分。可以通过额外指定 `from=part` 参数来跳过对路径初始部分中符号链接的检查。在这种情况下, 仅从指定初始部分之后的路径组成部分开始检查符号链接。如果该值不是被检查路径的初始部分, 则会完整检查路径, 就像根本没有指定此参数一样。如果该值与文件名完全匹配, 则不检查符号链接。参数值中可以使用变量。

示例:

```
disable_symlinks on from=$document_root;
```

此指令仅在具有 `openat()` 和 `fstatat()` 接口的系统上可用。此类系统包括现代版本的 FreeBSD、Linux 和 Solaris。

警告

`on` 和 `if_not_owner` 参数会增加处理开销。

在不支持仅为搜索而打开目录的系统上, 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* 模块目前忽略此指令。

early_hints

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

定义将"103 Early Hints" 响应传递给客户端的条件。如果字符串参数的至少一个值不为空且不等于 0, 则将传递响应:

```
map $http_sec_fetch_mode $early_hints {
    navigate $http2$http3;
}

server {
    ...
    location / {
        early_hints $early_hints;
        proxy_pass http://example.com;
    }
}
```

参数值可以包含变量。

error_log_user_tag

Added in version 1.12.0.

语法	`error_log_user_tag` *value*;
默认值	—
上下文	http, server, location, limit_except

为*error_log* 记录添加特定于请求的标签。*value* 中可以包含变量。可以多次指定该指令以添加多个标签。可以在*error_log* 中使用 `filter=tag:` 匹配标签。

error_page

语法	`error_page` *code* ... [=[*response*]] *uri*;
默认值	—
上下文	http, server, location, if in location

定义针对指定错误显示的 URI。*uri* 值可以使用变量。

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)。

此外, 可以使用类似 `=response` 的语法将响应代码更改为另一个代码, 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), 则可以传递它返回的代码:

```
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的 `location`:

```
location / {
  error_page 404 = @fallback;
}

location @fallback {
  proxy_pass http://backend;
}
```

备注

如果在处理 uri 期间发生错误, 则会将最后发生的错误代码的响应返回给客户端。

也可以使用 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, 默认情况下会向客户端返回响应代码 302。它只能更改为重定向响应代码之一 (301、302、303、307 和 308)。

etag

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

启用或禁用为静态资源自动生成 ETag 响应头字段。

http

语法	http { ... }
默认值	—
上下文	main

提供配置文件上下文, 在其中指定 HTTP 服务器指令。

if_modified_since

语法	if_modified_since off | exact | before;
默认值	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;
默认值	ignore_invalid_headers on;
上下文	http, server

控制 Angie 是否忽略具有无效名称的头字段。有效名称由英文字母、数字、连字符组成, 可能还包括下划线 (由*underscores_in_headers* 指令控制)。

如果在*server* 级别指定该指令, 则可以使用默认服务器的值。

internal

语法	internal;
默认值	—
上下文	location

指定给定的 location 只能用于内部请求。对于外部请求, 将向客户端返回 404 (Not Found) 错误。内部请求包括以下几种:

- 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*SSI* 模块的 `include virtual` 命令、*Addition* 模块指令以及*auth_request* 和*mirror* 指令形成的子请求;
- 由*rewrite* 指令更改的请求;
- 由*goto* 指令重定向的请求。

示例:

```
error_page 404 /404.html;

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

由于 404 错误是在带有 `internal` 指令的 `location` 上下文中返回的, 外部请求可以重定向到不同的 location。这允许对外部和内部请求使用相同的前缀, 但进行不同的处理, 例如:

```
location /path {

    internal;
    error_page 404 =@external;

    proxy_pass https://internal;
}

location @external {

    proxy_pass https://external;
}
```

在这里, 外部请求 `GET /path` 将被代理到 `https://external/path`, 而相同的内部请求将被代理到 `https://internal/path`。

备注

为了防止错误配置可能导致的循环, 内部重定向的次数限制为十次。当达到此限制时, 将返回 500 (Internal Server Error) 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

禁用与行为异常的浏览器的保持连接。*browser* 参数指定将受影响的浏览器。

`none`	对所有浏览器启用保持连接
`msie6`	一旦收到 POST 请求，禁用与旧版本 MSIE 的保持连接
`safari`	禁用与 macOS 和类 macOS 操作系统上的 Safari 和类 Safari 浏览器的保持连接

keepalive_requests

语法	`keepalive_requests` *number*;
默认值	`keepalive_requests 1000;`
上下文	http, server, location

设置通过一个保持连接可以处理的最大请求数。在达到最大请求数后，连接将被关闭。

定期关闭连接对于释放每个连接的内存分配是必要的。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

限制通过一个保持连接处理请求的最长时间。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* [*header_timeout*];
默认值	`keepalive_timeout 75s;`
上下文	http, server, location

timeout	设置保持连接的客户端连接在服务器端保持打开的超时时间
0	禁用保持连接的客户端连接

第二个 可选参数在响应中设置 `Keep-Alive: timeout=time` 头字段的值。这两个参数可以不同。

`Keep-Alive: timeout=time` 头字段被 Mozilla 和 Konqueror 识别。MSIE 会在大约 60 秒后自行关闭保持连接。

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

设置用于读取大型客户端请求头的缓冲区的最大数量和大小。请求行不能超过一个缓冲区的大小, 否则将向客户端返回 414 (Request-URI Too Large) 错误。请求头字段也不能超过一个缓冲区的大小, 否则将向客户端返回 400 (Bad Request) 错误。缓冲区仅在需要时分配。默认情况下, 缓冲区大小等于 8K 字节。如果在请求处理结束后连接转换为保持连接状态, 这些缓冲区将被释放。

如果在*server* 级别指定该指令, 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 location 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` 方法也会使 `HEAD` 方法被允许。可以使用*Access* 和*Auth Basic* 模块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 **除** `GET` 和 `HEAD` **之外**的所有方法。

limit_rate

语法	`limit_rate` *rate*;
默认值	`limit_rate 0;`
上下文	http, server, location, if in location

限制向客户端传输响应的速率。速率以每秒字节数指定。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, 总速率将是指定限制的两倍。

参数值可以包含变量。在需要根据特定条件限制速率的情况下, 这可能很有用:

```
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, 但不建议使用此方法:

```
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

设置初始数量, 在此之后向客户端传输响应的速率将受到限制。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`	Angie 将等待 并处理 来自客户端的额外数据, 然后完全关闭连接, 但仅当启发式方法表明客户端可能正在发送更多数据时。
`always`	Angie 将始终等待并处理额外的客户端数据。
`off`	Angie 不会等待更多数据, 将立即关闭连接。此行为会破坏协议, 在正常情况下不应使用。

要控制 HTTP/2 连接的关闭, 必须在*server* 级别指定该指令。

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 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, 连接将被关闭, 即使还有更多数据。

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, 连接将被关闭。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, 客户端保持连接仅在空闲时间至少达到 `lingering_timeout` 中指定的时间时才会关闭。

备注

在 nginx 中, 类似的指令称为 keepalive_min_timeout。

listen

在 1.10.0 版本发生变更.

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*]:[*keepintvl*]:[*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*]:[*keepintvl*]:[*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

设置监听套接字的 *address* 和 *port*，或服务器接受请求的 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 地址用方括号指定：

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

端口范围通过用连字符分隔的第一个和最后一个端口来指定：

```
listen 127.0.0.1:8000-8009;
listen 8000-8009;
```

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*，或仅指定 *address* 或仅指定 *port*。当省略某些部分时，适用以下规则：

- 如果仅给出 *address*，则使用端口 80。
- 如果仅给出 *port*，Angie 将监听所有可用的 IPv4（以及 IPv6，如果已启用）接口。该端口的第一个 `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`	指定了此参数的服务器将成为给定 *address:port* 对的默认服务器（它们共同构成一个 监听套接字）。 如果没有带 `default_server` 参数的指令，该监听套接字的默认服务器将是配置中为该套接字提供服务的第一个服务器。
`ssl`	表示在此监听套接字上接受的所有连接都应在 SSL 模式下工作。这允许为同时处理 HTTP 和 HTTPS 请求的服务器提供更紧凑的配置。
`http2`	配置端口接受 HTTP/2 连接。通常，为了使其工作，还应指定 `ssl` 参数，但 Angie 也可以配置为接受不带 SSL 的 HTTP/2 连接。 自 1.2.0 版本弃用. 请改用*http2* 指令。
`quic`	配置端口接受 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`	表示在此监听套接字上接受的所有连接都应使用 PROXY 协议。

`listen` 指令还可以指定几个特定于套接字相关系统调用的附加参数。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*	为监听套接字设置路由表 FIB（`SO_SETFIB` 选项）。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。 **警告** 除非服务器能够处理多次接收带有数据的相同 SYN 数据包，否则不要启用"TCP Fast Open"。
`backlog=`*number*	设置 `listen()` 调用中的 `backlog` 参数，该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*	为监听套接字设置接收缓冲区大小（`SO_RCVBUF` 选项）。
`sndbuf=`*size*	为监听套接字设置发送缓冲区大小（`SO_SNDBUF` 选项）。
`accept_filter=`*filt*	为监听套接字设置接受过滤器的名称（`SO_ACCEPTFILTER` 选项），该过滤器在将传入连接传递给 `accept()` 之前对其进行过滤。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 `accept()`（`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，因为如果有多个具有相同端口但不同地址的 `listen` 指令，并且其中一个 `listen` 指令监听给定 `port` 的所有地址（`*:port`），Angie 将只 `bind()` 到 `*:port`。应该注意的是，在这种情况下将进行 `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，那么对于给定的 `address:port` 对将始终进行单独的 `bind()` 调用。
`ipv6only=on | off`	确定（通过 `IPV6_V6ONLY` 套接字选项）监听通配符地址 [::] 的 IPv6 套接字是否只接受 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`	指示为每个工作进程创建单独的监听套接字(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。目前仅在 Linux 3.9+、DragonFly BSD 和 FreeBSD 12+ 上有效。 **警告** 不当使用 `reuseport` 参数可能会带来安全隐患。
`multipath`	启用通过 Multipath TCP (MPTCP) 接受连接，自 Linux 内核版本 5.6 起支持。此参数与 `quic` **不兼容**。
`so_keepalive=on | off | [keepidle]:[keepint`	为监听套接字配置"TCP keepalive" 行为。 `''` 如果省略此参数，则套接字将使用操作系统的设置 `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 keepalive 参数。在这些系统上（目前包括 Linux、NetBSD、Dragonfly、FreeBSD 和 macOS），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
```

将空闲超时（TCP_KEEPIDLE）设置为 30 分钟，将探测间隔（TCP_KEEPINTVL）保留为系统默认值，并将探测次数（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)+ { ... }
默认值	—
上下文	server, location

根据请求 URI 是否匹配任何匹配表达式来设置配置。

匹配是针对规范化的请求 URI（即*$uri* 的值）执行的，在解码以"%XX" 形式编码的文本、解析对相对路径组件"." 和".." 的引用，以及可能将两个或多个相邻斜杠压缩 为单个斜杠之后进行。

location 可以由前缀字符串定义，也可以由正则表达式定义。

正则表达式使用前置修饰符指定：

~*	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 location，Angie 首先检查使用前缀字符串定义的 location（前缀 location）。在这些 location 中，选择并记住具有最长匹配前缀的 location。

备注

对于不区分大小写的操作系统（如 macOS），前缀字符串匹配不区分大小写。但是，匹配仅限于单字节区域设置。

然后按照正则表达式在配置文件中出现的顺序检查正则表达式。在第一次匹配后停止搜索，并使用相应的配置。如果没有找到与正则表达式的匹配，则使用之前记住的前缀 location 的配置。

除了下面提到的一些例外情况，location 块可以嵌套。

正则表达式可以创建捕获组，这些捕获组稍后可以与其他指令一起使用。

如果最长匹配前缀 location 具有 `^~` 修饰符，则不检查正则表达式。

此外，使用 `=` 修饰符，可以定义 URI 和 location 的精确匹配。如果找到精确匹配，则搜索终止。例如，如果 `/` 请求频繁发生，定义 `location =/` 将加快这些请求的处理速度，因为搜索在第一次比较后终止。这样的 location 不能包含嵌套的 location，因为它定义了精确匹配。

一旦匹配到 `location`，请求将通过*root* 或*alias* 指令映射到文件，*try_files* 可在其中探测候选文件；概述请参阅选择位置。

示例：

```
location =/ {
    #configuration A
}

location / {
    #configuration B
}

location /documents/ {
    #configuration C
}

location ^~/images/ {
    #configuration D
}

location ~*\.(gif|jpg|jpeg)$ {
    #configuration E
}
```

- `/` 请求将匹配配置 A，
- `/index.html` 请求将匹配配置 B，
- `/documents/document.html` 请求将匹配配置 C，
- `/images/1.gif` 请求将匹配配置 D，
- `/documents/1.jpg` 请求将匹配配置 E。

备注

如果前缀 `location` 以斜杠字符结尾且*auto_redirect* 已启用，则会发生以下情况：当请求到达时，其 URI 没有尾部斜杠但在其他方面与前缀完全匹配，则会返回代码为 301 的永久重定向，指向附加了斜杠的请求 URI。

对于精确 URI 匹配的 location，不会应用重定向：

```
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 location。此类 location 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 location。

组合位置

多个定义相同配置块的 location 上下文可以通过在单个 location 中列出所有匹配表达式并使用单个配置块来压缩。这称为 * 组合 * location。

假设前面示例中的配置 A、D 和 E 定义了相同的配置; 您可以将它们组合成一个 location:

```
location =/
         ^~/images/
         ~*\.(gif|jpg|jpeg)$ {
   # 通用配置
}
```

命名的 location 也可以是组合的一部分:

```
location =/
         @named_combined {
   #...
}
```

警告

组合 location 的匹配表达式修饰符和表达式本身之间不能有空格。正确形式: location ~*/match(ing|es|er)$ ...。

备注

目前, 组合 location 不能 ** 直接 ** 包含设置了 URI 的 proxy_pass 指令, 也不能包含 api 或 alias。但是, 这些指令可以在嵌套于组合位置内的位置中使用。

log_not_found

语法	`log_not_found on	off;`
默认值	`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* 中。

log_subrequest

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

启用或禁用将子请求记录到*access_log* 中。

max_headers

语法	`max_headers` *number*;
默认值	`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。

当此指令在*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*;
默认值	—
上下文	http, server, location

限制字节范围请求中允许的最大范围数量。超过限制的请求将被视为未指定字节范围。默认情况下，范围数量不受限制。

0	完全禁用字节范围支持

merge_slashes

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

启用或禁用将 URI 中两个或多个相邻斜杠压缩为单个斜杠。

请注意，压缩对于正确匹配前缀字符串和正则表达式位置至关重要。如果没有压缩,:samp:*// scripts/ one.php* 请求将不会匹配

```
location /scripts/ { }
```

并且可能被作为静态文件处理。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。但是, 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;
默认值	msie_padding on;
上下文	http, server, location

启用或禁用为状态大于 400 的 MSIE 客户端响应添加注释, 以将响应大小增加到 512 字节。

msie_refresh

语法	msie_refresh on | off;
默认值	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*];
默认值	open_file_cache off;
上下文	http, server, location

配置可以存储以下内容的缓存:

- 打开的文件描述符、它们的大小和修改时间;
- 目录存在性信息;
- 文件查找错误, 例如” 文件未找到”、” 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; 缓存溢出时, 将删除最近最少使用 (LRU) 的元素
`inactive`	定义一个时间, 如果在此时间内未访问元素, 则将其从缓存中删除; 默认为 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;`
默认值	`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_events

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

启用使用内核事件验证*open_file_cache* 缓存项。本指令仅适用于*kqueue* 方法。请注意, 只有 NetBSD 2.0+ 和 FreeBSD 6.0+ 支持任意文件系统类型的事件; 其他操作系统仅对 UFS 或 FFS 等基本文件系统支持事件。

open_file_cache_min_uses

语法	`open_file_cache_min_uses` *number*`;`
默认值	`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, 文件描述符在缓存中保持打开所需的最小文件访问次数。

open_file_cache_valid

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

设置应验证*open_file_cache* 元素的时间。

output_buffers

语法	`output_buffers` *number size*;
默认值	`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数量和大小。

port_in_redirect

语法	`port_in_redirect on` | `off`;
默认值	`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。

在重定向中使用主服务器名称由*server_name_in_redirect* 指令控制。

postpone_output

语法	`postpone_output` *size*;
默认值	`postpone_output 1460;`
上下文	http, server, location

如果可能, 客户端数据的传输将被推迟, 直到 Angie 至少有指定数量的字节要发送。

0	禁用推迟数据传输

read_ahead

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

设置在处理文件时内核的预读取量。

在 Linux 上, 使用 `posix_fadvise(0, 0, 0, POSIX_FADV_SEQUENTIAL)` 系统调用, 因此 size 参数会被忽略。

在 FreeBSD 上, 使用 `fcntl(O_READAHEAD,` size *)* 系统调用, 该调用从 FreeBSD 9.0-CURRENT 开始支持。

recursive_error_pages

语法	`recursive_error_pages on | off;`
默认值	`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。此类重定向的次数是有限的。

request_pool_size

语法	`request_pool_size` *size*;
默认值	`request_pool_size 4k;`
上下文	http, server

允许精确调整每个请求的内存分配。此指令对性能的影响很小, 通常不应使用。

reset_timedout_connection

语法	`reset_timedout_connection on | off;`
默认值	`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。重置按如下方式执行。在关闭套接字之前, 为其设置 `SO_LINGER` 选项, 超时值为 0。当套接字关闭时,TCP RST 被发送到客户端, 并释放与此套接字关联的所有内存。这有助于避免长时间保持已关闭的套接字处于 FIN_WAIT1 状态且缓冲区已满。

备注

keep-alive 连接在超时时会正常关闭。

resolver

在 1.12.0 版本发生变更.

语法	`resolver` [*address* ...] [`conf` [=*file*]] [`valid=`*time*] [`ipv4=on` | `off`] [`ipv6=on` | `off`] [`status_zone=`*zone*]; `resolver off;`
默认值	`resolver conf;`
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。

除 `resolver off;` 形式外，必须至少指定一个 DNS 服务器地址或 `conf` 参数。

备注

建议使用本地可信解析器，例如 `127.0.0.53` (systemd-resolved)，而非公共解析器（如 `8.8.8.8`）。公共解析器会将 DNS 查询暴露给第三方，并增加缓存投毒攻击的风险。

备注

该指令值会被嵌套块继承，并可在其中根据需要覆盖。在单个块内，该指令只能指定一次；重复指定属于配置错误。

该指令启用域名解析。它用于通过变量引用或使用*resolve* 参数配置的上游服务器，以及 ACME 客户端和 OCSP stapling。

除了显式的服务器地址外，该指令还识别以下参数：

`off`	显式禁用域名解析
`conf` [=*file*]	从指定文件读取 DNS 服务器地址，如果未指定文件，则从 `/etc/resolv.conf` 读取

该文件可以包含 `nameserver` 行，其格式与系统解析器使用的格式相同：

```
nameserver 127.0.1.4
nameserver 127.0.1.5
```

从文件读取的地址会添加到指令中显式指定的任何地址中，然后所有这些地址一起以轮询方式查询。如果文件不存在、无法读取或未列出任何地址，且未显式配置任何地址，则 Angie 回退到默认服务器 `127.0.0.1:53`。

当文件发生变化时，Angie 会自动重新读取，并相应地更新活动的服务器集合。

默认情况下，Angie 会在 DNS 响应中 TTL 值的有效期内缓存每个答案。可选的 `valid` 参数可覆盖此行为：

`valid`	可选参数允许覆盖响应缓存的有效期

```
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，并将其暴露于 */status/resolvers/<zone>*、「*DNS Resolvers*」选项卡 及 *Prometheus* 输出中。如未设置，则不会收集这些指标，且不会输出任何警告信息

小技巧

为防止 DNS 欺骗, 建议在适当保护的可信本地网络中使用 DNS 服务器。

小技巧

在 Docker 中运行时, 使用相应的内部 DNS 服务器地址, 例如 `127.0.0.11`。

resolver_timeout

语法	`resolver_timeout` *time*;
默认值	`resolver_timeout 30s;`
上下文	http, server, location, upstream

设置名称解析的超时时间, 例如：

```
resolver_timeout 5s;
```

root

语法	root *path*;
默认值	root html;
上下文	http, server, location, if in location

设置请求的根目录。例如, 使用以下配置

```
location /i/ {
  root /data/w3;
}
```

将发送 /data/w3/i/top.gif 文件来响应 /i/top.gif 请求。

path 值可以包含变量, 但*$document_root* 和*$realpath_root* 除外。

文件路径的构造仅仅是将 URI 添加到 root 指令的值。如果需要修改 URI, 应使用*alias* 指令。*path* 值末尾的斜杠会被丢弃，因为追加的 URI 自身带有前导斜杠。

satisfy

语法	satisfy all | any;
默认值	satisfy all;
上下文	http, server, location

如果所有 (all) 或至少一个 (any) *Access*、*Auth Basic* 或*Auth Request* 模块允许访问, 则允许访问。

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

如果该指令设置为非零值,Angie 将尝试通过使用*kqueue* 方法的 `NOTE_LOWAT` 标志或 `SO_SNDLOWAT` 套接字选项来最小化客户端套接字上的发送操作次数。在这两种情况下都使用指定的大小。

send_timeout

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

设置向客户端传输响应的超时时间。超时仅在两次连续的写操作之间设置，而不是用于整个响应的传输。如果客户端在此时间内没有收到任何内容，连接将被关闭。

sendfile

语法	`sendfile on | off;`
默认值	`sendfile off;`
上下文	http, server, location, if in location

启用或禁用 `sendfile()` 的使用。

aio 可用于为 `sendfile()` 预加载数据:

```
location /video/ {
  sendfile       on;
  tcp_nopush     on;
  aio            on;
}
```

在此配置中,:samp:*sendfile()* 使用 `SF_NODISKIO` 标志调用，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，但后续读取将仅以 16K 块加载数据。这可以使用*read_ahead* 指令更改。

sendfile_max_chunk

语法	`sendfile_max_chunk` *size*;
默认值	`sendfile_max_chunk 2m;`
上下文	http, server, location

限制单次 `sendfile()` 调用中可以传输的数据量。如果没有限制，一个快速连接可能会完全占用工作进程。

server

语法	server { ... }
默认值	—
上下文	http

设置虚拟服务器的配置。基于 IP(基于 IP 地址) 和基于名称 (基于"Host" 请求头字段) 的虚拟服务器之间没有明确的分隔。相反,:ref:*listen* 指令描述应该接受服务器连接的所有地址和端口,:ref:*server_name* 指令列出所有服务器名称。

Angie 如何处理请求 文档中提供了配置示例。

server_name

语法	server_name *name* ...;
默认值	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;
}
```

第一个名称成为主服务器名称。

服务器名称可以包含星号 ("*") 来替换名称的第一部分或最后部分:

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

这样的名称称为通配符名称。

上述提到的前两个名称可以合并为一个:

```
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, 这些捕获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会创建变量，这些变量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

如果指令的参数设置为*$hostname*，则使用机器名称。

也可以指定空服务器名称：

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

按名称搜索虚拟服务器时，如果名称匹配多个指定的变体（例如，通配符名称和正则表达式都匹配），将按以下优先级顺序选择第一个匹配的变体：

- 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 (按配置文件中出现的顺序), 包括空名称。

警告

要在 TLS 中使用 `server_name`, 需要 TLS 连接终止。此指令匹配 HTTP 请求中的 `Host`, 因此必须完成握手并解密连接。

server_name_in_redirect

语法	`server_name_in_redirect on | off;`
默认值	`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* 指令指定的主服务器名称。

`on`	使用由*server_name* 指令设置的主服务器名称
`off`	使用”Host” 请求头字段中的名称。如果此字段不存在, 则使用服务器的 IP 地址。

重定向中端口的使用由*port_in_redirect* 指令控制。

server_names_hash_bucket_size

语法	`server_names_hash_bucket_size` *size*;
默认值	`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在单独的文档 中提供。

server_names_hash_max_size

语法	`server_names_hash_max_size` *size*;
默认值	`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在单独的文档 中提供。

server_tokens

语法	`server_tokens on` | `off` | `build` | *string*;
默认值	`server_tokens on;`
上下文	http, server, location

启用或禁用在错误页面和 `Server` 响应头字段中显示 Angie 版本。

`build` 参数启用显示构建名称, 该名称由相应的 configure 参数设置, 与版本一起显示。

在 Angie PRO 中, 如果指令设置了一个 *string*, 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后的值来代替服务器名称、版本和构建名称。空 *string* 将禁用 `Server` 字段的显示。

status_zone

在 1.12.0 版本发生变更.

语法	`status_zone off` | *zone* | *key* `zone=`*zone*[:*number*];
默认值	—
上下文	server, location, if in location

分配一个共享内存区域用于收集*/status/http/location_zones/<zone>* 和*/status/http/server_zones/<zone>* 指标。

多个 `server` 上下文可以共享同一个区域进行数据收集; 特殊值 `off` 在嵌套的 `location` 块中禁用数据收集。

使用单个 *zone* 值的语法将当前上下文的所有指标合并到一个共享内存区域中:

```
server {

    listen 80;
    server_name *.example.com;

    status_zone single;
    # ...
}
```

替代语法允许设置以下参数:

key	包含变量的字符串, 其值决定区域中请求的分组。所有在替换后产生相同值的请求被分组在一起。如果替换产生空值, 则不更新指标。
zone	共享内存区域的名称。
number (可选)	用于收集指标的独立组的最大数量。如果新的 *key* 值超过此限制, 它们将被分组到 `zone` 下。 默认值为 1。

在以下示例中, 所有共享相同 `$host` 值的请求被分组到 `host_zone` 中。为每个唯一的 `$host` 单独跟踪指标, 直到有 10 个指标组。一旦达到此限制, 任何额外的 `$host` 值都将包含在 `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 输出中按各个主机分割。

备注

仅在设置了 `status_zone` 时才会收集这些指标。如未设置，对应的 server 或 location 将不会出现在 */status/http/server_zones/<zone>*、*/status/http/location_zones/<zone>*、「*HTTP Zones*」小部件及 *Prometheus* 输出中，且不会输出任何警告信息。参见配置示例。

subrequest_output_buffer_size

语法	`subrequest_output_buffer_size` *size*;
默认值	`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。这是 `4K` 或 `8K`, 取决于平台。它可以设置得更小。

备注

该指令仅适用于响应体保存在内存中的子请求。例如, 这样的子请求由 *SSI* 创建。

tcp_nodelay

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

启用或禁用 `TCP_NODELAY` 选项的使用。当连接转换到保持活动状态时启用该选项。此外，它在 SSL 连接、无缓冲代理和 *WebSocket* 代理 中也被启用。

tcp_nopush

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

启用或禁用在 FreeBSD 上使用 `TCP_NOPUSH` 套接字选项或在 Linux 上使用 `TCP_CORK` 套接字选项。这些选项仅在使用 *sendfile* 时启用。启用该选项允许

- 在 Linux 和 FreeBSD 4.* 上，在一个数据包中发送响应头和文件的开头；
- 以完整数据包发送文件。

time_format

Added in version 1.12.0.

语法	`time_format` *$variable format*;
默认值	—
上下文	http

定义一个 *$variable* 来保存当前本地时间，使用自定义 *format* 字符串渲染，该字符串由类似 `strftime` 的说明符以及表示毫秒的 `%L` 组成。当 *$time_local* 和 *$time_iso8601* 无法满足需求时（例如因为它们不提供亚秒级精度），该指令很有用：

```
time_format $time_ms "%Y-%m-%dT%H:%M:%S.%L%Z";

log_format tf '$time_ms $remote_addr "$request" $status';
```

使用上述格式时，变量 `$time_ms` 会展开为类似 `2024-01-15T14:34:56.789+03:00` 的值。

重新定义内置变量（例如 *$time_local*），或针对同一名称重复使用 `time_format`，都会使配置加载失败，并报出类似"the duplicate "..." variable" 的错误。

format 字符串接受以下说明符，这些说明符在与 *$time_local* 相同的本地时区中求值：

%Y	4 位数年份
%y	2 位数年份
%m	月份，01-12
%d	一个月中的第几天，01-31
%e	一个月中的第几天，空格填充，1-31
%H	小时，00-23
%I	小时，01-12
%M	分钟，00-59
%S	秒，00-59
%L	毫秒，000-999
%p	大写 AM / PM
%P	小写 am / pm
%a	缩写的星期名称，Sun-Sat
%A	完整的星期名称，Sunday-Saturday
%b, %h	缩写的月份名称，Jan-Dec
%B	完整的月份名称，January-December
%z	时区偏移，+0300
%Z	时区偏移，ISO 8601 风格，+03:00
%n	换行符
%t	制表符
%%	字面量 %

备注

无法识别的说明符会原样复制到输出中，包括前导的 %；位于 *format* 字符串最末尾的单独 % 会被丢弃。

try_files

语法	`try_files` *file* ... *uri*; `try_files` *file* ... =*code*;
默认值	—
上下文	server, location

按指定顺序检查文件是否存在，并使用第一个找到的文件进行请求处理；处理在当前*location* 的上下文中执行。文件的路径根据 *file* 参数按照*root* 和*alias* 指令构造。可以通过在名称末尾指定斜杠来检查目录是否存在，例如 `$uri/`。如果没有找到任何文件，则进行内部重定向到最后一个参数中指定的 *uri*。

参数 *file* 和 *uri* 可以包含变量。

例如：

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

location = /images/default.gif {
  expires 30s;
}
```

最后一个参数可以是用于内部重定向的 URI、对命名 location 的引用 (例如 `@drupal`)、或以 `=code` 形式的响应码 (例如 `=404`):

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

应该注意的是, 过度使用 `try_files` 指令会增加系统调用的数量, 这可能会对性能产生负面影响。

因此,:samp:*try_files* 不应该用于复制实际上是默认行为的行为, 例如:

```
location /bad_pattern {

      # try_files $uri $uri/ =404;  # 不推荐!
}
```

此外,:samp:*try_files* 不应该仅用于在文件不存在时进行重定向。原因是 `try_files` 指令有两个特点:

- 首先, 它检查每个文件是否存在, 这会增加系统负载。
- 其次, 任何文件打开错误 (例如 打开文件过多、权限错误) 也被视为文件不存在并触发回退到备用处理程序, 这可能会用成功响应掩盖 5xx 错误并导致错误的缓存。

因此, 在实践中, 可能会遇到以下有问题的结构:

```
location / {
   try_files $uri $uri/ @drupal;  # 不推荐!
}
```

这里的问题是唯一的目的是重定向。使用 `try_files` 会导致上述缺点, 但没有任何好处, 因为不需要检查文件是否存在。正确的解决方案是使用*error_page* 指令, 它没有这些缺点:

```
error_page 404 = @drupal;
log_not_found off;
```

相比之下, 在以下示例中:

```
location ~ \.php$ {
  try_files $uri @drupal;

  fastcgi_pass ...;
```

```
  fastcgi_param SCRIPT_FILENAME /path/to$fastcgi_script_name;

#  ...
}
```

try_files 指令在将请求传递给同一块中配置的 FastCGI 服务器之前检查 PHP 文件是否存在; 这里使用 try_files 是合理的。

代理到 Mongrel 时的使用示例:

```
location / {
  try_files /system/maintenance.html
            $uri $uri/index.html $uri.html
            @mongrel;
}

location @mongrel {
  proxy_pass http://mongrel;
}
```

与 Drupal/FastCGI 一起使用的示例:

```
location / {
  error_page 404 = @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
}
```

与 Wordpress 和 Joomla 一起使用的示例:

```
location / {
  error_page 404 = @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 { ... }
默认值	types *text/html html; image/gif gif; image/jpeg jpg;*
上下文	http, server, location

将文件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一个类型, 例如:

```
types {
  application/octet-stream bin exe dll;
  application/octet-stream deb;
  application/octet-stream dmg;
}
```

Angie 随附了一个足够完整的映射表, 位于 `conf/mime.types` 文件中。

要使特定的 location 对所有响应返回"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*;
默认值	`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，请参阅单独讨论。

types_hash_max_size

语法	`types_hash_max_size` *size*;
默认值	`types_hash_max_size 1024;`
上下文	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，请参阅单独讨论。

underscores_in_headers

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

启用或禁用在客户端请求头字段中使用下划线。当禁用下划线的使用时，名称包含下划线的请求头字段将被标记为无效，并受*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

语法	`variables_hash_bucket_size` *size*;
默认值	`variables_hash_bucket_size 64;`
上下文	http

设置变量哈希表的桶大小。有关设置哈希表的详细信息，请参阅单独讨论。

variables_hash_max_size

在 1.11.0 版本发生变更.

语法	`variables_hash_max_size` *size*;
默认值	`variables_hash_max_size 2048;`
上下文	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, 请参阅单独讨论。

内置变量

`http_core` 模块支持与 Apache Server 变量名称匹配的内置变量。首先, 这些是表示客户端请求头字段的变量, 例如 `$http_user_agent`、`$http_cookie` 等。此外, 还有其他变量:

`$angie_version`

Angie 版本

`$arg_<name>`

请求行中的参数 *name*

`$args`

请求行中的参数

`$binary_remote_addr`

二进制形式的客户端地址, 对于 IPv4 地址, 值的长度始终为 4 字节, 对于 IPv6 地址则为 16 字节

`$body_bytes_sent`

发送给客户端的字节数, 不包括响应头; 此变量与 `mod_log_config` Apache 模块的"%B" 参数兼容

`$bytes_sent`

发送给客户端的字节数

`$connection`

连接序列号

`$connection_requests`

通过一个连接发出的当前请求数

`$connection_time`

连接时间 (以秒为单位), 具有毫秒分辨率

`$content_length`

`Content-Length` 请求头字段

`$content_type`

`Content-Type` 请求头字段

`$cookie_<name>`

具有指定 *name* 的 cookie

`$document_root`

当前请求的*root* 或*alias* 指令的值

`$document_uri`

与*$uri* 相同

`$host`

按以下优先顺序: 请求行中的主机名, 或"Host" 请求头字段中的主机名, 或与请求匹配的服务器名称

`$hostname`

主机名

`$http_<name>`

在 1.11.0 版本发生变更: 在 HTTP/3 请求中，如果客户端未传递 `Host` 请求头，`$http_host` 将从 `:authority` 伪头字段中初始化。

任意请求头字段; 变量名的最后部分对应于转换为小写并将破折号替换为下划线的字段名称

`$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

`$is_request_port`

如果*$request_port* 值非空, 则为 `:`, 否则为空字符串

`$limit_rate`

设置此变量可启用响应速率限制; 参见*limit_rate*

`$msec`

当前时间 (以秒为单位), 具有毫秒分辨率

`$nginx_version`

nginx 版本

`$pid`

工作进程的 PID

`$pipe`

如果请求是流水线式的, 则为 `p`, 否则为 `.`

`$proxy_protocol_addr`

来自 PROXY 协议头的客户端地址

必须先通过在*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

必须先通过在*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

必须先通过在*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

必须先通过在*listen* 指令中设置 `proxy_protocol` 参数来启用 PROXY 协议。

`$proxy_protocol_tlv_<name>`

来自 PROXY 协议头的 TLV。*name* 可以是 TLV 类型名称或其数值。在后一种情况下, 该值为十六进制, 应以 `0x` 为前缀:

```
$proxy_protocol_tlv_alpn
$proxy_protocol_tlv_0x01
```

SSL TLV 也可以通过 TLV 类型名称或其数值访问, 两者都以 `ssl_` 为前缀:

```
$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 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, 还支持以下特殊 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且验证成功, 则为 `0`, 否则为非零值

必须先通过在*listen* 指令中设置 `proxy_protocol` 参数来启用 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, 所有符号链接都解析为实际路径

`$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

当请求体被读取到内存缓冲区 时, 该变量的值在由*proxy_pass*、*fastcgi_pass*、*uwsgi_pass* 和*scgi_pass* 指令处理的位置中 * 可用 *。

`$request_body_file`

包含请求体的临时文件名

在处理结束时, 需要删除该文件。要始终将请求体写入文件, 请启用*client_body_in_file_only*。在代理请求或向 FastCGI/uwsgi/SCGI 服务器发送请求时传递临时文件名时, 应分别使用*proxy_pass_request_body off*、*fastcgi_pass_request_body off*、*uwsgi_pass_request_body off* 或*scgi_pass_request_body off* 指令禁用请求体本身的传递。

`$request_completion`

如果请求已完成, 则为 `OK`, 否则为空字符串

`$request_filename`

当前请求的文件路径, 基于*root* 或*alias* 指令以及请求 URI

`$request_id`

从 16 个随机字节生成的唯一请求标识符, 以十六进制表示

`$request_length`

请求长度 (包括请求行、头和请求体)

`$request_method`

请求方法, 通常为 `GET` 或 `POST`

`$request_port`

按以下优先顺序: 请求 URI 的 authority 组件中的端口号, 或”Host” 请求头字段中的端口号

`$request_time`

请求处理时间 (以秒为单位), 具有毫秒分辨率; 从客户端读取第一个字节以来经过的时间

`$request_uri`

完整的原始请求 URI (带参数), 在请求处理过程中不会被修改; 参阅*$uri* 了解当前 (可能已重写的) URI

`$scheme`

请求方案,"http" 或"https"

`$sent_body`

Added in version 1.11.0.

子请求或外部请求的响应体 (当其存储在内存中时); 否则为空字符串

`$sent_http_<name>`

任意响应头字段; 变量名的最后部分对应于转换为小写并将破折号替换为下划线的字段名称

`$sent_trailer_<name>`

在响应末尾发送的任意字段; 变量名的最后部分对应于转换为小写并将破折号替换为下划线的字段名称

`$server_addr`

接受请求的服务器地址

计算此变量的值通常需要一次系统调用。为避免系统调用,:ref:*listen* 指令必须指定地址并使用 `bind` 参数。

`$server_name`

接受请求的服务器名称

`$server_port`

接受请求的服务器端口

`$server_protocol`

请求协议, 通常为"HTTP/1.0"、"HTTP/1.1" 或"HTTP/2.0"

`$status`

响应状态

`$tcpinfo_rtt`, `$tcpinfo_rttvar`, `$tcpinfo_snd_cwnd`, `$tcpinfo_rcv_space`

关于客户端 TCP 连接的信息; 在支持 `TCP_INFO` 套接字选项的系统上可用

`$time_iso8601`

ISO 8601 标准格式的本地时间

`$time_local`

通用日志格式的本地时间

`$uri`

请求中的当前 URI, 已规范化

`$uri` 的值可能在请求处理过程中发生变化, 例如在使用*rewrite* 进行重写时、执行内部重定向时, 或在使用索引文件时。有关该值如何映射到文件系统路径，请参阅选择位置。

3.2.1.3 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

Added in version 1.10.0.

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。

备注

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

在此示例中，由 `example` ACME 客户端（在 `http` 块中声明）获取的证书使用默认的 HTTP 验证来保护 TCP 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;
    }
}
```

配置示例和设置说明（包括 DNS 验证）请参阅*Stream* 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。

备注

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
volatile	指示该变量不可缓存

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，需要使用 --with-stream_geoip_module 构建选项启用此模块。

备注

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_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_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_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”。

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。键可以包含变量。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路径、格式和配置。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。

警告

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` 参数指定的时间间隔更旧；
- 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，则缓冲区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，使用 `64K` 字节的缓冲区大小和压缩级别 `1`。数据以原子块压缩，日志文件可以在任何时候通过 `"zcat"` 工具解压或读取。

示例：

```
access_log /path/to/log.gz basic gzip flush=5m;
```

备注

为了支持 gzip 压缩，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；
- 缓冲不起作用；
- 每次日志写入时都会打开文件，写入后立即关闭。然而，由于可以将频繁使用的文件的描述符存储在缓存中，因此在日志轮转期间，可能会在*open_log_file_cache* 指令的 `valid` 参数指定的时间内继续写入旧文件。

`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，例如：

```
log_format proxy '$remote_addr [$time_local] '
                 '$protocol $status $bytes_sent $bytes_received '
                 '$session_time "$upstream_addr" '
                 '"$upstream_bytes_sent" "$upstream_bytes_received" "$upstream_connect_time"';
```

`escape` 参数允许在变量中设置 `json` 或 `default` 字符转义；默认使用 `default`。`none` 值禁用字符转义。

使用 `default` 时，字符”"”、”\” 以及值小于 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”-”。

使用 `json` 时，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，用于存储名称包含变量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；当缓存溢出时，关闭最近最少使用（LRU）的描述符。
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。

Metric

Added in version 1.12.0.

`ngx_stream_metric_module` 模块允许为 stream（TCP 和 UDP）流量创建任意的实时计算指标。这些指标值存储在共享内存中，并在 `/status/stream/metric_zones/` API 分支中实时显示。支持多种数据聚合类型（计数器、直方图、移动平均等），并可按任意键进行分组。

有关响应模式，请参见*Stream* 指标区域 *API* 参考。

配置示例

按客户端地址统计连接数：

```
stream {
    metric_zone connections:1m count;

    server {
        listen 12345;

        metric connections $remote_addr on=connect;
    }
}

http {
    server {
        listen 80;

        location /status/ {
            allow 127.0.0.1;
            deny all;
            api /status/;
        }
    }
}
```

如果向端口 12345 发起连接：

```
$ nc 127.0.0.1 12345 </dev/null
```

connections 指标会实时更新：

```
{
    "stream": {
       "metric_zones": {
           "connections": {
               "discarded": 0,
               "metrics": {
                   "127.0.0.1": 1
               }
           }
       }
    }
}
```

指令

metric

语法	metric *name* *key=value* [on=connect| preread| end];
默认值	—
上下文	stream, server

计算指定共享内存区域 *name* 的指标值。

参数：

- ***key* —任意字符串（通常是变量），用于对值进行分组。**
  最大长度为 255 字节；更长的键会被截断。在 API 输出中，Angie 会为每个 255 字节的键（包括原始长度恰好为 255 字节的键）附加 ...。键本身可以包含 = 字符—只有 最后一个 = 之后的文本才被视为值，因此它之前的所有内容（包括任何嵌入的 =）都成为键；
- ***value* —由所选模式处理的数字（可以是变量）。**
  如果省略，默认为 0。如果参数无法转换为数字，则默认为 1；
- on —可选参数，指定在会话处理的哪个阶段计算指标：
  - 如果为 on=connect，计算发生在 Pre-access 阶段，即在从客户端读取任何数据之前；
  - 如果为 on=preread，每个会话计算一次，即当来自上游侧的第一批数据—从使用*proxy_pass* 代理到的服务器接收，或由*return* 生成—被发送给客户端时；此时，proxy_protocol、*ssl_preread*、mqtt_preread 等模块已经处理了客户端的初始数据；
  - 如果为 on=end（默认），计算发生在 Log 阶段，即会话结束时。

使用示例：

```
metric sessions $remote_addr=$session_time on=end;
```

备注

具有空键或无效 key=value 对的指标将被忽略。省略的 *value* 被视为 0：

```
metric foo $bar;  # 等同于 $bar=0
```

这对于 count 模式很有用，该模式忽略数值，只对指标被更新这一事实做出反应。

备注

请记住，变量在不同阶段进行求值。例如，无法将 $upstream_bytes_sent（发送到上游的字节数）与 on=connect（在连接被代理到任何地方之前）一起使用。

metric_complex_zone

语法	metric_complex_zone *name:size* [expire=on| off] [discard_key=name] { ... }
默认值	—
上下文	stream

定义一个 复合指标—一组具有独立模式的指标。块体中的每一行定义一个 子指标名称、一个 模式以及可选的模式 参数。区域大小必须至少为八个系统内存页。

使用示例：

```
metric_complex_zone sessions:1m expire=on discard_key="old" {
    # 子指标名称        模式           参数
    min_time            min;
    avg_time            average exp    factor=60;
    max_time            max;
    total               count;
}
```

在 API 树中，这样的复合指标模板如下所示：

```
{
    "discarded": 3,
    "metrics": {
        "key1": {
            "min_time": 20,
            "avg_time": 50,
            "max_time": 80,
            "total": 2
        },
        "old": {
             "min_time": 3,
             "avg_time": 40,
             "max_time": 152,
             "total": 80
        }
    }
}
```

metric_zone

语法	metric_zone *name:size* [expire=on| off] [discard_key=name] *mode* [*parameters*];
默认值	—
上下文	stream

创建一个指定 *size* 大小和给定 *name* 名称的共享内存区域来存储指标。该区域名称作为 /status/

`stream/metric_zones/` 分支中的节点。区域大小必须至少为八个系统内存页。

参数：

- `expire=<on|off>` —区域满时的行为：
  - 如果为 `on`，则丢弃最旧的指标（按更新时间）以释放内存供新指标使用；
  - 如果为 `off`（默认）—丢弃新传入的指标，保留现有条目。
- `discard_key=<name>` —定义一个键为 *name* 的指标，用于累积被丢弃指标的值。默认情况下不创建此类指标。保留键不能手动更新。
- *mode* —数据处理算法（参见操作模式 部分）；
- *parameters* —所选模式的附加设置（例如，`average exp` 的 `factor`）。

使用示例：

```
metric_zone session_time:1m max;
```

在 API 树中，共享内存区域模板如下所示：

```
{
    "discarded": 0,
    "metrics": {
        "key1": 123,
        "key2": 10.5
    }
}
```

备注

在 1 MB 区域中，键大小为 39 字节且单一指标模式时，大约可以存储 8,000 个唯一键条目。

操作模式

可用指标操作模式列表：

- `count` —计数器；
- `gauge` —仪表（增加/减少）；
- `last` —最后接收到的值；
- `min` —最小值；
- `max` —最大值；
- `average exp` —指数移动平均（EMA）（参数 `factor`）；
- `average mean` —窗口内的平均值（参数 `window` 和 `count`）；

- histogram —跨” 桶” 的分布（阈值列表）。

以下示例使用配置示例中的 /status/ 端点。每个结果都可在 /status/stream/metric_zones/<zone>/metrics/ 下获取。

count

无论提供什么值，每次指标更新都会将计数器增加 1。

默认值—0。

示例：

```
metric_zone count:1m count;

# 作为复合指标的一部分：
#
# metric_complex_zone count:1m {
#     some_metric_name  count;
# }

server {
    listen 12345;

    metric count KEY;
}
```

更新指标：

```
$ nc 127.0.0.1 12345 </dev/null
$ nc 127.0.0.1 12345 </dev/null
$ nc 127.0.0.1 12345 </dev/null
$ nc 127.0.0.1 12345 </dev/null
```

API 中的预期指标值：

```
{
    "KEY": 4
}
```

gauge

gauge 根据传入数字的符号增加或减少其值。正值增加计数器，负值减少计数器。值为 0 时不改变计数器。

默认值—0。

示例：

```
metric_zone gauge:1m gauge;

# 作为复合指标的一部分：
#
# metric_complex_zone gauge:1m {
#     some_metric_name  gauge;
# }

server {
    listen 12351;
    metric gauge KEY;
}

server {
    listen 12352;
    metric gauge KEY=5;
}

server {
    listen 12353;
    metric gauge KEY=-5;
}

server {
    listen 12354;
    metric gauge KEY=8;
}
```

更新指标：

```
$ nc 127.0.0.1 12351 </dev/null
```

API 中的预期指标值：

```
{
    "KEY": 0
}
```

进一步更新：

```
$ nc 127.0.0.1 12352 </dev/null
$ nc 127.0.0.1 12353 </dev/null
$ nc 127.0.0.1 12354 </dev/null
```

API 中的预期指标值：

```
{
    "KEY": 8
```

```
}
```

last

存储最后接收到的值，不进行任何聚合。如果省略 *value*，则使用 0。

示例：

```
metric_zone last:1m last;

# 作为复合指标的一部分：
#
# metric_complex_zone last:1m {
#     some_metric_name  last;
# }

server {
    listen 12361;
    metric last KEY;
}

server {
    listen 12362;
    metric last KEY=8000;
}

server {
    listen 12363;
    metric last KEY=37;
}

server {
    listen 12364;
    metric last KEY=-3.5;
}
```

更新指标：

```
$ nc 127.0.0.1 12361 </dev/null
```

API 中的预期指标值：

```
{
    "KEY": 0
}
```

进一步更新：

```
$ nc 127.0.0.1 12362 </dev/null
$ nc 127.0.0.1 12363 </dev/null
$ nc 127.0.0.1 12364 </dev/null
```

API 中的预期指标值：

```
{
   "KEY": -3.5
}
```

min

保存两个值中的最小值—当前存储的值和新值。

示例：

```
metric_zone min:1m min;

# 作为复合指标的一部分：
#
# metric_complex_zone min:1m {
#     some_metric_name  min;
# }

server {
    listen 12371;
    metric min KEY=42.999;
}

server {
    listen 12372;
    metric min KEY=-512;
}

server {
    listen 12373;
    metric min KEY=1;
}
```

更新指标：

```
$ nc 127.0.0.1 12371 </dev/null
$ nc 127.0.0.1 12372 </dev/null
$ nc 127.0.0.1 12373 </dev/null
```

API 中的预期指标值：

```
{
    "KEY": -512
}
```

max

保存两个值中的最大值—当前存储的值和新值。

示例：

```
metric_zone max:1m max;

# 作为复合指标的一部分：
#
# metric_complex_zone max:1m {
#     some_metric_name  max;
# }

server {
    listen 12381;
    metric max KEY=42.999;
}

server {
    listen 12382;
    metric max KEY=-512;
}

server {
    listen 12383;
    metric max KEY=1;
}
```

更新指标：

```
$ nc 127.0.0.1 12381 </dev/null
$ nc 127.0.0.1 12382 </dev/null
$ nc 127.0.0.1 12383 </dev/null
```

API 中的预期指标值：

```
{
    "KEY": 42.999
}
```

average exp

使用 指数平滑 算法计算平均值。

接受可选参数 `factor=<number>` —决定新值对平均值影响程度的系数。允许的整数值范围为 0 到 99。默认值为 90。

系数越高，新值的权重越大。如果指定 90，结果将是新值的 90% 加上先前平均值的 10%。

示例：

```
metric_zone avg_exp:1m average exp factor=60;

# 作为复合指标的一部分：
#
# metric_complex_zone avg_exp:1m {
#     some_metric_name  average exp  factor=60;
# }

server {
    listen 12391;
    metric avg_exp KEY=100;
}

server {
    listen 12392;
    metric avg_exp KEY=200;
}

server {
    listen 12393;
    metric avg_exp KEY=0;
}

server {
    listen 12394;
    metric avg_exp KEY=8;
}

server {
    listen 12395;
    metric avg_exp KEY=30;
}
```

更新指标：

```
$ nc 127.0.0.1 12391 </dev/null
$ nc 127.0.0.1 12392 </dev/null
$ nc 127.0.0.1 12393 </dev/null
$ nc 127.0.0.1 12394 </dev/null
```

```
$ nc 127.0.0.1 12395 </dev/null
```

API 中的预期指标值：

```
{
    "KEY": 30.16
}
```

average mean

计算算术平均值。接受可选参数 `window=<off|time>` 和 `count=<number>`，分别定义用于平均的时间间隔和样本大小。默认为 `window=off`（使用整个样本）和 `count=10`。

备注

例如，`window=5s` 将仅考虑最近 5 秒内的事件。`window` 参数不能为 0。`count=number` 参数控制样本大小（缓存值），以实现更平滑的平均值计算。

示例：

```
metric_zone avg_mean:1m average mean window=5s count=8;

# 作为复合指标的一部分：
#
# metric_complex_zone avg_mean:1m {
#     some_metric_name  average mean  window=5s count=8;
# }

server {
    listen 12401;
    metric avg_mean KEY=0.1;
}

server {
    listen 12402;
    metric avg_mean KEY=0.4;
}

server {
    listen 12403;
    metric avg_mean KEY=10;
}

server {
    listen 12404;
    metric avg_mean KEY=1;
}
```

更新指标：

```
$ nc 127.0.0.1 12401 </dev/null
$ nc 127.0.0.1 12401 </dev/null
$ nc 127.0.0.1 12402 </dev/null
$ nc 127.0.0.1 12403 </dev/null
$ nc 127.0.0.1 12404 </dev/null
$ nc 127.0.0.1 12404 </dev/null
```

API 中的预期指标值：

```
{
    "KEY": 2.1
}
```

如果从最后一次更新起等待 5 秒，预期值将为：

```
{
    "KEY": 0
}
```

histogram

创建一组" 桶"，如果新值不超过桶的阈值，则递增相关计数器。参数以数值阈值列表的形式提供。对于分析分布（如会话持续时间）很有用。

必需参数是 *numbers* —桶的阈值，通常按升序列出。

备注

桶值 `inf` 或 `+Inf` 可用于捕获所有超过最高指定桶的值。

示例：

```
metric_zone hist:1m histogram 0.1 0.2 0.5 1 2 inf;

# 作为复合指标的一部分：
#
# metric_complex_zone hist:1m {
#     some_metric_name  histogram  0.1 0.2 0.5 1 2 inf;
# }

server {
    listen 12411;
    metric hist KEY=0.25;
}
```

```
server {
    listen 12412;
    metric hist KEY=2;
}

server {
    listen 12413;
    metric hist KEY=1000;
}
```

更新指标：

```
$ nc 127.0.0.1 12411 </dev/null
```

API 中的预期指标值：

```
{
    "KEY": {
        "0.1": 0,
        "0.2": 0,
        "0.5": 1,
        "1": 1,
        "2": 1,
        "inf": 1
    }
}
```

进一步更新：

```
$ nc 127.0.0.1 12412 </dev/null
```

API 中的预期指标值：

```
{
    "KEY": {
        "0.1": 0,
        "0.2": 0,
        "0.5": 1,
        "1": 1,
        "2": 2,
        "inf": 2
    }
}
```

进一步更新：

```
$ nc 127.0.0.1 12413 </dev/null
```

API 中的预期指标值：

```
{
    "KEY": {
       "0.1": 0,
       "0.2": 0,
       "0.5": 1,
       "1": 1,
       "2": 2,
       "inf": 3
    }
}
```

内置变量

为每个指标创建以下变量：

- $metric_<name>
- $metric_<name>_key
- $metric_<name>_value

对于复合指标，会添加额外的变量：

- $metric_<name>_value_<metric>

$metric_<name>

与*metric* 指令类似，$metric_<name> 变量设置器可用于更新指标。计算发生在设置该变量时—例如，通过*set* 指令（该指令在 Pre-access 阶段求值，与 on=connect 处于同一阶段），或以编程方式从 njs 模块（例如在 js_access 处理程序中）进行。

该设置器接受一个键，并可带有可选的 =value 后缀。最后一个 = 将键与值分开；如果没有该后缀，值为 0。键和值可以包含文本、变量或两者的组合。这些解析规则与*metric* 的相同。

使用示例：

```
stream {
    metric_zone hits:1m count;

    # 此时添加了 $metric_hits 变量

    server {
        listen 12345;

        metric hits client=1 on=connect;
    }
```

```
    server {
        listen 12346;

        set $metric_hits client=1;

        return "$metric_hits\n";
    }
}
```

在向端口 12345 发起三次连接并向端口 12346 发起一次连接后：

```
$ nc 127.0.0.1 12345 </dev/null
$ nc 127.0.0.1 12345 </dev/null
$ nc 127.0.0.1 12345 </dev/null
$ nc 127.0.0.1 12346 </dev/null
client=4
```

备注

回读 $metric_<name> 会以 key=value 形式返回该键以及指标当前的 计算后值—而不是所赋的字面字符串。在上面的示例中，连接赋予的字面值为 client=1，但由于 count 模式忽略所赋的值并在每次更新时简单地递增，因此随后回读 $metric_hits 会返回 client=4 —即计数器的新总数，其中已经包含了来自端口 12345 的三次先前更新。

此外，存储在 $metric_<name>_key 变量中的值会更改为指定的键。

$metric_<name>_key 和 $metric_<name>_value

$metric_<name>_key 和 $metric_<name>_value 变量分别定义键和值。当设置 $metric_<name>_value 时会发生指标更新，前提是 $metric_<name>_key 中的键已经定义。

备注

对于复合指标，$metric_<name>_value 变量中的子指标值使用 ", " 分隔符连接。

使用示例：

```
stream {
    metric_zone level:1m gauge;

    # 此处添加了变量 $metric_level、$metric_level_key 和 $metric_level_value。

    metric_complex_zone stats:1m {
        count count;
        min   min;
```

```
        avg   average exp;
    }

    # 此处添加了 $metric_stats、$metric_stats_key 和 $metric_stats_value。

    server {
        listen 12345;

        metric level sensor=10 on=connect;
    }

    server {
        listen 12346;

        set $metric_level_key   "sensor";
        set $metric_level_value 5;

        # 或者：set $metric_level sensor=5;

        return "Updated with '$metric_level'\nValue='$metric_level_value'\n";
    }

    server {
        listen 12347;

        set $metric_stats_key   bar;
        set $metric_stats_value 9;

        # 或者：set $metric_stats bar=9;

        return "Updated with '$metric_stats'\nValues='$metric_stats_value'\n";
    }
}
```

使用此配置，先向端口 12345（将 gauge 的起始值设置为 10）发起连接，然后向端口 12346 发起连接，产生：

```
$ nc 127.0.0.1 12345 </dev/null
$ nc 127.0.0.1 12346 </dev/null
Updated with 'sensor=15'
Value='15'
```

向端口 12347 发起连接，产生：

```
$ nc 127.0.0.1 12347 </dev/null
Updated with 'bar=1, 9, 9'
Values='1, 9, 9'
```

备注

如果将空字符串赋值给 $metric_<name>_value，该值将被识别为 0。如果字符串由无法转换为数字的字符组成，则被识别为 1。

仅在 $metric_<name>_key 和 $metric_<name>_value 都已设置后才会进行计算。

在这种情况下，存储在 $metric_<name> 中的值将等于新的 计算后 key=value 对，而不是刚刚赋予的字面值。

$metric_<name>_key 中的值表示通过变量指定的最后一个键。

$metric_<name>_value 中的值表示为 $metric_<name>_key 中设置的键计算的最后一个值。

$metric_<name>_value_<metric>

对于复合指标，可以使用 $metric_<name>_value_<metric> 变量获取特定子指标的值，其中 *<metric>* 是子指标的名称。

使用示例：

```
stream {
    metric_complex_zone foo:1m {
        count count;
        min   min;
        avg   average exp;
    }

    # 添加 $metric_foo、$metric_foo_key、$metric_foo_value，
    # 以及 $metric_foo_value_count、$metric_foo_value_min、$metric_foo_value_avg。

    server {
        listen 12345;

        set $metric_foo_key   bar;
        set $metric_foo_value 9;

        # 或者：set $metric_foo bar=9;

        return "Updated with '$metric_foo'\nValues='$metric_foo_value'\nCount='$metric_foo_
↪value_count'\n";
    }
}
```

使用此配置，向端口 12345 发起连接，产生：

```
$ nc 127.0.0.1 12345 </dev/null
Updated with 'bar=1, 9, 9'
Values='1, 9, 9'
Count='1'
```

其他示例

按协议监控连接

```
metric_zone protocols:1m count;

server {
    listen 12345;
    listen 12346 udp;

    metric protocols $protocol;
}
```

响应：

```
{
    "TCP": 340,
    "UDP": 58
}
```

上游会话时间分布

```
metric_zone upstream_time:10m expire=on histogram
    0.05 0.1 0.3 0.5 1 2 5 10 inf;

server {
    listen 12345;

    proxy_pass backend;
    metric upstream_time $upstream_addr=$upstream_session_time on=end;
}
```

响应：

```
{
    "discarded": 0,
    "metrics": {
        "backend1:8080": {
            "0.05": 12,
            "0.1": 28,
            "0.3": 56,
            "0.5": 78,
            "1": 92,
            "2": 97,
            "5": 99,
            "10": 100,
```

```
            "inf": 100
        }
    }
}
```

活动连接数

```
stream {
    metric_zone active_connections:2m gauge;

    server {
        listen 12345;

        metric active_connections service_a=1 on=connect;
        metric active_connections service_a=-1 on=end;
    }

    server {
        listen 12346;

        metric active_connections service_b=1 on=connect;
        metric active_connections service_b=-1 on=end;
    }
}

http {
    server {
        listen 8080;

        location /connections/ {
            allow 127.0.0.1;
            deny all;
            api /status/stream/metric_zones/active_connections/metrics/;
        }
    }
}
```

响应：

```
{
    "service_a": 42,
    "service_b": 17
}
```

Prometheus 支持

Angie 包含一个内置模块，用于以 Prometheus 格式 显示指标，该模块支持自定义指标，包括在 stream 侧收集的指标。由于 Prometheus 暴露是仅 HTTP 的功能，因此*prometheus_template* 和*prometheus* 指令在 `http` 块中配置，并通过同一个统一的 `/status/` API 树引用 stream 指标区域。

作为集成示例，请考虑以下配置：

```
stream {
    # 创建 "upload" 指标
    metric_complex_zone upload:1m discard_key="other" {
        stats    histogram 64 256 1024 4096 16384 +Inf;
        sum      gauge;
        count    count;
        avg_size average exp;
    }

    upstream backend {
        server 127.0.0.1:15001;
    }

    server {
        listen 12345;

        # 在会话结束时，使用从客户端接收到的
        # 字节总数更新该指标
        proxy_pass backend;
        metric upload angie=$bytes_received on=end;
    }
}

http {
    # 描述 "upload" 指标的 Prometheus 模板
    prometheus_template upload_metric {
        'stats{le="$1"}' $p8s_value
                         path=~^/stream/metric_zones/upload/metrics/angie/stats/(.+)$
                         type=histogram;

        'stats_sum'      $p8s_value
                         path=/stream/metric_zones/upload/metrics/angie/sum;
        'stats_count'    $p8s_value
                         path=/stream/metric_zones/upload/metrics/angie/count;

        'avg_size'       $p8s_value
                         path=/stream/metric_zones/upload/metrics/angie/avg_size;
    }

    server {
        listen 80;
```

```
        # 指标抓取目标
        location /prometheus/upload_metric/ {
            prometheus upload_metric;
        }
    }
}
```

在向端口 12345 发起五次连接，分别发送 16384、64448、64、1028 和 1028 字节的负载后：

```
$ head -c 16384 /dev/urandom | nc 127.0.0.1 12345
$ head -c 64448 /dev/urandom | nc 127.0.0.1 12345
$ head -c 64 /dev/urandom | nc 127.0.0.1 12345
$ head -c 1028 /dev/urandom | nc 127.0.0.1 12345
$ head -c 1028 /dev/urandom | nc 127.0.0.1 12345
```

指标值将为：

```
{
    "stats": {
        "64": 1,
        "256": 1,
        "1024": 1,
        "4096": 3,
        "16384": 4,
        "+Inf": 5
    },

    "sum": 82952,
    "count": 5,
    "avg_size": 1077.9376
}
```

以 Prometheus 格式，该指标可在 /prometheus/upload_metric/ 获取：

```
# Angie Prometheus template "upload_metric"
# TYPE stats histogram
stats{le="64"} 1
stats{le="256"} 1
stats{le="1024"} 1
stats{le="4096"} 3
stats{le="16384"} 4
stats{le="+Inf"} 5
stats_sum 82952
stats_count 5
avg_size 1077.9376
```

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。

该模块允许基于 SNI 执行选择性的 SSL 终止。

配置示例

在 `stream` 模块处理 SSL/TLS 终止后，连接被转发到 `http` 模块：

```
stream {

    server {

        listen 8000 default_server;
        ssl_preread on;
        # ...
    }

    server {

        listen 8000;
        server_name foo.example.com;
        pass 127.0.0.1:8001; # to HTTP
    }

    server {

        listen 8000;
        server_name bar.example.com;
        # ...
    }
}

http {

    server {

        listen 8001 ssl;
        # ...

        location / {

            root html;
        }
    }
}
```

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

使到代理服务器的出站连接源自指定的本地 IP 地址。参数值可以包含变量。特殊值 `off` 取消从上一级配置继承的 *proxy_bind* 指令的效果, 允许系统自动分配本地 IP 地址。

`transparent` 参数允许到代理服务器的出站连接源自非本地 IP 地址, 例如, 来自客户端的真实 IP 地址:

```
proxy_bind $remote_addr transparent;
```

要使此参数生效, 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, 工作进程会从主进程继承 *CAP_NET_RAW* 能力。

备注

还应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*;
默认值	`proxy_buffer_size 16k;`
上下文	stream, server

设置用于从代理服务器读取数据的缓冲区大小。同时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

在代理服务器从组中移除或被重新解析 进程或*API* 命令 DELETE 标记为永久不可用后, 启用终止到该代理服务器的所有会话。

当处理客户端或代理服务器的下一个读取或写入事件时, 会话将被终止。

设置 *time* 启用会话终止超时; 设置为 on 时, 会话立即断开。

proxy_download_rate

语法	proxy_download_rate *rate*;
默认值	proxy_download_rate 0;
上下文	stream, server

限制从代理服务器读取数据的速度。rate 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, 因此如果 Angie 同时打开两个到代理服务器的连接, 总速率将是指定限制的两倍。

参数值可以包含变量。在需要根据特定条件限制速率的情况下, 这可能很有用:

```
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 代理将保持到两端都关闭连接为止。

proxy_next_upstream

语法	`proxy_next_upstream on | off;`
默认值	`proxy_next_upstream on;`
上下文	stream, server

当无法与代理服务器建立连接时, 确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通过尝试次数 和时间 进行限制。

proxy_next_upstream_timeout

语法	`proxy_next_upstream_timeout` *time*;
默认值	`proxy_next_upstream_timeout 0;`
上下文	stream, server

限制将连接传递给下一个 服务器的允许时间。

`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, 所有地址将以轮询方式使用。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, 在描述的服务器组 中搜索服务器名称, 如果未找到, 则使用*resolver* 确定。

proxy_protocol

语法	`proxy_protocol on | off;`
默认值	`proxy_protocol off;`
上下文	stream, server

为到代理服务器的连接启用 PROXY 协议。

proxy_protocol_tlv

Added in version 1.11.0.

语法	`proxy_protocol_tlv` *name value*;
默认值	—
上下文	stream, server

向发送到代理服务器的 PROXY 协议 v2 头添加 TLV。*value* 可以包含变量。*name* 可以是 TLV 类型名称或其数字值; 在后一种情况下, 值以十六进制指定, 必须以 *0x* 开头。对于 SSL TLV, 使用 `ssl_` 前缀; 特殊的 `ssl_verify` 名称设置 SSL TLV 的验证字段。该指令仅在*proxy_protocol_version* 设置为 2 时使用。

proxy_protocol_version

语法	`proxy_protocol_version 1 | 2;`
默认值	`proxy_protocol_version 1;`
上下文	stream, server

设置用于到代理服务器的连接的 PROXY 协议版本。该设置在启用*proxy_protocol* 时生效。版本 2 允许发送由*proxy_protocol_tlv* 指令配置的 TLV。

proxy_requests

语法	`proxy_requests` *number*;
默认值	`proxy_requests 0;`
上下文	stream, server

设置客户端数据报的数量, 达到该数量时, 客户端与现有 UDP 流会话之间的绑定将被断开。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新会话。当所有客户端数据报都传输到代理服务器并接收到预期的响应数量 时, 或者当达到超时 时, 会话终止。

proxy_responses

语法	`proxy_responses` *number*;
默认值	—
上下文	stream, server

当使用 *UDP* 协议时, 设置期望从代理服务器响应客户端数据报的数据报数量。该数字作为会话终止的提示。默认情况下, 数据报的数量不受限制。

如果指定为零值, 则不期望响应。但是, 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

为到代理服务器的出站连接配置”TCP keepalive” 行为。

`off`	默认情况下, 套接字使用操作系统的设置。
`on`	为套接字启用 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

为到代理服务器的连接启用 SSL/TLS 协议。

proxy_ssl_certificate

语法	proxy_ssl_certificate *file* [*file*];
默认值	—
上下文	stream, server

指定一个包含 PEM 格式证书的文件，用于向代理服务器进行身份验证。文件名中可以使用变量。

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_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	stream, server

定义一个缓存，用于存储使用变量指定的*SSL* 证书 和私钥。

该指令支持以下参数：

- max —设置缓存中的最大元素数量。当缓存溢出时，最近最少使用（LRU）的元素将被移除。
- inactive —定义元素在此时间内未被访问即会被删除的时长，即使其仍处于 valid 周期内；每次访问都会重置此计时器。默认为 10 秒。
- valid —定义缓存元素被视为有效并可重用的时间。默认为 60 秒。在此期间之后，证书将被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*];
默认值	—
上下文	stream, server

指定一个包含 PEM 格式私钥的文件, 用于向代理服务器进行身份验证。

可以指定值 `engine:`name`:id` 来代替文件, 这将从名为 name 的 OpenSSL 引擎中加载具有指定 id 的私钥。

可以指定值 `store:scheme:id` 来代替文件, 用于通过已在 OpenSSL provider 中注册的 URI scheme 加载具有指定 id 的私钥, 例如 pkcs11。

文件名中可以使用变量。

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

指定向代理服务器发送请求时启用的加密套件。加密套件以 OpenSSL 库理解的格式指定。

加密套件列表取决于安装的 OpenSSL 版本。可以使用 `openssl ciphers` 命令查看完整列表。

警告

当使用 OpenSSL 时,:samp:*proxy_ssl_ciphers* 指令 * 不 * 配置 TLS 1.3 的加密套件。要使用 OpenSSL 配置 TLS 1.3 加密套件, 请使用*proxy_ssl_conf_command* 指令, 该指令是为高级 SSL 配置添加的。

- 在 LibreSSL 中,TLS 1.3 加密套件 * 可以 * 使用 `proxy_ssl_ciphers` 配置。
- 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。

备注

当使用 OpenSSL 1.0.2 或更高版本时支持该指令。要使用 OpenSSL 配置 TLS 1.3 加密套件, 请使用 `ciphersuites` 命令。

可以在同一级别指定多个 *proxy_ssl_conf_command* 指令。当且仅当当前级别没有定义 *proxy_ssl_conf_command* 指令时, 这些指令才从上一级配置继承。

警告

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

指定一个包含 PEM 格式吊销证书 (CRL) 的文件, 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` 来自 `proxy_pass` 的主机;
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, 以及在与代理服务器建立连接时通过 *SNI* 传递 的服务器名称。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

语法	proxy_ssl_ntls on | off;
默认值	proxy_ssl_ntls off;
上下文	stream, server

当使用 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;
}
```

备注

Angie 必须使用 --with-ntls 配置参数构建，并使用相应的支持 NTLS 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*;
默认值	—
上下文	stream, server

指定一个包含私钥 密码短语的文件，每个密码短语单独占一行。加载密钥时依次尝试这些密码短语。

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

为向代理服务器发送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

决定在与代理服务器工作时是否可以重用 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

指定一个包含 PEM 格式受信任 CA 证书的文件, 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次连续读或写操作之间的超时时间。如果在此时间内没有数据传输，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

该限制是针对每个连接设置的，因此如果客户端同时打开两个连接，总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可能很有用：

```
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。该值可以包含变量。

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。该值可以包含变量。

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

为流代理服务器提供使用 SSL/TLS 协议所需的支持。

当 从源代码构建时, 此模块默认不会被构建; 应该使用 --with-stream_ssl_module 构建选项来启用它。

在来自 我们仓库的软件包和镜像中, 此模块已包含在构建中。

备注

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 数量设置为等于处理器数量,
- 启用共享 会话缓存,
- 禁用内置 会话缓存,
- 并可能增加会话生存时间 (默认为 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* ...;
默认值	—
上下文	stream, server

指定支持的 ALPN 协议列表。如果客户端使用 ALPN, 则必须协商 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*;
默认值	—
上下文	stream, 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，则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 格式的密钥也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单独证书链。对于旧版本，只能使用一个证书链。

备注

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 握手都会加载证书，这可能会对性能产生负面影响。

可以指定 `data:`$variable`` 值来代替 `file`，这将从变量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_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	stream, server

定义一个缓存，用于存储使用变量指定的*SSL* 证书 和私钥。

该指令支持以下参数：

- `max` —设置缓存中的最大元素数量。当缓存溢出时，最近最少使用（LRU）的元素将被移除。
- `inactive` —定义元素在此时间内未被访问即会被删除的时长，即使其仍处于 `valid` 周期内；每次访问都会重置此计时器。默认为 10 秒。
- `valid` —定义缓存元素被视为有效并可重用的时间。默认为 60 秒。在此期间之后，证书将被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_compression

语法	`ssl_certificate_compression on | off;`
默认值	`ssl_certificate_compression off;`
上下文	stream, server

启用 TLS 1.3 服务器证书 压缩。

备注

使用 OpenSSL 3.2 或更高版本时支持该指令; 支持的压缩算法列表由库提供。

备注

使用 BoringSSL 时支持该指令; 支持的压缩算法列表包括 `zlib`。

如果启用了*ssl_stapling*, 则证书压缩将被禁用。

ssl_certificate_key

语法	`ssl_certificate_key` *file*;
默认值	—
上下文	stream, server

指定给定服务器的 PEM 格式密钥文件。

备注

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 `engine:`name`:id` 值来代替 `file`, 这将从 OpenSSL 引擎 *name* 加载具有指定 *id* 的密钥。

可以指定 `store:scheme:id` 值来代替 `file`, 这用于从具有指定 *id* 和 OpenSSL 提供程序注册的 URI *scheme* 加载密钥, 例如 pkcs11。

可以指定 `data:`$variable`` 值来代替 `file`, 这将从变量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	stream, 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;
```

密码套件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。

警告

使用 OpenSSL 时,:samp:*ssl_ciphers* 指令 不会配置 TLS 1.3 的密码套件。要使用 OpenSSL 配置 TLS 1.3 密码套件, 请使用*ssl_conf_command* 指令, 该指令专门用于支持高级 SSL 配置。

- 在 LibreSSL 中,* 可以 * 使用 `ssl_ciphers` 配置 TLS 1.3 密码套件。
- 在 BoringSSL 中, 根本无法配置 TLS 1.3 密码套件。

ssl_client_certificate

语法	`ssl_client_certificate` *file*;
默认值	—
上下文	stream, server

指定一个包含 PEM 格式受信任 CA 证书的文件，用于验证 客户端证书，以及在启用*ssl_stapling* 时验证 OCSP 响应。

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	stream, server

设置任意 OpenSSL 配置 命令。

备注

使用 OpenSSL 1.0.2 或更高版本时支持该指令。要使用 OpenSSL 配置 TLS 1.3 密码套件，请使用 `ciphersuites` 命令。

可以在同一级别指定多个 *ssl_conf_command* 指令：

```
ssl_conf_command Options PrioritizeChaCha;
ssl_conf_command Ciphersuites TLS_CHACHA20_POLY1305_SHA256;
```

仅当当前级别没有定义 *ssl_conf_command* 指令时，才会从上一级配置继承这些指令。

警告

直接配置 OpenSSL 可能导致意外行为。

ssl_crl

语法	`ssl_crl` *file*;
默认值	—
上下文	stream, server

指定一个包含 PEM 格式吊销证书 (CRL) 的文件，用于验证 客户端证书。

ssl_dhparam

语法	`ssl_dhparam` *file*;
默认值	—
上下文	stream, server

指定用于 DHE 密码套件的 DH 参数文件。

警告

默认情况下不设置任何参数, 因此不会使用 DHE 密码套件。

ssl_early_data

语法	`ssl_early_data on | off;`
默认值	`ssl_early_data off;`
上下文	stream, server

启用或禁用 TLS 1.3 早期数据。

备注

使用 OpenSSL 1.1.1 或更高版本或 BoringSSL 时支持该指令。

ssl_ecdh_curve

语法	`ssl_ecdh_curve` *curve*;
默认值	`ssl_ecdh_curve auto;`
上下文	stream, server

指定用于 ECDHE 密码套件的曲线。

备注

使用 OpenSSL 1.0.2 或更高版本时, 可以指定多条曲线, 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, 或在旧版本中使用 prime256v1。

备注

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，重要的是在证书中包含使用的曲线。

ssl_encrypted_hello_key

Added in version 1.11.0.

语法	`ssl_encrypted_hello_key` *file*;
默认值	—
上下文	stream, server

指定包含 PEM 格式 ECH 私钥和 ECHConfigList 的文件。该指令可以多次指定。需要支持加密客户端问候 (ECH) 的 OpenSSL 或 BoringSSL 构建；否则不支持。

ssl_handshake_timeout

语法	`ssl_handshake_timeout` *time*;
默认值	`ssl_handshake_timeout 60s;`
上下文	stream, server

指定 SSL 握手完成的超时时间。

ssl_ntls

语法	`ssl_ntls on | off;`
默认值	`ssl_ntls off;`
上下文	stream, server

使用 TongSuo 库启用服务器端 NTLS 支持。

```
listen ... ssl;
ssl_ntls  on;
```

备注

Angie 必须使用 `--with-ntls` 构建选项进行构建，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;
默认值	ssl_ocsp off;
上下文	stream, 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。

要使 OCSP 验证正常工作,:ref:*s_ssl_verify_client* 指令应设置为 `on` 或 `optional`。

要解析 OCSP 响应器主机名, 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*];
默认值	ssl_ocsp_cache off;
上下文	stream, server

设置用于存储 OCSP 验证客户端证书状态的缓存的名称和大小。缓存在所有工作进程之间共享。具有相同名称的缓存可以在多个虚拟服务器中使用。

`off` 参数禁止使用缓存。

ssl_ocsp_responder

语法	ssl_ocsp_responder *uri*;
默认值	—
上下文	stream, server

覆盖证书扩展 "Authority Information Access" 中指定的 OCSP 响应器 URI, 用于客户端证书的验证。

仅支持 `http://` OCSP 响应器:

```
ssl_ocsp_responder http://ocsp.example.com/;
```

ssl_password_file

语法	ssl_password_file *file*;
默认值	—
上下文	stream, server

指定包含密钥 密码短语的文件, 其中每个密码短语单独占一行。加载密钥时依次尝试这些密码短语。

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;

        # 命名管道也可以用来代替文件
        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	stream, server

指定在使用 SSLv3 和 TLS 协议时, 服务器密码套件应优先于客户端密码套件。

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	stream, server

启用指定的协议。

备注

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	stream, server

设置存储会话参数的缓存类型和大小。缓存可以是以下任意类型:

`off`	严格禁止使用会话缓存:Angie 明确告知客户端会话不能被重用。
`none`	温和地禁止使用会话缓存:Angie 告知客户端会话可以被重用, 但实际上不在缓存中存储会话参数。
`builtin`	OpenSSL 内置缓存; 仅由一个工作进程使用。缓存大小以会话数指定。如果未给出大小, 则等于 20480 个会话。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; 一兆字节可以存储约 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 会话票证密钥, 除非使用*ssl_session_ticket_key* 指令显式配置。

两种缓存类型可以同时使用, 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*;
默认值	—
上下文	stream, server

设置包含用于加密和解密 TLS 会话票证的密钥的文件。如果必须在多个服务器之间共享相同的密钥, 则此指令是必需的。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 会话票证。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, 将使用 AES256(用于 80 字节密钥) 或 AES128(用于 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	stream, server

启用或禁用通过 TLS 会话票证 进行会话恢复。

ssl_session_timeout

语法	`ssl_session_timeout` *time*;
默认值	`ssl_session_timeout 5m;`
上下文	stream, 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;`
默认值	`ssl_stapling off;`
上下文	http, server

启用或禁用服务器 装订 OCSP 响应。示例:

```
ssl_stapling on;
resolver 127.0.0.53;
```

要使 OCSP 装订正常工作, 应该知道服务器证书颁发者的证书。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该存在于*ssl_trusted_certificate* 文件中。

警告

为了解析 OCSP 响应器主机名, 还应该指定*resolver* 指令。

ssl_stapling_file

语法	`ssl_stapling_file` *file*;
默认值	—
上下文	http, server

设置后, 装订的 OCSP 响应将从指定的文件中获取, 而不是查询服务器证书中指定的 OCSP 响应器。

该文件应该是 DER 格式, 由 `openssl ocsp` 命令生成。

ssl_stapling_responder

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

覆盖证书扩展 "Authority Information Access" 中指定的 OCSP 响应器的 URI。

仅支持 `http://` OCSP 响应器:

```
ssl_stapling_responder http://ocsp.example.com/;
```

ssl_stapling_verify

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

启用或禁用服务器对 OCSP 响应的验证。

要使验证正常工作, 应该使用*ssl_trusted_certificate* 指令将服务器证书颁发者的证书、根证书和所有中间证书配置为受信任的证书。

ssl_trusted_certificate

语法	`ssl_trusted_certificate` *file*;
默认值	—
上下文	stream, server

指定一个包含 PEM 格式受信任 CA 证书的文件, 用于验证 客户端证书。

与*ssl_client_certificate* 设置的证书集不同, 这些证书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	stream, server

启用客户端证书验证。验证结果存储在*$ssl_client_verify* 变量中。如果在客户端证书验证期间发生错误，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，如果证书存在则进行验证。
`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。这适用于由 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` *number*;
默认值	`ssl_verify_depth 1;`
上下文	stream, server

设置客户端证书链中的验证深度。

内置变量

`stream_ssl` 模块支持以下变量：

$ssl_alpn_protocol

返回在 SSL 握手期间通过 ALPN 选择的协议，否则返回空字符串。

$ssl_cipher

返回已建立的 SSL 连接所使用的加密套件名称。

$ssl_ciphers

返回客户端支持的加密套件列表。已知的加密套件以名称列出，未知的以十六进制显示，例如：

```
AES128-SHA:AES256-SHA:0x00ff
```

备注

仅在使用 OpenSSL 1.0.2 或更高版本时完全支持该变量。对于旧版本，该变量仅适用于新会话，并且仅列出已知的加密套件。

`$ssl_client_cert`

返回已建立的 SSL 连接的客户端证书 (PEM 格式), 除第一行外的每一行都以制表符开头。

`$ssl_client_fingerprint`

返回已建立的 SSL 连接的客户端证书的 SHA1 指纹。

`$ssl_client_i_dn`

根据 RFC 2253 返回已建立的 SSL 连接的客户端证书的" 颁发者 DN" 字符串。

`$ssl_client_raw_cert`

返回已建立的 SSL 连接的客户端证书 (PEM 格式)。

`$ssl_client_s_dn`

根据 RFC 2253 返回已建立的 SSL 连接的客户端证书的" 主题 DN" 字符串。

`$ssl_client_serial`

返回已建立的 SSL 连接的客户端证书的序列号。

`$ssl_client_sigalg`

返回已建立的 SSL 连接的客户端证书的 签名算法。

备注

仅在使用 OpenSSL 3.5 或更高版本时支持该变量。对于旧版本, 该变量值将为空字符串。

备注

该变量仅适用于新会话。

`$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剩余天数。

`$ssl_client_v_start`

返回客户端证书的起始日期。

`$ssl_client_verify`

返回客户端证书验证的结果:`SUCCESS`、`FAILED:reason`, 如果证书不存在则返回 `NONE`。

`$ssl_curve`

返回 SSL 握手密钥交换过程中协商使用的曲线。已知的曲线以名称列出, 未知的以十六进制显示, 例如:

prime256v1

备注

仅在使用 OpenSSL 3.0 或更高版本时支持该变量。对于旧版本, 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

备注

仅在使用 OpenSSL 1.0.2 或更高版本时支持该变量。对于旧版本, 该变量值将为空字符串。

该变量仅适用于新会话。

`$ssl_early_data`

如果使用了 TLS 1.3 早期数据 且握手未完成, 则返回"1", 否则返回""。

`$ssl_encrypted_hello`

Added in version 1.11.0.

如果使用了加密客户端问候 (ECH), 则返回"1", 否则返回""。

`$ssl_protocol`

返回已建立的 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`

返回已建立的 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_sigalg`

返回已建立的 SSL 连接的服务器证书的 签名算法。

备注

仅在使用 OpenSSL 3.5 或更高版本时支持该变量。对于旧版本，该变量值将为空字符串。

备注

该变量仅适用于新会话。

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_alpn_protocols

客户端通过 ALPN 广告的协议列表。这些值用逗号分隔。

$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。

Upstream

提供用于描述服务器组的上下文, 可在*proxy_pass* 指令中使用。

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

backup_switch (PRO)

Added in version 1.10.0: PRO

语法	`backup_switch permanent`[=*time*];
默认值	—
上下文	upstream

该指令启用从 活动组而不是主组开始服务器选择的能力，即从之前成功找到服务器的组开始。如果在活动组中无法为下一个请求找到服务器，并且搜索移至备份组，该备份组将成为活动组，后续请求将首先定向到该组中的服务器。

如果定义了 `permanent` 参数但没有时间 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，负载均衡器将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 {
    zone media_backend 1m;
    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 分钟过期后，负载均衡器重新检查主服务器，如果它们正常工作，则再次使其成为活动组。

feedback (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，该值根据 *variable* 值随时间变化，并受可选条件约束。

可以指定以下参数：

variable	从中获取反馈值的变量。它应该表示性能或健康指标；假定由服务器提供。 该值在服务器的每个响应中进行评估，并根据 `inverse` 和 `factor` 设置纳入移动平均值。
`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加权因子。有效值为 0 到 99 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% 的先前值和仅 10% 的新值。
`account`	指定一个条件变量，用于控制如何在计算中统计连接。仅当条件变量不等于 `""` 或 `"0"` 时，才使用反馈值更新平均值。 **备注** 默认情况下，来自探测 的流量不包含在计算中；将*$upstream_probe* 变量与 `account` 结合使用可以包含它们或甚至排除其他所有内容。

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";
}
```

此配置根据各个会话中使用的协议按反馈级别对服务器进行分类，并在*$upstream_probe* 上添加条件，仅统计 `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

为组指定负载均衡方法，其中客户端-服务器映射使用哈希键值确定。键可以包含变量。请注意，从组中添加或删除任何服务器可能会导致大多数键重新映射到不同的服务器。该方法与 Perl Cache::Memcached 库兼容。

使用示例：

```
hash $remote_addr;
```

当使用解析为多个 IP 地址的域名时（例如，使用 `resolve` 参数），服务器不会对接收到的地址进行排序，因此它们的顺序可能在不同服务器之间有所不同，这会影响客户端分布。为确保一致的分布，请使用 `consistent` 参数。

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。该方法确保当从组中添加或删除服务器时，只有最少数量的键会重新映射到其他服务器。将该方法用于缓存服务器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

为组指定负载均衡方法，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，则按照它们的权重以循环（轮询）方式选择它们。

least_time

语法	`least_time` `connect` | `first_byte` | `last_byte` [`factor=`*number*] [`account=`*condition_variable*];
默认值	—
上下文	upstream

为组指定负载均衡方法，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；响应时间越低，服务器将接收到的连接就越多。

connect	该指令考虑建立连接的平均时间。
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。

factor	执行与*response_time_factor* 相同的功能，如果指定了该参数，则会覆盖它。
account	指定一个条件变量，用于控制在计算中考虑哪些连接。仅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 *(PRO)* 不包含在计算中；将*$upstream_probe* 变量与 account 结合使用可以包含它们，甚至排除其他所有内容。

当前值在 API 的*upstream* 指标 中，以服务器的 health 对象中的 connect_time、first_byte_time 和 last_byte_time 形式呈现。

no_port (PRO)

Added in version 1.12.0: PRO

语法	no_port;
默认值	—
上下文	upstream

显式将服务器组标记为无端口：其*server* 地址不带端口指定，连接端口改由引用该组的每个*proxy_pass* 指令提供。

该指令适用于不静态列出任何服务器、而是通过配置 *API* 动态填充的组。此类组需要一个共享内存区域，且默认是有端口的，因此 no_port 使其成为无端口。这样一个无端口的组便可重复用于代理到不同端口，而无需重复服务器列表：

```
upstream backend {
    zone backend 1m;
    no_port;
}

server {
    listen 8443;
    proxy_pass backend:8443;
}

server {
    listen 8080;
```

```
    proxy_pass backend:8080;
}
```

当组的*server* 地址是不带端口的域名或 IP 地址时，即使没有该指令，该组也会被视为无端口。`no_port` 指令显式声明此属性，并拒绝包含端口的地址。

无端口的组只接受不带端口的域名和 IP 地址；不允许使用 UNIX 域套接字地址以及使用 `service=` 参数进行 DNS SRV 解析的服务器。同一组中的域名和 IP 地址必须全部包含端口或全部省略端口；不允许混用端口。

random

语法	`random` [`two`];
默认值	—
上下文	upstream

为组指定负载均衡方法，将连接传递给随机选择的服务器，同时考虑服务器权重。

如果指定了可选的 `two` 参数，Angie 会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*，它将连接传递给活动连接数最少的服务器。

response_time_factor

语法	`response_time_factor` *number*;
默认值	`response_time_factor 90;`
上下文	upstream

为*least_time* 负载均衡方法指定在使用 指数加权移动平均 公式计算平均响应时间时，**先前**值的平滑因子。

指定的 *number* 越大，新值对平均值的影响就越小；如果指定为 `90`，将采用 90% 的先前值，仅采用 10% 的新值。有效值范围为 0 到 99（含）。

当前计算结果在 API 的*upstream* 指标 中，以服务器的 `health` 对象中的 `connect_time` （建立连接的时间）、`first_byte_time` （接收响应第一个字节的时间）和 `last_byte_time` （接收完整响应的时间）形式呈现。

备注

计算中仅考虑成功的响应；什么构成不成功的响应由*proxy_next_upstream* 指令确定。

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，或者指定为 `unix:` 前缀后的 UNIX 域套接字路径。解析为多个 IP 地址的域名会一次定义多个服务器。对于域名或 IP 地址，端口是必需的，但在无端口 组（PRO）中除外，此时端口被省略，改由*proxy_pass* 指令提供。

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。如果服务器组不在共享内存 中, 则该限制对每个工作进程生效。

`max_fails=`*number* —设置在*fail_timeout* 设置的持续时间内与服务器通信的失败尝试次数, 达到该次数后将服务器视为不可用; 然后在相同的持续时间后重试。

这里, 失败尝试是指与服务器建立连接时的错误或超时。

备注

如果组中的 `server` 指令解析为多个服务器, 其 `max_fails` 设置将单独应用于每个服务器。

如果在解析所有 `server` 指令后,upstream 只包含一个服务器, 则 `max_fails` 设置无效并将被忽略。

`max_fails=1`	默认尝试次数。
`max_fails=0`	禁用尝试次数统计。

`fail_timeout=`*time* —设置时间段, 在此期间应发生指定次数的与服务器通信失败尝试 (*max_fails*) 才将服务器视为不可用。然后服务器在相同的时间段内保持不可用状态, 之后才会重试。

默认情况下, 此值设置为 10 秒。

备注

如果组中的 `server` 指令解析为多个服务器, 其 `fail_timeout` 设置将单独应用于每个服务器。

如果在解析所有 `server` 指令后,upstream 只包含一个服务器, 则 `fail_timeout` 设置无效并将被忽略。

backup	将服务器标记为备份服务器。当主服务器不可用时，它将接收请求。
down	将服务器标记为永久不可用。
drain	将服务器标记为排空状态；这意味着它只接收来自之前通过*sticky* 绑定的会话的请求。否则其行为类似于 down。

警告

backup 参数不能与*hash* 和*random* 负载均衡方法一起使用。

down 和 drain 参数互斥。

resolve	启用监控与域名对应的 IP 地址列表的变化，无需重新加载配置即可更新。该组必须位于共享内存区域 中；还必须定义*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，还必须指定 *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，则根据 RFC 标准形成名称：服务名称前加 _ 前缀，然后在点后添加 _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来解析 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。 • 最高优先级的 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 server 设置了 backup，最高优先级的 SRV 记录解析为备份服务器，其他记录被忽略。 • 权重类似于 server 指令的 weight 参数。如果指令和 SRV 记录都设置了权重，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;
```

sid=*id*	设置组中的服务器 ID。如果未指定该参数，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。

slow_start=*time*	设置服务器在恢复服务时，使用轮询 或*least_conn* 负载均衡方法恢复其权重的时间。 如果设置了该参数，并且服务器在根据*max_fails* 和*upstream_probe (PRO)* 判定失败后再次被认为健康，服务器将在指定的时间段内逐渐恢复其指定的权重。 如果未设置该参数，在类似情况下，服务器将立即以其指定的权重开始工作。

备注

如果 upstream 中只指定了一个 `server`，`slow_start` 将不起作用并被忽略。

state (PRO)

语法	`state` *file*;
默认值	—
上下文	upstream

指定持久化存储上游服务器列表的 *file*。从 我们的软件包安装时，会创建一个具有适当权限的专用目录 `/var/lib/angie/state/`（FreeBSD 上为 `/var/db/angie/state/`）用于存储此类文件，因此您只需在配置中添加文件名：

```
upstream backend {

    zone backend 1m;
    state /var/lib/angie/state/<文件名>;
}
```

这里的服务器列表格式类似于 `server`。每当通过配置 API 在*/config/stream/upstreams/* 部分修改服务器时，文件内容都会更改。该文件在 Angie 启动或配置重载时读取。

警告

要在 `upstream` 块中使用 `state` 指令，其中不应有 `server` 指令，但需要共享内存区（*zone*）。

sticky

在 1.10.0 版本发生变更: PRO

在 1.11.0 版本发生变更: PRO

语法	`sticky route` *value...*; `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* [`remote_uri=`*uri*]; (PRO)
默认值	—
上下文	upstream

根据第一个参数中指定的模式，配置客户端与上游服务器之间的粘性会话。要逐步将带有 `sticky` 的服务器退出轮换，可以在*server* 块中使用 `drain` 选项。

警告

`sticky` 指令必须出现在所有负载均衡方法指令之后，否则将无法工作。

`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 Angie 可访问的连接属性中。它的灵活性较低，因为它依赖于预定义的值，但如果已经在使用此类标识符，则更适合。

在这里，建立连接时，上游服务器可以为客户端分配一个路由，并以双方都知道的方式返回其标识符。路由标识符应使用*server* 指令的*sid* 参数的值。请注意，如果指定了*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连接必须包含服务器颁发的标识符，并以这样的方式使其最终出现在 Angie 变量中。

指令参数指定可能包含变量的字符串，用于路由。要选择传入连接定向到的服务器，使用第一个非空值；然后将其与*server* 指令的*sid* 参数进行比较。如果服务器选择失败或所选服务器无法接受连接，将根据配置的负载均衡方法选择另一台服务器。

这里 Angie 在 `$route` 变量中查找路由标识符，该变量根据*$ssl_preread_server_name* 接收其值（注意必须启用*ssl_preread*）：

```
stream {

    map $ssl_preread_server_name $route {

        a.example.com            a;
        b.example.com            b;
        default                  "";
    }

    upstream backend {

        zone backend 1m;

        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

在此模式下，使用动态生成的密钥将客户端绑定到特定的上游服务器；此模式更加灵活，因为它动态分配服务器，将会话存储在共享内存区域中，并支持多种传递会话标识符的方式。

在这里，会话是基于来自上游服务器的连接属性创建的。`create` 和 `lookup` 参数列出了指定如何创建新会话和查找现有会话的变量。这两个参数都可以多次使用。

会话标识符是使用 `create` 指定的第一个非空变量的值；例如，这可以是上游服务器名称。

会话存储在共享内存区域中；其名称和大小由 `zone` 参数指定。如果会话在 `timeout` 指定的时间 内未被访问，则会被删除。默认值为 1 小时。

默认情况下，Angie 通过在每次使用会话时更新最后访问时间戳来延长会话生命周期。`norefresh` 参数改变了这种行为：会话严格按超时时间过期，即使正在使用中。

希望使用会话的客户端的后续连接必须包含其标识符。`lookup` 参数使用指定的变量列表在连接中搜索会话标识符，在第一个非空变量处停止。如果未找到任何内容，则该请求被视为新请求。找到的标识符的值与共享内存中的会话进行匹配。如果服务器选择失败或所选服务器无法处理连接，将根据配置的负载均衡方法选择另一台服务器。

`connect` 参数允许在与上游服务器建立连接后立即创建会话。如果没有它，会话仅在连接处理完成后创建。（对于 UDP 连接，会话在服务器选择后立即创建。）

在示例中，Angie 使用 *$rdp_cookie* 变量创建和查找会话：

```
stream {

    upstream backend {

        zone backend 1m;

        server 127.0.0.1:3390;
        server 127.0.0.1:3391;

        sticky learn lookup=$rdp_cookie create=$rdp_cookie zone=sessions:1m;
    }

    server {

        listen 127.0.0.1:3389;

        rdp_preread on;

        proxy_pass backend;
    }
}
```

带 `remote_action` 的 `learn` 模式（PRO 1.10.0+）

`remote_action` 和 `remote_result` 参数允许使用远程会话存储（PRO）动态分配和管理会话标识符。

与带 `zone` 的 `learn` 模式不同，此模式不在本地缓存会话，而是为每个连接查询远程存储。

`remote_action` 参数必须指向*client* 上下文中的 `location`。`remote_uri` 参数指定客户端 HTTP 请求到指定 `location` 的 URI。默认情况下，它是 `/`。`remote_uri` 值可以包含变量。

此模式的一般操作原理如下：如果在本地未找到会话标识符，Angie 会向 `remote_action` 参数指定的远程存储发送同步子请求。

当新连接到达时，Angie 执行以下操作：

- 首先，从 `lookup` 列表中的第一个非空变量中提取会话标识符。如果所有变量都为空，则使用正常的负载均衡算法，不使用粘性会话。
- 然后 Angie 向 `remote_action` 参数指定的远程存储发送同步 HTTP 子请求，该请求应以存储能理解的格式包含：
  - 来自 `lookup` 参数的 会话标识符（在配置中，这是*$sticky_sessid* 变量）；
  - 预选 服务器的标识符：`server` 指令中 `sid=` 参数的值（如果指定），或服务器名称的 MD5 哈希（在配置中，这是*$sticky_sid* 变量）。

  `$sticky_sessid` 和 `$sticky_sid` 变量会自动以 `stream_` 前缀导出到 HTTP 上下文：`$stream_sticky_sessid`、`$stream_sticky_sid`。这允许直接在 HTTP 指令中使用它们，例如通过*proxy_set_header* 使用 HTTP 头。
- 远程存储处理请求并返回 HTTP 响应：

  代码为 200、201 或 204 的响应确认所选服务器。远程存储可以同时在 HTTP 头或响应正文（PRO）中返回替代服务器标识符；可以通过 `remote_result` 提取它。

  当从存储接收到任何其他 HTTP 代码时（包括网络错误和超时）或不存在的服务器标识符，Angie 使用最初选择的服务器。

服务器标识符通过 `remote_result` 参数从远程存储响应中提取：它可以指定带有 `upstream_http_` 前缀的变量，这些变量由 Angie 自动创建，用于访问来自远程存储的 HTTP 响应头，或使用*$sent_body* 来使用响应正文。例如，此类响应中的 `X-Sid: server1` 头在 `$upstream_http_x_sid` 变量中可访问，其值为 `server1`。

下面是一个简化的配置示例。远程存储在 `X-Sticky-Sid` 头中返回服务器标识符，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；
            # 它将这些变量以 stream_* 前缀添加到 HTTP 上下文
```

```
            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 {

        zone u 1m;

        server 127.0.0.1:8081 sid=backend-01;
        server 127.0.0.1:8082 sid=backend-02;

        sticky learn lookup=$remote_addr             # stream 变量
        remote_action=@sticky_client1                # 来自 client 块的 location
        remote_result=$upstream_http_x_sticky_sid    # HTTP 变量
        remote_uri=/foo;                             # 默认为 /
    }

    server {

        listen 127.0.0.1:8080;
        proxy_pass u;
    }
}
```

这里，来自远程存储的响应如下：

```
HTTP/1.1 200 OK
...
X-Sticky-Sid: backend-01
X-Session-Info: active
```

两个变量变为可用：

- `$upstream_http_x_sticky_sid`，值为 `backend-01`；
- `$upstream_http_x_session_info`，值为 `active`。

由于 `$upstream_http_x_sticky_sid` 变量在 `remote_result` 参数中指定，其值将被用于选择具有 `sid=backend-01` 的服务器。

sticky 指令会考虑*upstream* 中服务器的状态：

- 标记为 down 或由于故障而暂时不可用的服务器将被排除在选择之外。
- 已达到最大连接数的服务器（使用 max_conns 时）将被暂时跳过。
- 带有 drain 选项的服务器在 sticky 模式下当标识符匹配时可以被选择用于创建新会话。
- 如果先前不可用的服务器恢复，sticky 会自动恢复使用它。

sticky 的行为可以通过*sticky_secret* 和*sticky_strict* 指令进一步配置。如果 sticky 无法选择服务器或服务器不可用，请求将根据所选的负载均衡方法进行处理，除非启用了 sticky_strict 指令。在 sticky_strict on; 模式下，请求将被拒绝并返回错误。

在 sticky 指令的 zone 参数中指定的共享内存区域不能在不同的 upstream 组之间共享；每个组必须使用自己的区域。

sticky_secret

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 route 模式下的*sticky* 指令。*String* 可以包含变量，例如 $remote_addr：

```
upstream backend {
    zone backend 1m;
    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;
}
```

盐值添加在哈希值之后；要独立验证哈希机制：

```
$ echo -n "<VALUE><SALT>" | md5sum
```

sticky_strict

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 将向客户端返回连接错误，而不是回退到另一个可用的服务器，后者是未找到匹配服务器时的默认行为。

upstream

语法	upstream *name* { ... }
默认值	—
上下文	stream

描述一个服务器组。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 {
    zone backend 1m;
    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，连接使用加权轮询均衡方法在服务器之间分配。在上面的示例中，每 7 个连接将按如下方式分配:5 个连接到 backend1.example.com:1935，第二个和第三个服务器各 1 个连接。该分配是平滑的: 权重较高的服务器的连接会分散到整个轮询周期中，而不是集中成一批连续发送。

如果在与服务器通信期间发生错误，连接将传递给下一个服务器，依此类推，直到尝试所有正常工作的服务器。如果与所有服务器的通信都失败，连接将被关闭。

备注

默认情况下，偶尔出现失败尝试但未达到*max_fails* 阈值的服务器会暂时收到较少的连接份额，并在后续连接中逐渐恢复到完整份额。这与*slow_start* 不同: 后者仅在服务器被标记为不可用并随后恢复后，才使其逐步回到完整权重。

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。在这种情况下，只需指定一次大小即可。

备注

只有在配置的 `size` 不变时，重载才会保留区域内容。任何大小变更——增大或减小——都会导致区域被重新创建为空。

备注

仅在配置了此区域时才会收集上游指标。如未配置，该组将不会出现在*/status/stream/upstreams/<upstream>*、「*TCP/UDP Upstreams*」小部件 及*Prometheus* 输出中，且不会输出任何警告信息。

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), 或者服务器名称的 MD5 哈希值。

`$upstream_addr`

存储上游服务器的 IP 地址和端口, 或 UNIX 域套接字的路径。如果在代理过程中联系了多个服务器, 它们的地址用逗号分隔, 例如:

192.168.1.1:1935, 192.168.1.2:1935, unix:/tmp/sock

如果无法选择服务器, 该变量保留服务器组 的名称。

`$upstream_bytes_received`

从上游服务器接收的字节数。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。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_connect_time`

连接到上游服务器的时间; 时间以秒为单位保存, 具有毫秒分辨率。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_current_addr (PRO)

为当前连接尝试所选择的被代理服务器的地址；与*$upstream_addr* 不同，后者会累积每次重试所尝试的每个服务器的地址，而该变量始终只保存单个地址，并且在服务器被选择后立即可用，包括在对*proxy_bind* 变量求值期间。

$upstream_first_byte_time

接收第一个数据字节的时间；时间以秒为单位保存，具有毫秒分辨率。多个连接的时间用逗号分隔，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，具有毫秒分辨率。多个连接的时间用逗号分隔，类似于*$upstream_addr* 变量中的地址。

$upstream_sticky_status

粘性连接的状态。

""	连接路由到未启用 sticky 的服务器组。
NEW	连接不包含粘性信息。
HIT	包含粘性信息的连接路由到所需的服务器。
MISS	包含粘性信息的连接路由到由负载均衡算法选择的服务器。

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 Probe

该模块为*stream_upstream* 实现了主动健康探测。

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 Semantics)，HTTP 头部必须使用 CRLF 序列 (\r\n) 分隔，而不是仅使用 \n。

指令

upstream_probe (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*];
默认值	—
上下文	server

为*upstream* 组中的服务器定义主动健康探测，该组在与 upstream_probe 指令位于同一 server 上下文中的*proxy_pass* 指令中指定。

如果对服务器的请求成功，则服务器通过探测，需考虑 upstream_probe 指令的所有参数设置以及影响定义它的 server 上下文如何使用上游的所有参数，包括*proxy_next_upstream* 指令。

要使用探测功能，上游必须具有共享内存区域 (*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。
port	探测请求的备用端口号。
interval	探测之间的间隔。默认值—5s。
test	探测的条件，定义为变量字符串。如果变量替换结果为 "" 或 "0"，则探测未通过。
essential	如果设置，将检查服务器的初始状态，因此服务器在通过探测之前不会接收客户端请求。
persistent	设置此参数需要首先启用 essential；在配置重载 之前被认为健康的 persistent 服务器无需首先通过此探测即可开始接收请求。
fails	使服务器变为不健康的连续失败探测次数。默认值—1。
passes	使服务器变为健康的连续通过探测次数。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	探测模式，取决于服务器的健康状态： • always —无论服务器状态如何都进行探测； • idle —探测影响不健康的服务器以及自上次客户端请求以来已过 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	为探测发送的数据：带有 data: 前缀的内联数据或文件路径（绝对路径或相对于 /usr/local/angie/ 的路径）。 使用文件时： • 工作进程 在每次访问时打开并读取文件；内容不会缓存在内存中。 • 文件更改时不需要重新加载配置；下次访问时将读取新内容。 • 所需的访问权限：文件为 644，目录为 755。 • 使用移动命令 (mv) 更新文件，而不是直接编辑。

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";
}
```

探测操作的详细信息：

- 最初，服务器在通过为其配置的 所有 `essential` 探测之前不会接收客户端请求，如果配置已重载且服务器在此之前被认为健康，则跳过 `persistent` 探测。如果没有此类探测，则认为服务器健康。
- 如果为服务器配置的 任何探测达到 `fails` 或服务器达到*max_fails*，则认为服务器不健康且不会接收客户端请求。
- 要使不健康的服务器再次被认为健康，为其配置的 所有探测必须达到各自的 `passes`；之后，还会考虑*max_fails*。

upstream_probe_timeout (PRO)

语法	`upstream_probe_timeout` *time*;
默认值	`upstream_probe_timeout 50s;`
上下文	server

设置使用*upstream_probe (PRO)* 指令配置的健康探测与服务器建立的连接的最大空闲时间；如果超过此限制，连接将被关闭。

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* 配置的主动探测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：包括定义服务器块、流量路由、配置代理、SSL/TLS 支持，以及管理流式服务（如数据库、DNS 和其他基于 TCP 和 UDP 运行的协议）的连接。

本节中的其他模块扩展了此功能，允许您灵活地配置和优化流服务器以适应各种场景和需求。

当 从源代码构建时，此模块默认不会被构建；应使用 `--with-stream` 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。

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

在 1.10.0 版本发生变更.

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`]];
默认值	—
上下文	server

设置服务器将接受连接的套接字的 *address* 和 *port*。可以只指定 *port*，这样 Angie 将监听所有可用的 IPv4（以及 IPv6，如果已启用）接口。地址也可以是主机名，例如：

```
listen 127.0.0.1:12345;
listen *:12345;
listen 12345;      # 与 *:12345 相同
listen localhost:12345;
```

IPv6 地址用方括号指定：

```
listen [::1]:12345;
listen [::]:12345;
```

端口范围通过用连字符分隔的第一个和最后一个端口来指定：

```
listen 127.0.0.1:12345-12399;
listen 12345-12399;
```

UNIX 域套接字使用 `unix:` 前缀指定：

```
listen unix:/var/run/angie.sock;
```

备注

不同的服务器必须监听不同的 *address:port* 对。

`ssl`	允许指定在此端口上接受的所有连接应在 SSL 模式下工作。
`udp`	配置监听套接字以处理数据报。为了在同一会话中处理来自相同地址和端口的数据包，还应指定 *reuseport* 参数。
`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。

`setfib=`*number*	为监听套接字设置关联的路由表 FIB（`SO_SETFIB` 选项）。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。 **警告** 除非服务器能够处理多次接收 相同的带数据的 SYN 数据包，否则不要启用此功能。
`backlog=`*number*	设置 `listen()` 调用中的 `backlog` 参数，该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（`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()` 之前对其进行过滤。这仅在 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()`（`TCP_DEFER_ACCEPT` 套接字选项）。
`bind`	此参数指示为给定的 *address:port* 对进行单独的 `bind()` 调用。事实上，如果有多个具有相同 *port* 但不同地址的 `listen` 指令，并且其中一个 `listen` 指令监听给定端口的所有地址（**:port*），Angie 将只 `bind()` 到 **:port*。应该注意的是，在这种情况下将进行 `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* 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）确定监听通配符地址 *[::]* 的 IPv6 套接字是仅接受 IPv6 连接还是同时接受 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。目前仅在 Linux 3.9+、DragonFly BSD 和 FreeBSD 12+ 上有效。 **警告** 不当使用此选项可能会带来安全隐患。
`multipath`	启用通过 多路径 *TCP* <*https://en.wikipedia.org/wiki/Multipath_TCP*>‘__（*MPTCP*）协议接受连接，从 *5.6* 版本开始在 *Linux* 内核中支持。此参数与 *:samp:‘udp* **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，则套接字将使用操作系统的设置
`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 keepalive 参数。在这些系统上（目前包括 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
```

将空闲超时（`TCP_KEEPIDLE`）设置为 30 分钟，将探测间隔（`TCP_KEEPINTVL`）保留为系统默认值，并将探测次数（`TCP_KEEPCNT`）设置为 10 次探测。

preread_buffer_size

语法	`preread_buffer_size` *size*;
默认值	`preread_buffer_size 16k;`
上下文	stream, server

指定预读 缓冲区的大小。

preread_timeout

语法	`preread_timeout` *timeout*;
默认值	`preread_timeout 30s;`
上下文	stream, server

指定预读 阶段的超时时间。

proxy_protocol_timeout

语法	`proxy_protocol_timeout` *timeout*;
默认值	`proxy_protocol_timeout 30s;`
上下文	stream, server

指定读取 PROXY 协议头完成的超时时间。如果在此时间内未传输完整个头部，则连接将被关闭。

resolver

语法	`resolver` *address* ... [`valid=`*time*] [`ipv4=on` | `off`] [`ipv6=on` | `off`] [`status_zone=`*zone*];
默认值	—
上下文	stream, server, upstream

配置用于将上游服务器名称解析为地址的域名服务器，例如：

```
resolver 127.0.0.53 [::1]:5353;
```

地址可以指定为域名或 IP 地址，并可选端口。如果未指定端口，则使用端口 53。域名服务器以轮询方式查询。

备注

建议使用本地可信解析器，例如 `127.0.0.53` (systemd-resolved)，而非公共解析器（如 `8.8.8.8`）。公共解析器会将 DNS 查询暴露给第三方，并增加缓存投毒攻击的风险。

备注

该指令值会被嵌套块继承，并可在其中根据需要覆盖。在单个块内，该指令只能指定一次。如果重复指定，则最后的定义生效。

默认情况下，Angie 使用响应的 TTL 值缓存答案。如果未指定 `resolver` 指令且不执行动态 DNS 查询（例如，在*Proxy* 中使用固定名称而不使用变量时），则不需要指定解析器：名称将在启动时使用系统解析器进行解析。可选的 `valid` 参数允许覆盖此行为：

`valid`	可选参数允许覆盖响应缓存的有效期

```
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 地址查找
`ipv6=off`	禁用 IPv6 地址查找

`status_zone`	可选参数; 在指定区域中启用 DNS 服务器请求和响应指标的收集 (*/status/resolvers/<zone>*)

小技巧

为防止 DNS 欺骗, 建议在适当安全的可信本地网络中使用 DNS 服务器。

小技巧

在 Docker 中运行时, 使用相应的内部 DNS 服务器地址, 例如 `127.0.0.11`。

resolver_timeout

语法	`resolver_timeout` *time;*
默认值	`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 { ... }`
默认值	—
上下文	stream

设置服务器的配置。

server_name

语法	`server_name` *name ...;*
默认值	`server_name "";`
上下文	server

设置虚拟服务器的名称。

警告

在 `stream` 模块中,:samp:*server_name* 指令基于服务器名称指示 (*SNI*), 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。

配置示例:

```
server {
    listen 443 ssl;
    server_name example.com www.example.com;
    ssl_certificate /etc/angie/cert.pem;
    ssl_certificate_key /etc/angie/key.pem;
}
```

第一个名称成为主服务器名称。

服务器名称可以包含星号 (*) 来替换名称的第一部分或最后一部分:

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

这些名称称为通配符名称。

您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, 可在其他指令中使用:

```
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, 则会插入机器的主机名。

按名称搜索虚拟服务器时, 如果名称匹配多个指定的变体 (例如, 通配符名称和正则表达式都匹配), 将按以下优先级顺序选择第一个匹配的变体:

- 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 (按配置文件中出现的顺序)

server_names_hash_bucket_size

语法	`server_names_hash_bucket_size` *size*;
默认值	`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*;
默认值	`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*];
默认值	—
上下文	server

分配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* 语法将当前上下文的所有指标聚合在一个共享内存区域中:

```
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, 其值决定区域中连接的分组。所有在替换后产生相同值的连接会被分组在一起。如果替换产生空值, 则不更新指标。
zone	共享内存区域的名称。
count (可选)	用于收集指标的单独组的最大数量。如果新的 *key* 值超过此限制, 它们将被分组到 *zone* 下。 默认值为 1。

在以下示例中，所有具有相同 `$server_addr` 值的连接被分组到 `host_zone` 中。为每个唯一的 `$server_addr` 单独收集指标，直到指标组数量达到 10。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;
    }
}
```

生成的指标在 API 输出中按各个服务器拆分。

备注

仅在设置了 `status_zone` 时才会收集这些指标。如未设置，对应的 server 将不会出现在*/status/stream/server_zones/<zone>*、「*TCP/UDP Zones*」小部件 及*Prometheus* 输出中，且不会输出任何警告信息。

stream

语法	`stream { ... }`
默认值	—
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;`
默认值	`tcp_nodelay on;`
上下文	stream、server

启用或禁用 `TCP_NODELAY` 选项的使用。该选项对客户端连接和到代理服务器的连接都启用。

variables_hash_bucket_size

语法	`variables_hash_bucket_size` *size*;
默认值	`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*;
默认值	`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

核心 stream 模块支持以下内置变量:

$angie_version

Angie 版本

$binary_remote_addr

二进制形式的客户端地址,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 (秒), 精度为毫秒

`$nginx_version`

nginx 版本

`$pid`

工作进程的 PID

`$protocol`

用于与客户端通信的协议:`TCP` 或 `UDP`

`$proxy_protocol_addr`

来自 PROXY 协议头的客户端地址。必须先通过在*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。必须先通过在*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。必须先通过在*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。必须先通过在*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, 该值以十六进制指定, 并且必须以 *0x* 开头:

```
$proxy_protocol_tlv_alpn
$proxy_protocol_tlv_0x01
```

SSL TLV 也可以通过 TLV 类型名称和其数值来访问, 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`) - 使用的密码套件名称
- `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且验证成功则为 0, 否则为非零值

必须先通过在*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。为了避免系统调用,:ref:*s_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, 精度为毫秒

`$status`

会话状态，可以是以下之一：

200	会话成功完成
400	无法解析客户端数据，例如 PROXY 协议头
403	禁止访问，例如当访问受特定客户端地址 限制时
500	内部服务器错误
502	错误网关，例如无法选择或访问上游服务器时
503	服务不可用，例如当访问受连接数 限制时

`$time_iso8601`

ISO 8601 标准格式的本地时间

`$time_local`

通用日志格式的本地时间

3.2.1.4 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/xoauth2/oauthbearer/none
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
```

当使用 XOAUTH2 或 OAUTHBEARER 时,:samp:*Auth-User* 和 `Auth-Pass` 头部包含从初始 SASL 响应中提取的用户身份和持有者令牌。

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‘ 代码将默认添加到 `Auth-Status` 头部。

对于 XOAUTH2 和 OAUTHBEARER, 错误响应还可能包含 `Auth-Error-SASL` 头部。其值作为额外的 SASL 质询发送给客户端 (SMTP:*334*,IMAP/POP3:*+*)。在客户端对 XOAUTH2 响应空响应或对 OAUTHBEARER 响应 `AQ==` 后, 返回来自 `Auth-Status` 的错误。

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`,:samp:*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

在 1.11.0 版本发生变更.

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
xoauth2	AUTH=XOAUTH2
oauthbearer	AUTH=OAUTHBEARER

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,:samp:*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,:ref:*m_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。

指定客户端代理到的 IMAP 后端所支持的扩展是有意义的 (如果这些扩展与身份验证后使用的命令相关，即当 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

在 1.11.0 版本发生变更.

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
`xoauth2`	AUTH=XOAUTH2
`oauthbearer`	AUTH=OAUTHBEARER

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,:samp:*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,:ref:*m_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目标 (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

在 1.11.0 版本发生变更.

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
`xoauth2`	AUTH XOAUTH2
`oauthbearer`	AUTH OAUTHBEARER
`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。

备注

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_compression

语法	`ssl_certificate_compression on | off;`
默认值	`ssl_certificate_compression off;`
上下文	mail, server

启用 TLS 1.3 服务器证书压缩。

备注

该指令在使用 OpenSSL 3.2 或更高版本时受支持；支持的压缩算法列表由库提供。

备注

该指令在使用 BoringSSL 时受支持; 支持的压缩算法列表包括 `zlib`。

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 `store:scheme:id` 来代替 `file`, 用于加载具有指定 *id* 和 OpenSSL 提供程序注册 URI *scheme* 的密钥, 例如 pkcs11。

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。

警告

当使用 OpenSSL 时,:samp:*ssl_ciphers* 指令 * 不会 * 配置 TLS 1.3 的密码。要使用 OpenSSL 调整 TLS 1.3 密码, 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 密码。
- 在 BoringSSL 中, 完全无法配置 TLS 1.3 密码。

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。

备注

该指令在使用 OpenSSL 1.0.2 或更高版本时受支持。要使用 OpenSSL 配置 TLS 1.3 密码, 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。

警告

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。

警告

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。

备注

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 库内置的列表, 或在较旧版本中使用 prime256v1。

备注

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 的密码短语，每个密码短语单独一行。加载密钥时会依次尝试这些密码短语。

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

启用指定的协议。

备注

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类型和大小。缓存可以是以下任意类型:

`off`	严格禁止使用会话缓存:Angie 明确告知客户端会话不可重用。
`none`	温和地禁止使用会话缓存:Angie 告知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 内置缓存; 仅由一个工作进程使用。缓存大小以会话数指定。如果未指定大小, 则等于 20480 个会话。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。缓存大小以字节指定; 一兆字节可以存储约 4000 个会话。每个共享缓存应有一个任意名称。具有相同名称的缓存可以在多个服务器中使用。除非使用*ssl_session_ticket_key* 指令显式配置, 否则它还用于自动生成、存储和定期轮换 TLS 会话票证密钥。

两种缓存类型可以同时使用, 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*;
默认值	—
上下文	mail, server

设置包含用于加密和解密 TLS 会话票证的密钥的文件。如果需要在多个服务器之间共享相同的密钥, 则此指令是必需的。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 会话票证。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(用于 80 字节密钥) 或 AES128(用于 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 会话票证 恢复会话。

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

指定包含 PEM 格式的受信任 CA 证书的文件, 用于验证 客户端证书。

与*ssl_client_certificate* 设置的证书集不同, 这些证书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。这适用于由 Angie 外部的服务执行实际证书验证的情况。证书的内容可以通过发送 到认证服务器的请求访问。

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`	允许对 POP3 使用 STLS 命令, 对 IMAP 和 SMTP 使用 STARTTLS 命令;
`off`	拒绝使用 STLS 和 STARTTLS 命令;
`only`	要求预先进行 TLS 转换。

核心邮件模块实现了邮件代理服务器的基本功能: 包括对 SMTP、IMAP 和 POP3 协议的支持、配置服务器块、邮件请求路由、用户身份验证, 以及用于保护邮件连接的 SSL/TLS 支持。

本节中的其他模块扩展了此功能, 允许您灵活地配置和优化邮件服务器以适应各种场景和需求。

当 从源代码构建时, 此模块默认不会被构建; 应使用 `--with-mail` 构建选项启用它。在来自 我们的软件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;
}

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

在 1.10.0 版本发生变更.

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`]];
默认值	—
上下文	server

设置服务器将接受连接的套接字的 *address* 和 *port*。可以只指定 *port*,Angie 将监听所有可用的 IPv4 接口 (如果启用, 还包括 IPv6)。*address* 也可以是主机名, 例如:

```
listen 127.0.0.1:110;
listen *:110;
listen 110;      # 与 *:110 相同
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;
```

备注

不同的服务器必须监听不同的 *address:port* 对。

`ssl`	指定在此端口上接受的所有连接都应在 SSL 模式下工作。
`proxy_protocol`	指定在此端口上接受的所有连接都应使用 PROXY 协议。获取的信息将传递给身份验证服务器, 并可用于更改客户端地址。

`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。

`backlog=`*number*	在 `listen()` 调用中设置 *backlog* 参数, 该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*	为监听套接字设置接收缓冲区大小 (*SO_RCVBUF* 选项)。
`sndbuf=`*size*	为监听套接字设置发送缓冲区大小 (*SO_SNDBUF* 选项)。
`bind`	指示为给定的 *address:port* 对进行单独的 `bind()` 调用。事实上, 如果有多个具有相同端口但不同地址的 `listen` 指令, 并且其中一个 `listen` 指令监听给定端口的所有地址 (**:port*),Angie 将只 `bind()` 到 **:port*。应该注意的是, 在这种情况下将进行 `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, 那么对于给定的 *address:port* 对将始终进行单独的 `bind()` 调用。
`ipv6only=on` | `off`	确定 (通过 *IPV6_V6ONLY* 套接字选项) 监听通配符地址 *[::]* 的 IPv6 套接字是只接受 IPv6 连接还是同时接受 IPv6 和 IPv4 连接。此参数默认开启。它只能在启动时设置一次。
`multipath`	启用通过 Multipath TCP (MPTCP) 接受连接, 在 Linux 内核 5.6 及更高版本中支持。

`so_keepalive=on` | `off` | [`keepidle`]:[`keepintvl`]:[`keepcnt`]

为监听套接字配置"TCP keepalive" 行为。

`''`	如果省略此参数, 则套接字将使用操作系统的设置
`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 keepalive 参数。在这些系统上，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
```

将空闲超时 (TCP_KEEPIDLE) 设置为 30 分钟，将探测间隔 (TCP_KEEPINTVL) 保留为系统默认值，并将探测次数 (TCP_KEEPCNT) 设置为 10 次探测。

不同的服务器必须监听不同的 *address:port* 对。

mail

语法	`mail { ... }`
默认值	—
上下文	main

提供配置文件上下文，在其中指定邮件服务器指令。

max_commands

语法	`max_commands` *number*;
默认值	`max_commands 1000;`
上下文	mail, server

设置身份验证期间发出的最大命令数，以增强对 DoS 攻击的防护。

max_errors

语法	`max_errors` *number*;
默认值	`max_errors 5;`
上下文	mail, server

设置协议错误的次数，超过此次数后将关闭连接。

protocol

语法	`protocol` *imap* | *pop3* | *smtp*;
默认值	—
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*] | `off`;
默认值	`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，以将其传递给身份验证服务器，并在代理 SMTP 时用于*XCLIENT* 命令。例如:

```
resolver 127.0.0.53 [::1]:5353;
```

特殊值 `off` 会禁用客户端主机名解析, 并取消继承的指令值。

地址可以指定为域名或 IP 地址, 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。

备注

建议使用本地可信解析器，例如 `127.0.0.53` (systemd-resolved)，而非公共解析器（如 `8.8.8.8`)。公共解析器会将 DNS 查询暴露给第三方，并增加缓存投毒攻击的风险。

备注

指令值会被嵌套块继承, 并且可以在嵌套块中根据需要进行覆盖。在单个块内, 指令只能指定一次。如果重复指定, 则最后一次定义生效。

默认情况下,Angie 使用响应的 TTL 值来缓存答案。如果未指定 `resolver` 指令且不执行动态 DNS 查询 (例如, 在*Proxy* 中使用固定名称而不使用变量时), 则不需要指定解析器: 名称将在启动时使用系统解析器进行解析。可选的 `valid` 参数允许覆盖此行为:

`valid`	可选参数, 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 地址查找
`ipv6=off`	禁用 IPv6 地址查找
`status_zone`	可选参数, 在指定区域中启用 DNS 服务器请求和响应信息的收集

小技巧

为了防止 DNS 欺骗, 建议在安全可靠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, 例如 `127.0.0.11`。

resolver_timeout

语法	`resolver_timeout` *time;*
默认值	`resolver_timeout 30s;`
上下文	mail, server

设置名称解析的超时时间, 例如:

```
resolver_timeout 5s;
```

server

语法	`server { ... }`
默认值	—
上下文	mail

设置服务器的配置。

server_name

语法	`server_name` *name*;
默认值	`server_name hostname;`
上下文	mail, server

设置服务器名称, 用于:

- POP3/SMTP 服务器的初始问候语;
- SASL CRAM-MD5 认证期间的盐值;
- 连接到 SMTP 后端时的 EHLO 命令, 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;*
默认值	`timeout 60s;`
上下文	mail, server

设置开始代理到后端之前使用的超时时间。

3.2.1.5 Google PerfTools 模块

使用 Google Performance Tools 对 Angie 工作进程进行性能分析。该模块面向 Angie 开发人员，通过提供有关内存使用、CPU 负载和其他性能指标的详细信息，帮助他们分析和优化服务器性能。

在 从源代码构建时，默认不构建此模块；需要使用 `--with-google_perftools_module` 构建参数启用。

备注

此模块需要 gperftools 库。

配置示例

```
google_perftools_profiles /var/log/angie/perftools;
```

性能分析文件将存储在类似 `/var/log/angie/perftools.< 工作进程 PID>` 的文件中。

指令

google_perftools_profiles

语法	`google_perftools_profiles` 文件前缀;
默认值	—
上下文	main

设置文件名前缀，用于存储 Angie 工作进程的性能分析信息。工作进程 ID 会附加在文件名末尾的点号之后，例如：`/var/log/angie/perftools.1234`。

3.2.1.6 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_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_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_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。

3.2.1.7 核心模块

Core	管理服务文件、进程和其他 Angie 模块。

3.2.1.8 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。
ACME	使用 ACME 协议为 HTTP 服务器自动获取和续期 SSL 证书。
Docker	基于 Docker 容器标签动态更新代理服务器组。
Addition	在响应正文之前或之后插入指定的代码片段。
API	RESTful HTTP 接口，用于获取基本 Web 服务器信息和 JSON 格式的统计数据，以及管理代理服务器组。
Auth Basic	基于用户名和密码的基本 HTTP 身份验证用于访问控制。
Auth Request	使用对外部 HTTP 服务的子请求进行授权。
AutoIndex	在没有索引文件时自动生成目录列表。
Browser (已弃用)	基于 `User-Agent` 头的浏览器识别。
Charset	配置和转换响应编码。
DAV	使用 WebDAV 协议在服务器上进行文件管理。
DoH	DNS over HTTPS (RFC 8484) 服务器，通过 UDP 或 TCP 将 DNS 查询代理到一组 DNS 服务器。
Empty GIF	提供一个单像素透明 GIF。
FastCGI	将请求代理到 FastCGI 服务器。
FLV	Flash Video (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 GZip 压缩的响应以进行修改，以及在客户端不支持压缩的情况下使用。
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[1]	图像转换。
Index	配置索引文件，用于处理以斜杠 (`/`) 结尾的请求。
Limit Conn	限制并发请求数 (活动连接数) 以防止过载。
Limit Req	限制请求频率以防止过载和密码猜测。
Log	配置请求日志以跟踪资源访问用于监控和分析目的。
Map	基于预定义的键值对转换变量。
Metric	实时统计 API 中的自定义数值指标。
Memcached	从 Memcached 服务器检索响应。
Mirror	将请求镜像到其他服务器。

续下页

表 1 – 接上页

MP4	MP4 文件的伪流式传输。
Perl[1]	通过在 Perl 语言中指定附加逻辑来扩展功能的处理程序。
Prometheus	Prometheus 兼容格式的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	验证 `Referer` 头值。
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	创建具有限制访问时间能力的安全链接。
Slice	将请求拆分为多个子请求以获取单个片段，以便更好地缓存大型响应。
Split Clients	创建用于 A/B 测试、金丝雀发布、分片和其他需要按比例分组的场景的变量。
SSI	处理响应中的 SSI(服务器端包含) 命令。
SSL	用于处理 HTTPS 请求的 SSL/TLS 配置。
Stub Status (已弃用)	文本格式的全局连接和请求计数器。
Sub	在响应正文中搜索和替换片段。
Tunnel	处理 HTTP/1.1 CONNECT 请求并建立端到端隧道。
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探测。
UserID	发放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[1]	使用 XSLT 语言转换 XML 文档。

[1] 在我们的构建中, 这些模块是动态编译的, 并作为 独立软件包安装; 详细信息请参阅每个模块的说明。

3.2.1.9 Stream 模块

Stream	用于在 L4 层面平衡 TCP 和 UDP 协议的核心流服务器功能。
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续期 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。
Limit Conn	限制并发连接数以防止过载。
Log	配置会话日志以跟踪资源访问用于监控和分析目的。
Map	基于预定义的键值对转换变量。
Metric	实时统计 API 中的自定义数值指标。
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	配置到其他服务器的代理。
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而不进行进一步代理。
Set	设置指定的变量值。
Split Clients	创建用于 A/B 测试、金丝雀发布、分片和其他需要按比例分组的场景的变量。
SSL	SSL/TLS 和 DTLS 协议终止。
SSL Preread	在不进行 SSL/TLS 终止的情况下从 `ClientHello` 消息中提取信息，并在做出负载均衡决策之前使用。
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探测。

3.2.1.10 邮件模块

Mail	核心邮件代理服务器功能。
Auth HTTP	使用 HTTP 请求向外部服务器进行用户身份验证和服务器选择，以便后续代理。
IMAP	IMAP 协议支持。
POP3	POP3 协议支持。
Proxy	配置到其他服务器的代理。
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。

3.2.1.11 Google PerfTools 模块

Google PerfTools	负责与 Google Performance Tools 库集成，用于应用程序性能分析。

3.2.1.12 WASM 模块

WASM?pagename? 559, 1	核心 WASM 功能，在 Angie 中启用 WASM 代码执行。
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>*
$acme_hook_challenge	
$acme_hook_client	
$acme_hook_domain	
$acme_hook_keyauth	
$acme_hook_name	
$acme_hook_token	
$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	
$content_length	
$content_type	
$cookie_<name>	
$date_local	
$date_gmt	
$document_root	
$document_uri	
$fastcgi_script_name	
$fastcgi_path_info	
$gzip_ratio	

续下页

表 2 – 接上页

HTTP 模块	*Stream* 模块
$host	
$hostname	*$hostname*
$http2	
$http3	
$http_<name>	
$https	
$invalid_referer	
$is_args	
$is_request_port	
$limit_conn_status	*$limit_conn_status*
$limit_rate	
$limit_req_status	
$memcached_key	
$metric_<name>	*$metric_<name>*
$metric_<name>_key 和 *$metric_<name>_value*	*$metric_<name>_key* 和 *$metric_<name>_value*
$metric_<name>_key 和 *$metric_<name>_value*	*$metric_<name>_key* 和 *$metric_<name>_value*
$metric_<name>_value_<metric>	*$metric_<name>_value_<metric>*
$modern_browser	
	$mqtt_preread_clientid
	$mqtt_preread_username
$msec	*$msec*
$msie	
协议	
$nginx_version	*$nginx_version*
$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	
	$rdp_cookie, *$rdp_cookie_<name>*

续下页

表 2 – 接上页

HTTP 模块	Stream 模块
$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_port	
$request_time	
$request_uri	
$scheme	
$secure_link	
$secure_link_expires	
$sent_http_<name>	
$sent_body	
$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

续下页

表 2 – 接上页

HTTP 模块	Stream 模块
$ssl_client_sigalg	$ssl_client_sigalg
$ssl_client_v_end	$ssl_client_v_end
$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_encrypted_hello	$ssl_encrypted_hello
	$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
$ssl_sigalg	$ssl_sigalg
$ssl_sigalgs	
$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_current_addr (PRO)	$upstream_current_addr (PRO)
$upstream_bytes_received	$upstream_bytes_received
$upstream_bytes_sent	$upstream_bytes_sent
$upstream_cache_status	
$upstream_cache_key	
$upstream_connect_time	$upstream_connect_time
$upstream_cookie_<name>	
	$upstream_first_byte_time
$upstream_header_time	
$upstream_http_<name>	
$upstream_request_method	

续下页

表 2 – 接上页

HTTP 模块	*Stream 模块*
$upstream_probe (PRO)	*$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*
$upstream_transport	
$upstream_trailer_<name>	
$upstream_queue_time	
$uri	

3.2.3 NJS API 参考

NJS 模块提供了用于扩展 Angie 功能的对象、方法和属性。

本参考仅包含 NJS 特定的、不符合 ECMAScript 标准的属性、方法和模块。符合 ECMAScript 标准的 NJS 属性和方法的定义可以在 ECMAScript 规范 中找到。

3.2.3.1 Angie 对象

HTTP 请求

- `r.args{}`
- `r.done()`
- `r.error()`
- `r.finish()`
- `r.headersIn{}`
- `r.headersOut{}`
- `r.httpVersion`
- `r.internal`
- `r.internalRedirect()`
- `r.log()`
- `r.method`
- `r.parent`
- `r.remoteAddress`

- r.requestBody
- r.requestBuffer
- r.requestText
- r.rawHeadersIn[]
- r.rawHeadersOut[]
- r.responseBody
- r.responseBuffer
- r.responseText
- r.return()
- r.send()
- r.sendBuffer()
- r.sendHeader()
- r.setReturnValue()
- r.status
- r.subrequest()
- r.uri
- r.rawVariables{}
- r.variables{}
- r.warn()

HTTP 请求对象仅在 HTTP JS 模块中可用。在 0.8.5 之前，该对象的所有字符串属性都是字节字符串。

r.args{}

请求参数对象，只读。

查询字符串以对象形式返回。从 0.7.6 开始，重复的键以数组形式返回，键区分大小写，键和值都会进行百分号解码。

例如，查询字符串

```
a=1&b=%32&A=3&b=4&B=two%20words
```

会被转换为 r.args：

```
{a: "1", b: ["2", "4"], A: "3", B: "two words"}
```

更高级的解析场景可以使用 *Query String* 模块和 $args 变量来实现，例如：

```
import qs from 'querystring';

function args(r) {
    return qs.parse(r.variables.args);
}
```

参数对象在首次访问 `r.args` 时进行求值。如果只需要单个参数，例如 `foo`，可以使用 Angie 变量：

```
r.variables.arg_foo
```

这里，Angie 变量对象返回给定键的第一个值，不区分大小写，不进行百分号解码。

要将 `r.args` 转换回字符串，可以使用 Query String 的 `stringify` 方法。

`r.done()`
调用此函数后，后续的数据块将直接传递给客户端，而不调用 `js_body_filter` （0.5.2）。只能从 `js_body_filter` 函数中调用。

`r.error(string)`
将 `string` 以 `error` 日志级别写入错误日志。

> **备注**
>
> 由于 Angie 有硬编码的最大行长度限制，只有字符串的前 2048 字节可以被记录。

`r.finish()`
完成向客户端发送响应。

`r.headersIn{}`
传入的请求头对象，只读。

可以使用以下语法访问 `Foo` 请求头：`headersIn.foo` 或 `headersIn['Foo']`。

`Authorization`、`Content-Length`、`Content-Range`、`Content-Type`、`ETag`、`Expect`、`From`、`Host`、`If-Match`、`If-Modified-Since`、`If-None-Match`、`If-Range`、`If-Unmodified-Since`、`Max-Forwards`、`Proxy-Authorization`、`Referer`、`Transfer-Encoding` 和 `User-Agent` 请求头只能有一个字段值（0.4.1）。`Cookie` 头中的重复字段值用分号（`;`）分隔。所有其他请求头中的重复字段值用逗号分隔。

`r.headersOut{}`
主请求的传出响应头对象，可写。

如果 `r.headersOut{}` 是子请求的响应对象，它表示响应头。在这种情况下，`Accept-Ranges`、`Connection`、`Content-Disposition`、`Content-Encoding`、`Content-Length`、`Content-Range`、`Date`、`Keep-Alive`、`Server`、`Transfer-Encoding`、`X-Accel-*` 响应头中的字段值可能会被省略。

可以使用以下语法访问 `Foo` 响应头：`headersOut.foo` 或 `headersOut['Foo']`。

传出的响应头应该在响应头发送给客户端之前设置；否则，头的更新将被忽略。这意味着 `r.headersOut{}` 在以下情况下有效可写：

- 在 `js_content` 处理程序中，在调用 `r.sendHeader()` 或 `r.return()` 之前
- 在 `js_header_filter` 处理程序中

多值响应头（0.4.0）的字段值可以使用以下语法设置：

```
r.headersOut['Foo'] = ['a', 'b']
```

输出将是：

```
Foo: a
Foo: b
```

`Foo` 响应头的所有先前字段值将被删除。

对于只接受单个字段值的标准响应头（如 `Content-Type`），只有数组的最后一个元素会生效。`Set-Cookie` 响应头的字段值始终以数组形式返回。`Age`、`Content-Encoding`、`Content-Length`、`Content-Type`、`ETag`、`Expires`、`Last-Modified`、`Location`、`Retry-After` 响应头中的重复字段值会被忽略。所有其他响应头中的重复字段值用逗号分隔。

`r.httpVersion`
HTTP 版本，只读。

`r.internal`
布尔值，对于内部 location 为 `true`。

`r.internalRedirect(uri)`
执行到指定 `uri` 的内部重定向。如果 URI 以 `@` 前缀开头，则被视为命名 location。在新 location 中，所有请求处理从普通 location 的 `NGX_HTTP_SERVER_REWRITE_PHASE` 开始重复，从命名 location 的 `NGX_HTTP_REWRITE_PHASE` 开始重复。因此，重定向到命名 location 不会检查 `client_max_body_size` 限制。重定向的请求变为内部请求，可以访问内部 location。实际的重定向发生在处理程序执行完成后。

> **备注**
> 重定向后，在目标 location 中启动一个新的 NJS VM，原始 location 中的 VM 停止。Angie 变量的值会保留，可用于向目标 location 传递信息。从 0.5.3 开始，可以使用通过 `js_var` 指令为 HTTP 或 Stream 声明的变量。

> **备注**
> 从 0.7.4 开始，该方法接受转义的 URI。

`r.log(string)`
将 `string` 以 `info` 日志级别写入错误日志。

备注

由于 Angie 有硬编码的最大行长度限制，只有字符串的前 2048 字节可以被记录。

`r.method`
: HTTP 方法，只读。

`r.parent`
: 引用父请求对象。

`r.remoteAddress`
: 客户端地址，只读。

`r.requestBody`
: 该属性在 0.5.0 中已过时，在 0.8.0 中被移除。应该使用 `r.requestBuffer` 或 `r.requestText` 属性代替。

`r.requestBuffer`
: 客户端请求体（如果尚未写入临时文件）（从 0.5.0 开始）。要确保客户端请求体在内存中，其大小应受 `client_max_body_size` 限制，并且应使用 `client_body_buffer_size` 设置足够的缓冲区大小。该属性仅在 `js_content` 指令中可用。

`r.requestText`
: 与 `r.requestBuffer` 相同，但返回一个字符串。注意，它可能会将 UTF-8 编码中无效的字节转换为替换字符。

`r.rawHeadersIn[]`
: 返回键值对数组，完全按照从客户端接收的原样（0.4.1）。

  例如，对于以下请求头：

```
Host: localhost
Foo:  bar
foo:  bar2
```

  `r.rawHeadersIn` 的输出将是：

```
[
    ['Host', 'localhost'],
    ['Foo', 'bar'],
    ['foo', 'bar2']
]
```

  可以使用以下语法收集所有 `foo` 头：

```
r.rawHeadersIn.filter(v=>v[0].toLowerCase() == 'foo').map(v=>v[1])
```

  输出将是：

```
['bar', 'bar2']
```

头字段名称不会转换为小写，重复的字段值不会合并。

`r.rawHeadersOut[]`
返回响应头的键值对数组（0.4.1）。头字段名称不会转换为小写，重复的字段值不会合并。

`r.responseBody`
该属性在 0.5.0 中已弃用，在 0.8.0 中被移除。应该使用 `r.responseBuffer` 或 `r.responseText` 属性代替。

`r.responseBuffer`
包含子请求响应体，只读（从 0.5.0 开始）。`r.responseBuffer` 的大小受 `subrequest_output_buffer_size` 指令限制。

`r.responseText`
与 `r.responseBuffer` 相同，但返回字符串（从 0.5.0 开始）。注意，它可能会将 UTF-8 编码中无效的字节转换为替换字符。

`r.return(status[, string | Buffer])`
向客户端发送带有指定 `status` 的完整响应。响应可以是字符串或 Buffer（0.5.0）。

可以将重定向 URL（用于代码 301、302、303、307 和 308）或响应正文文本（用于其他代码）指定为第二个参数。

`r.send(string | Buffer)`
向客户端发送响应正文的一部分。发送的数据可以是字符串或 Buffer（0.5.0）。

`r.sendBuffer(data[, options])`
将数据添加到要转发到下一个正文过滤器的数据块链中（0.5.2）。实际转发会在稍后发生，即当前链的所有数据块都处理完毕时。

数据可以是字符串或 Buffer。`options` 是一个对象，用于覆盖从传入数据块缓冲区派生的 Angie 缓冲区标志。可以使用以下标志覆盖这些标志：

`last`
布尔值，`true` 表示该缓冲区是最后一个缓冲区。

`flush`
布尔值，`true` 表示该缓冲区应具有 `flush` 标志。

该方法只能从 `js_body_filter` 函数中调用。

`r.sendHeader()`
向客户端发送 HTTP 头。

`r.setReturnValue(value)`
设置 `js_set` 处理程序的返回值（0.7.0）。与普通的 return 语句不同，当处理程序是 JS 异步函数时应使用此方法。例如：

```
async function js_set(r) {
    const digest = await crypto.subtle.digest('SHA-256', r.headersIn.host);
    r.setReturnValue(digest);
}
```

`r.status`
: 状态，可写。

`r.subrequest(uri[, options[, callback]])`
: 使用给定的 `uri` 和 `options` 创建子请求，并安装可选的完成 `callback`。

  子请求与客户端请求共享其输入标头。要向代理服务器发送与原始标头不同的标头，可以使用 `proxy_set_header` 指令。要向代理服务器发送全新的标头集，可以使用 `proxy_pass_request_headers` 指令。

  如果 `options` 是字符串，则它保存子请求参数字符串。否则，:samp:*options* 应为具有以下键的对象：

  `args`
  : 参数字符串，默认使用空字符串。

  `body`
  : 请求主体，默认使用父请求对象的请求主体。

  `method`
  : HTTP 方法，默认使用 `GET` 方法。

  `detached`
  : 布尔标志 (0.3.9)；如果为 `true`，则创建的子请求是分离的子请求。对分离子请求的响应将被忽略。与普通子请求不同，分离的子请求可以在变量处理程序内创建。`detached` 标志和 callback 参数是互斥的。

  完成 `callback` 接收一个子请求响应对象，其方法和属性与父请求对象相同。

  从 0.3.8 开始，如果未提供 `callback`，则返回解析为子请求响应对象的 `Promise` 对象。

  例如，要查看子请求中的所有响应标头：

```
async function handler(r) {
    const reply = await r.subrequest('/path');

    for (const h in reply.headersOut) {
        r.log(`${h}: ${reply.headersOut[h]}`);
    }

    r.return(200);
}
```

`r.uri`
: 请求中的当前 URI，已规范化，只读。

`r.rawVariables{}`
: Angie 变量作为 Buffers, 可写 (自 0.5.0 起)。

`r.variables{}`
: Angie 变量对象, 可写 (自 0.2.8 起)。

例如, 要获取 `$foo` 变量, 可以使用以下语法之一:

```
r.variables['foo']
r.variables.foo
```

从 0.8.6 开始, 可以使用以下语法访问正则表达式捕获:

```
r.variables['1']
r.variables[1]
```

Angie 对在 `angie.conf` 中引用的变量和未引用的变量的处理方式不同。当变量被引用时, 它可能是可缓存的, 但当它未被引用时, 它始终是不可缓存的。例如, 当 `$request_id` 变量仅从 NJS 访问时, 每次求值时它都有一个新值。但是, 当 `$request_id` 被引用时, 例如:

```
proxy_set_header X-Request-Id $request_id;
```

`r.variables.request_id` 每次都返回相同的值。

变量可写的条件是:

- 它是使用 HTTP 或 Stream 的 `js_var` 指令创建的 (自 0.5.3 起)
- 它在 Angie 配置文件中被引用

即便如此, 某些内置变量仍然无法被赋值 (例如 `$http_`)。

`r.warn(string)`
: 在 `warning` 日志级别将 `string` 写入错误日志。

> **备注**
>
> 由于 Angie 具有硬编码的最大行长度限制, 因此只能记录字符串的前 2048 个字节。

Stream 会话

- `s.allow()`
- `s.decline()`
- `s.deny()`
- `s.done()`
- `s.error()`
- `s.log()`

- `s.off()`
- `s.on()`
- `s.remoteAddress`
- `s.rawVariables{}`
- `s.send()`
- `s.sendDownstream()`
- `s.sendUpstream()`
- `s.status`
- `s.setReturnValue()`
- `s.variables{}`
- `s.warn()`

流会话对象仅在 Stream JS 模块中可用。在 0.8.5 之前, 该对象的所有字符串属性都是字节字符串。

`s.allow()`
: `s.done(0)` 的别名 (0.2.4)。

`s.decline()`
: `s.done(-5)` 的别名 (0.2.4)。

`s.deny()`
: `s.done(403)` 的别名 (0.2.4)。

`s.done([code])`
: 将当前阶段处理程序的退出 `code` 设置为代码值, 默认为 `0`。实际的终结发生在 js 处理程序完成并且所有待处理事件 (例如来自 `ngx.fetch()` 或 `setTimeout()` 的事件) 都被处理之后 (0.2.4)。

  可能的代码值:

  - `0` —成功终结, 将控制权传递给下一个阶段
  - `-5` —未决定, 将控制权传递给当前阶段的下一个处理程序 (如果有)
  - `403` —访问被禁止

  只能从阶段处理程序函数调用:`js_access` 或 `js_preread`。

`s.error(string)`
: 在 `error` 日志级别将发送的 `string` 写入错误日志。

  备注

  由于 Angie 具有硬编码的最大行长度限制, 因此只能记录字符串的前 2048 个字节。

`s.log(string)`
在 `info` 日志级别将发送的 `string` 写入错误日志。

> **备注**
>
> 由于 Angie 具有硬编码的最大行长度限制, 因此只能记录字符串的前 2048 个字节。

`s.off(eventName)`
注销由 `s.on()` 方法设置的回调 (0.2.4)。

`s.on(event, callback)`
为指定的 `event` 注册 `callback` (0.2.4)。

`event` 可以是以下字符串之一:

`upload`
来自客户端的新数据 (字符串)。

`download`
发送到客户端的新数据 (字符串)。

`upstream`
来自客户端的新数据 (Buffer)(自 0.5.0 起)。

`downstream`
发送到客户端的新数据 (Buffer)(自 0.5.0 起)。

完成回调具有以下原型:`callback(data, flags)`, 其中 `data` 是字符串或 Buffer(取决于事件类型);:samp:*flags* 是具有以下属性的对象:

`last`
布尔值,:samp:*true* 表示数据是最后一个缓冲区。

`s.remoteAddress`
客户端地址, 只读。

`s.rawVariables`
Angie 变量作为 Buffers, 可写 (自 0.5.0 起)。

`s.send(data[, options])`
将数据添加到将在正向方向转发的数据块链中: 在下载回调中发送到客户端; 在上传中发送到上游服务器 (0.2.4)。实际转发会在稍后发生, 即当前链的所有数据块都处理完毕时。

数据可以是字符串或 Buffer(0.5.0)。`options` 是一个对象, 用于覆盖从传入数据块缓冲区派生的 Angie 缓冲区标志。可以使用以下标志覆盖这些标志:

`last`
布尔值,:samp:*true* 表示该缓冲区是最后一个缓冲区。

`flush`
布尔值,:samp:*true* 表示该缓冲区应具有 `flush` 标志。

该方法可以在每次回调调用中多次调用。

`s.sendDownstream()`
: 与 `s.send()` 相同, 但它始终将数据发送到客户端 (自 0.7.8 起)。

`s.sendUpstream()`
: 与 `s.send()` 相同, 但它始终从客户端发送数据 (自 0.7.8 起)。

`s.status`
: 会话状态代码,:samp:*$status* 变量的别名, 只读 (自 0.5.2 起)。

`s.setReturnValue(value)`
: 设置 `js_set` 处理程序的返回值 (0.7.0)。与普通的 return 语句不同, 当处理程序是 JS 异步函数时应使用此方法。例如:

```
async function js_set(r) {
    const digest = await crypto.subtle.digest('SHA-256', r.headersIn.host);
    r.setReturnValue(digest);
}
```

`s.variables{}`
: Angie 变量对象, 可写 (自 0.2.8 起)。变量只有在 Angie 配置文件中被引用时才可写。即便如此, 某些内置变量仍然无法被赋值。

`s.warn(string)`
: 在 `warning` 日志级别将发送的 `string` 写入错误日志。

> **备注**
>
> 由于 Angie 具有硬编码的最大行长度限制, 因此只能记录字符串的前 2048 个字节。

周期会话

- `PeriodicSession.rawVariables{}`
- `PeriodicSession.variables{}`

`Periodic Session` 对象作为 HTTP 和 Stream 的 `js_periodic` 处理程序的第一个参数提供 (自 0.8.1 起)。

`PeriodicSession.rawVariables{}`
: Angie 变量作为 Buffers, 可写。

`PeriodicSession.variables{}`
: Angie 变量对象, 可写。

Headers

- `Headers()`
- `Headers.append()`
- `Headers.delete()`
- `Headers.get()`
- `Headers.getAll()`
- `Headers.forEach()`
- `Headers.has()`
- `Headers.set()`

Fetch API 的 `Headers` 接口自 0.5.1 起可用。

可以使用 `Headers()` 构造函数创建新的 `Headers` 对象 (自 0.7.10 起):

`Headers([init])`

> **`init`**
> 包含用于预填充 `Headers` 对象的 HTTP 标头的对象, 可以是 `string`、名称-值对的 `array`, 或现有的 `Headers` 对象。

可以使用以下属性和方法创建新的 `Headers` 对象:

`append()`
向 `Headers` 对象中的现有标头追加新值, 如果标头不存在则添加该标头 (自 0.7.10 起)。

`delete()`
从 `Headers` 对象中删除标头 (自 0.7.10 起)。

`get()`
返回包含指定名称的所有标头值的字符串, 值之间用逗号和空格分隔。

`getAll(name)`
返回包含指定名称的所有标头值的数组。

`forEach()`
对 `Headers` 对象中的每个键/值对执行一次提供的函数 (自 0.7.10 起)。

`has()`
返回布尔值, 指示是否存在具有指定名称的标头。

`set()`
为 `Headers` 对象内的现有标头设置新值, 如果标头不存在则添加该标头 (自 0.7.10 起)。

Request

- Request()
- Request.arrayBuffer()
- Request.bodyUsed
- Request.cache
- Request.credentials
- Request.headers
- Request.json()
- Request.method
- Request.mode
- Request.text()
- Request.url

Fetch API 的 Request 接口自 0.7.10 起可用。

可以使用 Request() 构造函数创建新的 Request 对象:

Request[resource[, options]])
: 创建一个可以稍后传递给 ngx.fetch() 的 Request 对象来获取。resource 可以是 URL 或现有的 Request 对象。options 是可选参数, 应为具有以下键的对象:

  body
  : 请求主体, 默认为空。

  headers
  : 响应标头对象——包含用于预填充 Headers 对象的 HTTP 标头的对象, 可以是 string、名称-值对的 array, 或现有的 Headers 对象。

  method
  : HTTP 方法, 默认使用 GET 方法。

可以使用以下属性和方法创建新的 Request 对象:

arrayBuffer()
: 返回一个 Promise, 该 Promise 解析为 ArrayBuffer。

bodyUsed
: 布尔值, 如果主体已在请求中使用则为 true。

cache
: 包含请求的缓存模式。

credentials
: 包含请求的凭据, 默认为 same-origin。

`headers`
: 与 `Request` 关联的只读 `Headers` 对象。

`json()`
: 返回一个 `Promise`, 该 Promise 解析为将请求主体解析为 JSON 的结果。

`method`
: 包含请求方法。

`mode`
: 包含请求的模式。

`text()`
: 返回一个 `Promise`, 该 Promise 解析为请求主体的字符串表示形式。

`url`
: 包含请求的 URL。

Response

- `Response()`
- `Response.arrayBuffer()`
- `Response.bodyUsed`
- `Response.headers`
- `Response.json()`
- `Response.ok`
- `Response.redirected`
- `Response.status`
- `Response.statusText`
- `Response.text()`
- `Response.type`
- `Response.url`

`Response` 接口自 0.5.1 起可用。

可以使用 `Response()` 构造函数创建新的 `Response` 对象 (自 0.7.10 起):

`Response[body[, options]])`
: 创建 `Response` 对象。`body` 是可选参数, 可以是 `string` 或 `buffer`, 默认为 `null`。`options` 是可选参数, 应为具有以下键的对象:

  `headers`
  : 响应标头对象——包含用于预填充 `Headers` 对象的 HTTP 标头的对象, 可以是 `string`、名称-值对的 `array`, 或现有的 `Headers` 对象。

`status`
: 响应的状态码。

`statusText`
: 与状态码对应的状态消息。

可以使用以下属性和方法创建新的 `Response()` 对象:

`arrayBuffer()`
: 获取 `Response` 流并读取至完成。返回一个 `Promise`, 该 Promise 解析为 `ArrayBuffer`。

`bodyUsed`
: 布尔值, 如果主体已被读取则为 `true`。

`headers`
: 与 `Response` 关联的只读 `Headers` 对象。

`json()`
: 获取 `Response` 流并读取至完成。返回一个 `Promise`, 该 Promise 解析为将主体文本解析为 JSON 的结果。

`ok`
: 布尔值, 如果响应成功 (状态码在 200–299 之间) 则为 `true`。

`redirected`
: 布尔值, 如果响应是重定向的结果则为 `true`。

`status`
: 响应的状态码。

`statusText`
: 与状态码对应的状态消息。

`text()`
: 获取 `Response` 流并读取至完成。返回一个 `Promise`, 该 Promise 解析为字符串。

`type`
: 响应的类型。

`url`
: 响应的 URL。

ngx

- `ngx.build`
- `ngx.conf_file_path`
- `ngx.conf_prefix`
- `ngx.error_log_path`
- `ngx.fetch()`

- `ngx.log()`
- `ngx.prefix`
- `ngx.version`
- `ngx.version_number`
- `ngx.worker_id`

`ngx` 全局对象自 0.5.0 起可用。

`ngx.build`
包含可选 Angie 构建名称的字符串，对应于 configure 脚本的 `--build=`*name* 参数，默认为 `""` (0.8.0)。

`ngx.conf_file_path`
包含当前 Angie 配置文件的文件路径的字符串 (0.8.0)。

`ngx.conf_prefix`
包含 Angie 配置前缀的文件路径的字符串——Angie 当前查找配置的目录 (0.7.8)。

`ngx.error_log_path`
包含当前错误日志文件的文件路径的字符串 (0.8.0)。

`ngx.fetch(resource, [options])`
发起请求以获取 `resource` (0.5.1)，可以是 URL 或 `Request` 对象 (0.7.10)。返回一个 `Promise`，该 Promise 解析为 `Response` 对象。从 0.7.0 开始，支持 `https://` 协议；不处理重定向。

如果 `resource` 中的 URL 指定为域名，则使用解析器进行解析。如果指定了 `https://` 协议，则应配置 `js_fetch_trusted_certificate` 指令以对 `resource` 的 HTTPS 服务器进行身份验证。

`options` 参数应为包含以下键的对象：

`body`
请求主体，默认为空。

`buffer_size`
读取响应的缓冲区大小，默认为 `4096`。

`headers`
请求头对象。

`max_response_body_size`
响应主体的最大大小 (以字节为单位)，默认为 `32768`。

`method`
HTTP 方法，默认使用 `GET` 方法。

`verify`
启用或禁用 HTTPS 服务器证书验证，默认为 `true` (0.7.0)。

示例：

```
let reply = await ngx.fetch('http://example.com/');
let body = await reply.text();

r.return(200, body);
```

ngx.log(level, message)
使用指定的日志级别将消息写入错误日志。`level` 参数指定日志级别之一;:samp:*message* 参数可以是字符串或 Buffer。可以指定以下日志级别:`ngx.INFO`、`ngx.WARN` 和 `ngx.ERR`。

> **备注**
>
> 由于 Angie 具有硬编码的最大行长度限制, 因此只能记录字符串的前 2048 个字节。

ngx.prefix
包含 Angie 前缀文件路径的字符串—保存服务器文件的目录 (0.8.0)。

ngx.version
包含 Angie 版本的字符串, 例如:`1.25.0` (0.8.0)。

ngx.version_number
包含 Angie 版本的数字, 例如:`1025000` (0.8.0)。

ngx.worker_id
对应于 Angie 内部工作进程 ID 的数字, 该值介于 `0` 和 `worker_processes` 指令中指定的值之间 (0.8.0)。

ngx.shared

`ngx.shared` 全局对象从 0.8.0 开始可用。

SharedDict

- `ngx.shared.SharedDict.add()`
- `ngx.shared.SharedDict.capacity`
- `ngx.shared.SharedDict.clear()`
- `ngx.shared.SharedDict.delete()`
- `ngx.shared.SharedDict.freeSpace()`
- `ngx.shared.SharedDict.get()`
- `ngx.shared.SharedDict.has()`
- `ngx.shared.SharedDict.incr()`
- `ngx.shared.SharedDict.items()`
- `ngx.shared.SharedDict.keys()`

- ngx.shared.SharedDict.name
- ngx.shared.SharedDict.pop()
- ngx.shared.SharedDict.replace()
- ngx.shared.SharedDict.set()
- ngx.shared.SharedDict.size()
- ngx.shared.SharedDict.type

共享字典对象从 0.8.0 开始可用。共享字典的名称、类型和大小通过 HTTP 或 Stream 中的 js_shared_dict_zone 指令设置。

SharedDict() 对象具有以下属性和方法:

ngx.shared.SharedDict.add(key, value [,timeout])
仅当键尚不存在时，才为字典中的指定 key 设置 value。key 参数是表示要添加项的键的字符串;:samp:*value* 参数是要添加项的值。

可选的 timeout 参数以毫秒为单位指定，并覆盖 HTTP 或 Stream 中 js_shared_dict_zone 指令的 timeout 参数 (从 0.8.5 开始)。当某些键预期具有唯一的超时时间时，这可能很有用。

如果值已成功添加到 SharedDict 字典，则返回 true; 如果键已存在于字典中，则返回 false。如果 SharedDict 字典中没有足够的可用空间，则抛出 SharedMemoryError。如果 value 的类型与此字典预期的类型不同，则抛出 TypeError。

ngx.shared.SharedDict.capacity
返回 SharedDict 字典的容量，对应于 HTTP 或 Stream 中 js_shared_dict_zone 指令的 size 参数。

ngx.shared.SharedDict.clear()
从 SharedDict 字典中删除所有项。

ngx.shared.SharedDict.delete(key)
从 SharedDict 字典中删除与指定键关联的项; 如果字典中的项存在并已删除，则返回 true，否则返回 false。

ngx.shared.SharedDict.freeSpace()
返回可用页面大小 (以字节为单位)。如果大小为零，则 SharedDict 字典仍将接受新值 (如果已占用页面中有空间)。

ngx.shared.SharedDict.get(key)
通过其 key 检索项; 返回与 key 关联的值，如果没有则返回 undefined。

ngx.shared.SharedDict.has(key)
通过其 key 搜索项; 如果存在此类项，则返回 true，否则返回 false。

ngx.shared.SharedDict.incr(key,delta[[,init], timeout])
将与 key 关联的整数值增加 delta。key 参数是字符串;:samp:*delta* 参数是要增加或减少值的数字。如果键不存在，则该项将初始化为可选的 init 参数，默认为 0。

可选的 timeout 参数以毫秒为单位指定, 并覆盖 HTTP 或 Stream 中 js_shared_dict_zone 指令的 timeout 参数 (从 0.8.5 开始)。当某些键预期具有唯一的超时时间时, 这可能很有用。

返回新值。如果 SharedDict 字典中没有足够的可用空间, 则抛出 SharedMemoryError。如果此字典不期望数字, 则抛出 TypeError。

> **备注**
>
> 仅当使用 HTTP 或 Stream 中 js_shared_dict_zone 指令的 type=number 参数声明字典类型时, 才能使用此方法。

ngx.shared.SharedDict.items([maxCount])
返回 SharedDict 字典键值项的数组 (从 0.8.1 开始)。maxCount 参数设置要检索的最大项数, 默认为 1024。

ngx.shared.SharedDict.keys([maxCount])
返回 SharedDict 字典键的数组。maxCount 参数设置要检索的最大键数, 默认为 1024。

ngx.shared.SharedDict.name
返回 SharedDict 字典的名称, 对应于 HTTP 或 Stream 中 js_shared_dict_zone 指令的 zone= 参数。

ngx.shared.SharedDict.pop(key)
从 SharedDict 字典中删除与指定 key 关联的项; 返回与 key 关联的值, 如果没有则返回 undefined。

ngx.shared.SharedDict.replace(key, value)
仅当键已存在时, 才替换指定 key 的 value; 如果值已成功替换, 则返回 true, 如果键在 SharedDict 字典中不存在, 则返回 false。如果 SharedDict 字典中没有足够的可用空间, 则抛出 SharedMemoryError。如果 value 的类型与此字典预期的类型不同, 则抛出 TypeError。

ngx.shared.SharedDict.set(key, value [,timeout])
为指定的 key 设置 value; 返回此 SharedDict 字典 (用于方法链)。

可选的 timeout 参数以毫秒为单位指定, 并覆盖 HTTP 或 Stream 中 js_shared_dict_zone 指令的 timeout 参数 (从 0.8.5 开始)。当某些键预期具有唯一的超时时间时, 这可能很有用。

ngx.shared.SharedDict.size()
返回 SharedDict 字典的项数。

ngx.shared.SharedDict.type
返回 string 或 number, 对应于 HTTP 或 Stream 中 js_shared_dict_zone 指令的 type= 参数设置的 SharedDict 字典类型。

3.2.3.2 内置对象

console

- `console.error()`
- `console.info()`
- `console.log()`
- `console.time()`
- `console.timeEnd()`
- `console.warn()`

`console` 对象从 Angie 0.8.2 开始可用, 从 CLI 0.2.6 开始可用。

`console.error(msg[, msg2 ...])`
输出一条或多条错误消息。消息可以是字符串或对象。

`console.info(msg[, msg2 ...])`
输出一条或多条信息消息。消息可以是字符串或对象。

`console.log(msg[, msg2 ...])`
输出一条或多条日志消息。消息可以是字符串或对象。

`console.time(label)`
启动一个计时器, 可以跟踪操作所需的时间。`label` 参数允许命名不同的计时器。如果使用相同的名称调用 `console.timeEnd()`, 则将输出自计时器启动以来经过的时间 (以毫秒为单位)。

`console.timeEnd(label)`
停止先前由 `console.time()` 启动的计时器。`label` 参数允许命名不同的计时器。

`console.warn(msg[, msg2 ...])`
输出一条或多条警告消息。消息可以是字符串或对象。

crypto

- `crypto.getRandomValues()`
- `crypto.subtle.encrypt()`
- `crypto.subtle.decrypt()`
- `crypto.subtle.deriveBits()`
- `crypto.subtle.deriveKey()`
- `crypto.subtle.digest()`
- `crypto.subtle.exportKey()`
- `crypto.subtle.generateKey()`
- `crypto.subtle.importKey()`

- `crypto.subtle.sign()`
- `crypto.subtle.verify()`

`crypto` 对象是一个全局对象, 允许使用加密功能 (自 0.7.0 起)。

`crypto.getRandomValues(typedArray)`
获取加密强度的随机值。返回作为 `typedArray` 传递的同一数组, 但其内容被替换为新生成的随机数。可能的值:

`typedArray`
可以是 `Int8Array`、`Int16Array`、`Uint16Array`、`Int32Array` 或 `Uint32Array`。

`crypto.subtle.encrypt(algorithm, key, data)`
使用提供的 `algorithm` 和 `key` 加密 `data`。返回一个 `Promise`, 该 Promise 将使用包含密文的 `ArrayBuffer` 来兑现。可能的值:

`algorithm`
一个对象, 指定要使用的算法以及所需的任何额外参数:

- 对于 `RSA-OAEP`, 传递一个具有以下键的对象:

  `name`
  一个字符串, 应设置为 `RSA-OAEP`:

  ```
  crypto.subtle.encrypt({name: "RSA-OAEP"}, key, data)
  ```

- 对于 `AES-CTR`, 传递一个具有以下键的对象:

  `name`
  一个字符串, 应设置为 `AES-CTR`。

  `counter`
  一个 `ArrayBuffer`、`TypedArray` 或 `DataView` —计数器块的初始值, 必须为 16 字节长 (AES 块大小)。此块最右边的 length 位用于计数器, 其余部分用于随机数。例如, 如果 length 设置为 64, 则 counter 的前半部分是随机数, 后半部分用于计数器。

  `length`
  计数器块中用于实际计数器的位数。计数器必须足够大, 以免回绕。

- 对于 `AES-CBC`, 传递一个具有以下键的对象:

  `name`
  一个字符串, 应设置为 `AES-CBC`。

  `iv`
  初始化向量, 是一个 `ArrayBuffer`、`TypedArray` 或 `DataView`, 必须为 16 字节, 不可预测, 最好是加密随机的。但是, 它不需要保密, 例如, 它可以与密文一起未加密传输。

- 对于 `AES-GCM`, 传递一个具有以下键的对象:

  `name`
  一个字符串, 应设置为 `AES-GCM`。

iv
初始化向量，是一个 ArrayBuffer、TypedArray 或 DataView，必须为 16 字节，并且对于使用给定密钥执行的每个加密操作必须是唯一的。

additionalData
(可选) 是一个 ArrayBuffer、TypedArray 或 DataView，包含不会被加密但将与加密数据一起进行身份验证的附加数据。如果指定了 additionalData，则必须在相应的 decrypt() 调用中指定相同的数据：如果提供给 decrypt() 调用的数据与原始数据不匹配，解密将抛出异常。additionalData 的位长度必须小于 2^64 - 1。

tagLength
(可选，默认为 128) - 一个 number，确定在加密操作中生成的身份验证标签的位大小，并在相应的解密中用于身份验证。可能的值：32、64、96、104、112、120 或 128。AES-GCM 规范建议应为 96、104、112、120 或 128，尽管在某些应用程序中 32 或 64 位可能是可接受的。

key
一个 CryptoKey，包含用于加密的密钥。

data
一个 ArrayBuffer、TypedArray 或 DataView，包含要加密的数据 (也称为明文)。

crypto.subtle.decrypt(algorithm, key, data)
解密加密的数据。返回一个包含解密数据的 Promise。可能的值：

algorithm
一个对象，指定要使用的算法以及所需的任何额外参数。为额外参数提供的值必须与传递到相应 encrypt() 调用中的值匹配。

- 对于 RSA-OAEP，传递一个具有以下键的对象：

  name
  一个字符串，应设置为 RSA-OAEP：

  ```
  crypto.subtle.encrypt({name: "RSA-OAEP"}, key, data)
  ```

- 对于 AES-CTR，传递一个具有以下键的对象：

  name
  一个字符串，应设置为 AES-CTR。

  counter
  一个 ArrayBuffer、TypedArray 或 DataView —计数器块的初始值，必须为 16 字节长 (AES 块大小)。此块最右边的 length 位用于计数器，其余部分用于随机数。例如，如果 length 设置为 64，则 counter 的前半部分是随机数，后半部分用于计数器。

  length
  计数器块中用于实际计数器的位数。计数器必须足够大，以免回绕。

- 对于 AES-CBC，传递一个具有以下键的对象：

name
一个字符串, 应设置为 AES-CBC。

iv
初始化向量, 是一个 ArrayBuffer、TypedArray 或 DataView, 必须为 16 字节, 不可预测, 最好是加密随机的。但是, 它不需要保密 (例如, 它可以与密文一起未加密传输)。

- 对于 AES-GCM, 传递一个具有以下键的对象:

name
一个字符串, 应设置为 AES-GCM。

iv
初始化向量, 是一个 ArrayBuffer、TypedArray 或 DataView, 必须为 16 字节, 并且对于使用给定密钥执行的每个加密操作必须是唯一的。

additionalData
(可选) 是一个 ArrayBuffer、TypedArray 或 DataView, 包含不会被加密但将与加密数据一起进行身份验证的附加数据。如果指定了 additionalData, 则必须在相应的 decrypt() 调用中指定相同的数据: 如果提供给 decrypt() 调用的数据与原始数据不匹配, 解密将抛出异常。additionalData 的位长度必须小于 2^64 - 1。

tagLength
(可选, 默认为 128) - 一个 number, 确定在加密操作中生成的身份验证标签的位大小, 并在相应的解密中用于身份验证。可能的值: 32、64、96、104、112、120 或 128。AES-GCM 规范建议应为 96、104、112、120 或 128, 尽管在某些应用程序中 32 或 64 位可能是可接受的。

key
一个 CryptoKey, 包含用于解密的密钥。如果使用 RSA-OAEP, 这是 CryptoKeyPair 对象的 privateKey 属性。

data
一个 ArrayBuffer、TypedArray 或 DataView, 包含要解密的数据 (也称为密文)。

crypto.subtle.deriveBits(algorithm, baseKey, length)
从基础密钥派生位数组。返回一个 Promise, 该 Promise 将使用包含派生位的 ArrayBuffer 来兑现。可能的值:

algorithm
一个对象, 定义要使用的派生算法:

- 对于 HKDF, 传递一个具有以下键的对象:

name
一个字符串, 应设置为 HKDF。

hash
一个字符串, 包含要使用的摘要算法: SHA-1、SHA-256、SHA-384 或 SHA-512。

salt
一个 ArrayBuffer、TypedArray 或 DataView, 表示与 digest 函数的输出长度相同

的随机或伪随机值。与传递到 deriveKey() 中的输入密钥材料不同,salt 不需要保密。

info
一个 ArrayBuffer、TypedArray 或 DataView, 表示特定于应用程序的上下文信息, 用于将派生密钥绑定到应用程序或上下文, 并允许在使用相同输入密钥材料的同时为不同上下文派生不同的密钥。此属性是必需的, 但可以是空缓冲区。

- 对于 PBKDF2, 传递一个具有以下键的对象:

name
一个字符串, 应设置为 PBKDF2。

hash
一个字符串, 包含要使用的摘要算法: SHA-1、SHA-256、SHA-384 或 SHA-512。

salt
一个 ArrayBuffer、TypedArray 或 DataView, 表示至少 16 字节的随机或伪随机值。与传递到 deriveKey() 中的输入密钥材料不同,salt 不需要保密。

iterations
一个 number, 表示在 deriveKey() 中哈希函数将被执行的次数。

- 对于 ECDH, 传递具有以下键的对象 (自 0.9.1 起):

name
一个字符串, 应设置为 ECDH。

public
一个 CryptoKey, 表示另一方的公钥。该密钥必须使用与基础密钥相同的曲线生成。

baseKey
一个 CryptoKey, 表示派生算法的输入 - 派生函数的初始密钥材料: 例如, 对于 PBKDF2, 它可能是一个密码, 使用 crypto.subtle.importKey() 作为 CryptoKey 导入。

length
一个数字, 表示要派生的比特数。为了浏览器兼容性, 该数字应该是 8 的倍数。

crypto.subtle.deriveKey(algorithm, baseKey, derivedKeyAlgorithm, extractable, keyUsages)
从主密钥派生一个秘密密钥。可能的值:

algorithm
一个对象, 定义要使用的派生算法:

- 对于 HKDF, 传递具有以下键的对象:

name
一个字符串, 应设置为 HKDF。

hash
一个字符串, 包含要使用的摘要算法: SHA-1、SHA-256、SHA-384 或 SHA-512。

salt
一个 ArrayBuffer、TypedArray 或 DataView, 表示与 digest 函数输出长度相同的随机或伪随机值。与传递给 deriveKey() 的输入密钥材料不同, 盐不需要保密。

info
一个 ArrayBuffer、TypedArray 或 DataView, 表示特定于应用程序的上下文信息, 用于将派生密钥绑定到应用程序或上下文, 并允许在使用相同输入密钥材料的同时为不同上下文派生不同的密钥。此属性是必需的, 但可以是空缓冲区。

- 对于 PBKDF2, 传递具有以下键的对象:

name
一个字符串, 应设置为 PBKDF2。

hash
一个字符串, 包含要使用的摘要算法: SHA-1、SHA-256、SHA-384 或 SHA-512。

salt
一个 ArrayBuffer、TypedArray 或 DataView, 表示至少 16 字节的随机或伪随机值。与传递给 deriveKey() 的输入密钥材料不同, 盐不需要保密。

iterations
一个 number, 表示在 deriveKey() 中哈希函数将被执行的次数。

- 对于 ECDH, 传递具有以下键的对象 (自 0.9.1 起):

name
一个字符串, 应设置为 ECDH。

publicKey
一个 CryptoKey, 表示另一方的公钥。该密钥必须使用与基础密钥相同的曲线生成。

baseKey
一个 CryptoKey, 表示派生算法的输入 - 派生函数的初始密钥材料: 例如, 对于 PBKDF2, 它可能是一个密码, 使用 crypto.subtle.importKey() 作为 CryptoKey 导入。

derivedKeyAlgorithm
一个对象, 定义派生密钥将用于的算法:

- 对于 HMAC, 传递具有以下键的对象:

name
一个字符串, 应设置为 HMAC。

hash
一个字符串, 包含要使用的摘要函数的名称: SHA-1、SHA-256、SHA-384 或 SHA-512。

length
(可选) 是一个 number, 表示密钥的比特长度。如果未指定, 密钥的长度等于所选哈希函数的块大小。

- 对于 AES-CTR、AES-CBC 或 AES-GCM, 传递具有以下键的对象:

name
一个字符串, 应根据所使用的算法设置为 AES-CTR、AES-CBC 或 AES-GCM。

length
一个 number, 表示要生成的密钥的比特长度: 128、192 或 256。

extractable
一个布尔值, 指示是否可以导出密钥。

keyUsages
一个 Array, 指示可以对派生密钥执行的操作。密钥用途必须被 derivedKeyAlgorithm 中设置的算法所允许。可能的值:

encrypt
用于加密消息的密钥。

decrypt
用于解密消息的密钥。

sign
用于签名消息的密钥。

verify
用于验证签名的密钥。

deriveKey
用于派生新密钥的密钥。

deriveBits
用于派生比特的密钥。

wrapKey
用于包装密钥的密钥。

unwrapKey
用于解包密钥的密钥。

crypto.subtle.digest(algorithm, data)
生成给定数据的摘要。接受要使用的摘要算法的标识符和要摘要的数据作为参数。返回一个 Promise, 该 Promise 将使用摘要来完成。可能的值:

algorithm
一个字符串, 定义要使用的哈希函数: SHA-1 (不用于加密应用程序)、SHA-256、SHA-384 或 SHA-512。

data
一个 ArrayBuffer、TypedArray 或 DataView, 包含要摘要的数据。

crypto.subtle.exportKey(format, key)
导出密钥: 将密钥作为 CryptoKey 对象, 并以外部可移植格式返回密钥 (自 0.7.10 起)。如果 format 是 jwk, 则 Promise 使用包含密钥的 JSON 对象来完成。否则,promise 使用包含密钥的 ArrayBuffer 来完成。可能的值:

`format`
: 一个字符串, 描述应导出密钥的数据格式, 可以是以下值:

  `raw`
  : 原始数据格式。

- 对于 `HMAC`, 传递具有以下键的对象:

  `name`
  : 字符串, 应设置为 `HMAC`。

  `hash`
  : 字符串，包含要使用的摘要函数的名称：可以是 `SHA-1`、`SHA-256`、`SHA-384` 或 `SHA-512`。

  `length`
  : (可选) 是一个 `number`, 表示密钥的比特长度。如果未指定, 密钥的长度等于所选哈希函数的块大小。

  – 对于 `AES-CTR`、`AES-CBC` 或 `AES-GCM`, 传递具有以下键的对象:

  `name`
  : 字符串, 应根据所使用的算法设置为 `AES-CTR`、`AES-CBC` 或 `AES-GCM`。

  `length`
  : 一个 `number`, 表示要生成的密钥的比特长度: 可以是 `128`、`192` 或 `256`。

`extractable`
: 布尔值, 指示是否可以导出密钥。

`keyUsages`
: 一个 `Array`, 指示派生密钥可以执行的操作。密钥用途必须被 `derivedKeyAlgorithm` 中设置的算法所允许。可能的值:

  `encrypt`
  : 用于加密消息的密钥。

  `decrypt`
  : 用于解密消息的密钥。

  `sign`
  : 用于签名消息的密钥。

  `verify`
  : 用于验证签名的密钥。

  `deriveKey`
  : 用于派生新密钥的密钥。

  `deriveBits`
  : 用于派生比特的密钥。

`wrapKey`
用于包装密钥的密钥。

`unwrapKey`
用于解包密钥的密钥。

`crypto.subtle.digest(algorithm, data)`
生成给定数据的摘要。接受要使用的摘要算法的标识符和要摘要的数据作为参数。返回一个 `Promise`, 该 Promise 以摘要来兑现。可能的值:

`algorithm`
定义要使用的哈希函数的字符串: 可以是 `SHA-1` (不用于加密应用)、`SHA-256`、`SHA-384` 或 `SHA-512`。

`data`
包含要摘要的数据的 `ArrayBuffer`、`TypedArray` 或 `DataView`。

`crypto.subtle.exportKey(format, key)`
导出密钥: 接受一个密钥作为 `CryptoKey` 对象, 并以外部可移植格式返回密钥 (自 0.7.10 起)。如果 `format` 是 `jwk`, 则 `Promise` 以包含密钥的 JSON 对象来兑现。否则,Promise 以包含密钥的 `ArrayBuffer` 来兑现。可能的值:

`format`
描述密钥应以何种数据格式导出的字符串, 可以是以下值:

`raw`
原始数据格式。

`pkcs8`
PKCS #8 格式。

`spki`
SubjectPublicKeyInfo 格式。

`jwk`
JSON Web Key (JWK) 格式 (自 0.7.10 起)。

`key`
包含要导出的密钥的 `CryptoKey`。

`crypto.subtle.generateKey(algorithm, extractable, usage)`
为对称算法生成新密钥或为公钥算法生成密钥对 (自 0.7.10 起)。返回一个 `Promise`, 该 Promise 以生成的密钥作为 `CryptoKey` 或 `CryptoKeyPair` 对象来兑现。可能的值:

`algorithm`
一个字典对象, 定义要生成的密钥类型并提供额外的特定于算法的参数:

- 对于 `RSASSA-PKCS1-v1_5`、`RSA-PSS` 或 `RSA-OAEP`, 传递具有以下键的对象:

  `name`
  字符串, 应根据所使用的算法设置为 `RSASSA-PKCS1-v1_5`、`RSA-PSS` 或 `RSA-OAEP`。

hash
字符串, 表示要使用的 digest 函数的名称, 可以是 SHA-256、SHA-384 或 SHA-512。

- 对于 ECDSA, 传递具有以下键的对象:

name
字符串, 应设置为 ECDSA。

namedCurve
字符串, 表示要使用的椭圆曲线的名称, 可以是 P-256、P-384 或 P-521。

- 对于 HMAC, 传递具有以下键的对象:

name
字符串, 应设置为 HMAC。

hash
字符串, 表示要使用的 digest 函数的名称, 可以是 SHA-256、SHA-384 或 SHA-512。

length
(可选) 是一个数字, 表示密钥的比特长度。如果省略, 密钥的长度等于所选摘要函数生成的摘要的长度。

- 对于 AES-CTR、AES-CBC 或 AES-GCM, 传递标识算法的字符串或形式为 *"name": "ALGORITHM"* 的对象, 其中 ALGORITHM 是算法的名称。

- 对于 ECDH, 传递具有以下键的对象 (自 0.9.1 起):

name
字符串, 应设置为 ECDH。

namedCurve
字符串, 表示要使用的椭圆曲线的名称, 可以是 P-256、P-384 或 P-521。

extractable
布尔值, 指示是否可以导出密钥。

usage
一个 array, 指示密钥的可能操作:

encrypt
用于加密消息的密钥。

decrypt
用于解密消息的密钥。

sign
用于签名消息的密钥。

verify
用于验证签名的密钥。

deriveKey
用于派生新密钥的密钥。

deriveBits
用于派生比特的密钥。

wrapKey
用于包装密钥的密钥。

unwrapKey
用于解包密钥的密钥。

crypto.subtle.importKey(format, keyData, algorithm, extractable, keyUsages)
导入密钥: 接受外部可移植格式的密钥作为输入, 并返回一个 CryptoKey 对象。返回一个 Promise, 该 Promise 以导入的密钥作为 CryptoKey 对象来兑现。可能的值:

format
描述要导入的密钥的数据格式的字符串, 可以是以下值:

raw
原始数据格式。

pkcs8
PKCS #8 格式。

spki
SubjectPublicKeyInfo 格式。

jwk
JSON Web Key (JWK) 格式 (自 0.7.10 起)。

keyData
包含给定格式密钥的 ArrayBuffer、TypedArray 或 DataView 对象。

algorithm
定义要导入的密钥类型并提供额外的算法特定参数的字典对象:

- 对于 RSASSA-PKCS1-v1_5、RSA-PSS 或 RSA-OAEP, 传递具有以下键的对象:

  name
  字符串, 应根据所使用的算法设置为 RSASSA-PKCS1-v1_5、RSA-PSS 或 RSA-OAEP。

  hash
  字符串, 表示要使用的 digest 函数的名称, 可以是 SHA-1、SHA-256、SHA-384 或 SHA-512。

- 对于 ECDSA, 传递具有以下键的对象:

  name
  字符串, 应设置为 ECDSA。

  namedCurve
  字符串, 表示要使用的椭圆曲线的名称, 可以是 P-256、P-384 或 P-521。

- 对于 HMAC, 传递具有以下键的对象:

name
字符串, 应设置为 HMAC。

hash
字符串, 表示要使用的 digest 函数的名称, 可以是 SHA-256、SHA-384 或 SHA-512。

length
(可选) 是一个数字, 表示密钥的比特长度。如果省略, 密钥的长度等于所选摘要函数生成的摘要的长度。

- 对于 AES-CTR、AES-CBC 或 AES-GCM, 传递标识算法的字符串或形式为 *"name": "ALGORITHM"* 的对象, 其中 ALGORITHM 是算法的名称。
- 对于 PBKDF2, 传递 PBKDF2 字符串。
- 对于 HKDF, 传递 HKDF 字符串。
- 对于 ECDH, 传递具有以下键的对象 (自 0.9.1 起):

name
字符串, 应设置为 ECDH。

namedCurve
字符串, 表示要使用的椭圆曲线的名称, 可以是 P-256、P-384 或 P-521。

extractable
布尔值, 指示是否可以导出密钥。

keyUsages
一个 array, 指示密钥的可能操作:

encrypt
用于加密消息的密钥。

decrypt
用于解密消息的密钥。

sign
用于签名消息的密钥。

verify
用于验证签名的密钥。

deriveKey
用于派生新密钥的密钥。

deriveBits
用于派生比特的密钥。

wrapKey
用于包装密钥的密钥。

unwrapKey
用于解包密钥的密钥。

`crypto.subtle.sign(algorithm, key, data)`

返回 `signature` 作为 `Promise`, 该 Promise 以包含签名的 `ArrayBuffer` 来兑现。可能的值:

algorithm

指定要使用的签名算法及其参数的字符串或对象:

- 对于 `RSASSA-PKCS1-v1_5`, 传递标识算法的字符串或形式为 *"name": "ALGORITHM"* 的对象。
- 对于 `RSA-PSS`, 传递具有以下键的对象:

  name

  字符串, 应设置为 `RSA-PSS`。

  saltLength

  长整型 `integer`, 表示要使用的随机盐的长度 (以字节为单位)。
- 对于 `ECDSA`, 传递具有以下键的对象:

  name

  字符串, 应设置为 `ECDSA`。

  hash

  要使用的摘要算法的标识符, 可以是 `SHA-256`、`SHA-384` 或 `SHA-512`。
- 对于 `HMAC`, 传递标识算法的字符串或形式为 *"name": "ALGORITHM"* 的对象。

key

包含用于签名的密钥的 `CryptoKey` 对象。如果算法标识公钥密码系统, 则这是私钥。

data

包含要签名的数据的 `ArrayBuffer`、`TypedArray` 或 `DataView` 对象。

`crypto.subtle.verify(algorithm, key, signature, data)`

验证数字签名; 返回一个 `Promise`, 该 Promise 以布尔值兑现: 如果签名有效则为 `true`, 否则为 `false`。可能的值:

algorithm

指定要使用的算法及其参数的字符串或对象:

- 对于 `RSASSA-PKCS1-v1_5`, 传递标识算法的字符串或形式为 *"name": "ALGORITHM"* 的对象。
- 对于 `RSA-PSS`, 传递具有以下键的对象:

  name

  字符串, 应设置为 `RSA-PSS`。

  saltLength

  长整型 `integer`, 表示要使用的随机盐的长度 (以字节为单位)。
- 对于 `ECDSA`, 传递具有以下键的对象:

name
字符串, 应设置为 ECDSA。

hash
要使用的摘要算法的标识符, 可以是 SHA-256、SHA-384 或 SHA-512。

- 对于 HMAC, 传递标识算法的字符串或形式为 *"name": "ALGORITHM"* 的对象。

key
包含用于验证的密钥的 CryptoKey 对象。对于对称算法, 它是密钥; 对于公钥系统, 它是公钥。

signature
包含要验证的签名的 ArrayBuffer、TypedArray 或 DataView。

data
包含要验证其签名的数据的 ArrayBuffer、TypedArray 或 DataView 对象。

CryptoKey

- CryptoKey.algorithm
- CryptoKey.extractable
- CryptoKey.type
- CryptoKey.usages

CryptoKey 对象表示从 SubtleCrypto 方法之一获得的加密 key: crypto.subtle.generateKey()、crypto.subtle.deriveKey()、crypto.subtle.importKey()。

CryptoKey.algorithm
返回一个对象, 描述此密钥可用于的算法以及任何相关的额外参数 (自 0.8.0 起), 只读。

CryptoKey.extractable
布尔值, 如果密钥可以导出则为 true (自 0.8.0 起), 只读。

CryptoKey.type
字符串值, 指示对象表示的密钥类型, 只读。可能的值:

secret
此密钥是用于对称算法的秘密密钥。

private
此密钥是非对称算法 CryptoKeyPair 的私钥部分。

public
此密钥是非对称算法 CryptoKeyPair 的公钥部分。

CryptoKey.usages
字符串数组, 指示此密钥可用于什么用途 (自 0.8.0 起), 只读。可能的数组值:

encrypt
用于加密消息的密钥。

decrypt
: 用于解密消息的密钥。

sign
: 用于签名消息的密钥。

verify
: 用于验证签名的密钥。

deriveKey
: 用于派生新密钥的密钥。

deriveBits
: 用于派生比特的密钥。

CryptoKeyPair

- `CryptoKeyPair.privateKey`
- `CryptoKeyPair.publicKey`

`CryptoKeyPair` 是 WebCrypto API 的字典对象, 表示非对称密钥对。

CryptoKeyPair.privateKey
: 表示私钥的 `CryptoKey` 对象。

CryptoKeyPair.publicKey
: 表示公钥的 `CryptoKey` 对象。

njs

- `njs.version`
- `njs.version_number`
- `njs.dump()`
- `njs.memoryStats`
- `njs.on()`

`njs` 对象是表示当前 VM 实例的全局对象 (自 0.2.0 起)。

njs.version
: 返回包含当前 NJS 版本的字符串 (例如,"0.7.4")。

njs.version_number
: 返回包含当前 NJS 版本的数字。例如,"0.7.4" 返回为 `0x000704` (自 0.7.4 起)。

njs.dump(value)
: 返回值的美化打印字符串表示。

njs.memoryStats
: 包含当前 VM 实例内存统计信息的对象 (自 0.7.8 起)。

`size`
: NJS 内存池从操作系统申请的内存量 (以字节为单位)。

`njs.on(event, callback)`
: 为指定的 VM 事件注册回调 (自 0.5.2 起)。事件可以是以下字符串之一:

  `exit`
  : 在 VM 销毁之前调用。回调不带参数调用。

process

- `process.argv`
- `process.env`
- `process.kill()`
- `process.pid`
- `process.ppid`

`process` 对象是提供有关当前进程信息的全局对象 (0.3.3)。

`process.argv`
: 返回包含启动当前进程时传递的命令行参数的数组。

`process.env`
: 返回包含用户环境的对象。

> **备注**
>
> 默认情况下,Angie 会删除从其父进程继承的所有环境变量, 但 TZ 变量除外。使用 `env` 指令来保留一些继承的变量。

`process.kill(pid, number | string)`
: 向由 `pid` 标识的进程发送信号。信号名称是数字或字符串, 例如 `SIGINT` 或 `SIGHUP`。有关更多信息, 请参阅 kill(2)。

`process.pid`
: 返回当前进程的 PID。

`process.ppid`
: 返回当前父进程的 PID。

String

默认情况下,NJS 中的所有字符串都是 Unicode 字符串。它们对应于包含 Unicode 字符的 ECMAScript 字符串。在 0.8.0 之前, 还支持字节字符串。

字节字符串（已移除）

备注

自 0.8.0 起，已移除对字节字符串和字节字符串方法的支持。在处理字节序列时，应使用*Buffer* 对象和 Buffer 属性，例如 `r.requestBuffer`、`r.rawVariables`。

字节字符串包含字节序列，用于将 Unicode 字符串序列化为外部数据以及从外部源反序列化。例如,:samp:*toUTF8()* 方法使用 UTF-8 编码将 Unicode 字符串序列化为字节字符串。`toBytes()` 方法将代码点最多为 255 的 Unicode 字符串序列化为字节字符串；否则返回 `null`。

以下方法已过时并在 0.8.0 中移除:

- `String.bytesFrom()` (在 0.8.0 中移除，使用 `Buffer.from()`)
- `String.prototype.fromBytes()` (在 0.8.0 中移除)
- `String.prototype.fromUTF8()` (在 0.8.0 中移除，使用 `TextDecoder`)
- `String.prototype.toBytes()` (在 0.8.0 中移除)
- `String.prototype.toString()` 带编码 (在 0.8.0 中移除)
- `String.prototype.toUTF8()` (在 0.8.0 中移除，使用 `TextEncoder`)

3.2.3.3 Web API

TextDecoder

- `TextDecoder()`
- `TextDecoder.prototype.encoding`
- `TextDecoder.prototype.fatal`
- `TextDecoder.prototype.ignoreBOM`
- `TextDecoder.prototype.decode()`

`TextDecoder` 从字节流生成代码点流 (0.4.3)。

`TextDecoder([encoding], options])`
为指定的 `encoding` 创建新的 `TextDecoder` 对象；目前仅支持 UTF-8。`options` 是具有以下属性的 `TextDecoderOptions` 字典:

`fatal`
布尔标志，指示当发现编码错误时 `TextDecoder.decode()` 是否必须抛出 `TypeError` 异常，默认为 `false`。

`TextDecoder.prototype.encoding`
返回包含 `TextDecoder()` 使用的编码名称的字符串，只读。

`TextDecoder.prototype.fatal`
: 布尔标志, 如果错误模式是致命的则为 `true`, 只读。

`TextDecoder.prototype.ignoreBOM`
: 布尔标志, 如果忽略字节顺序标记则为 `true`, 只读。

`TextDecoder.prototype.decode(buffer, [options])`
: 返回包含由 `TextDecoder()` 从 `buffer` 解码的文本的字符串。缓冲区可以是 `ArrayBuffer`。`options` 是具有以下属性的 `TextDecodeOptions` 字典:

  stream
  : 布尔标志, 指示后续调用 `decode()` 时是否会有额外数据: 如果分块处理数据则为 `true`, 如果是最后一块或数据未分块则为 `false`。默认为 `false`。

  示例:

```
>> (new TextDecoder()).decode(new Uint8Array([206,177,206,178]))
```

TextEncoder

- `TextEncoder()`
- `TextEncoder.prototype.encode()`
- `TextEncoder.prototype.encodeInto()`

`TextEncoder` 对象从代码点流生成 UTF-8 编码的字节流 (0.4.3)。

`TextEncoder()`
: 返回新构造的 `TextEncoder`, 它将生成 UTF-8 编码的字节流。

`TextEncoder.prototype.encode(string)`
: 将 `string` 编码为包含 UTF-8 编码文本的 `Uint8Array`。

`TextEncoder.prototype.encodeInto(string, uint8Array)`
: 将 `string` 编码为 UTF-8, 将结果放入目标 `Uint8Array`, 并返回显示编码进度的字典对象。字典对象包含两个成员:

  read
  : 从源 `string` 转换为 UTF-8 的 UTF-16 代码单元数。

  written
  : 目标 `Uint8Array` 中修改的字节数。

3.2.3.4 定时器

- `clearTimeout()`
- `setTimeout()`

`clearTimeout(timeout)`
: 取消由 `setTimeout()` 创建的 `timeout` 对象。

`setTimeout(function, milliseconds[, argument1, argumentN])`

在指定的 `milliseconds` 毫秒数后调用 `function`。可以向指定的函数传递一个或多个可选 `arguments`。返回 `timeout` 对象。

示例:

```
function handler(v)
{
    // ...
}

t = setTimeout(handler, 12);

// ...

clearTimeout(t);
```

全局函数

- `atob()`
- `btoa()`

`atob(encodedData)`

解码使用 Base64 编码的数据字符串。`encodedData` 参数是包含 Base64 编码数据的二进制字符串。返回包含从 `encodedData` 解码的数据的字符串。

类似的 `btoa()` 方法可用于编码和传输可能导致通信问题的数据, 然后传输它并使用 `atob()` 方法再次解码数据。例如, 您可以编码、传输和解码控制字符, 如 ASCII 值 0 到 31。

示例:

```
const encodedData = btoa("text to encode"); // 编码字符串
const decodedData = atob(encodedData); // 解码字符串
```

`btoa(stringToEncode)`

从二进制字符串创建 Base64 编码的 ASCII 字符串。`stringToEncode` 参数是要编码的二进制字符串。返回包含 `stringToEncode` 的 Base64 表示的 ASCII 字符串。

该方法可用于编码可能导致通信问题的数据, 传输它, 然后使用 `atob()` 方法再次解码数据。例如, 您可以编码控制字符, 如 ASCII 值 0 到 31。

示例:

```
const encodedData = btoa("text to encode"); // 编码字符串
const decodedData = atob(encodedData); // 解码字符串
```

3.2.3.5 内置模块

Buffer

`Buffer` 对象是处理二进制数据的 Node.js 兼容方式。由于文件内容过于庞大, 本节仅限于 Buffer 方法的完整列表。

- `Buffer.alloc()`
- `Buffer.allocUnsafe()`
- `Buffer.byteLength()`
- `Buffer.compare()`
- `Buffer.concat()`
- `Buffer.from(array)`
- `Buffer.from(arrayBuffer)`
- `Buffer.from(buffer)`
- `Buffer.from(object)`
- `Buffer.from(string)`
- `Buffer.isBuffer()`
- `Buffer.isEncoding()`
- `buffer[]`
- `buf.buffer`
- `buf.byteOffset`
- `buf.compare()`
- `buf.copy()`
- `buf.equals()`
- `buf.fill()`
- `buf.includes()`
- `buf.indexOf()`
- `buf.lastIndexOf()`
- `buf.length`
- `buf.readIntBE()`
- `buf.readIntLE()`
- `buf.readUIntBE()`

- `buf.readUIntLE()`
- `buf.readDoubleBE()`
- `buf.readDoubleLE()`
- `buf.readFloatBE()`
- `buf.readFloatLE()`
- `buf.subarray()`
- `buf.slice()`
- `buf.swap16()`
- `buf.swap32()`
- `buf.swap64()`
- `buf.toJSON()`
- `buf.toString()`
- `buf.write()`
- `buf.writeIntBE()`
- `buf.writeIntLE()`
- `buf.writeUIntBE()`
- `buf.writeUIntLE()`
- `buf.writeDoubleBE()`
- `buf.writeDoubleLE()`
- `buf.writeFloatBE()`
- `buf.writeFloatLE()`

有关 Buffer 方法的详细文档, 请参阅 Node.js Buffer 文档。

Crypto

Crypto 模块提供加密功能支持。Crypto 模块对象使用 `import crypto from 'crypto'` 导入。

备注

自 0.7.0 起, 扩展的 crypto API 可作为全局*crypto* 对象使用。

- `crypto.createHash()`
- `crypto.createHmac()`

`crypto.createHash(algorithm)`
: 创建并返回一个 Hash 对象, 可使用给定的 `algorithm` 生成哈希摘要。算法可以是 `md5`、`sha1` 和 `sha256`。

`crypto.createHmac(algorithm, secret key)`
: 创建并返回一个 HMAC 对象, 使用给定的 `algorithm` 和 `secret key`。算法可以是 `md5`、`sha1` 和 `sha256`。

Hash

- `hash.update()`
- `hash.digest()`

`hash.update(data)`
: 使用给定的 `data` 更新哈希内容。

`hash.digest([encoding])`
: 计算使用 `hash.update()` 传递的所有数据的摘要。编码可以是 `hex`、`base64` 和 `base64url`。如果未提供编码, 则返回 Buffer 对象 (0.4.4)。

> **备注**
>
> 在 0.4.4 版本之前, 返回的是字节字符串而不是 Buffer 对象。

`hash.copy()`
: 复制哈希的当前状态 (自 0.7.12 起)。

示例:

```
import crypto from 'crypto';

crypto.createHash('sha1').update('A').update('B').digest('base64url');
/* BtlFlCqiamG-GMPiK_GbvKjdK10 */
```

HMAC

- `hmac.update()`
- `hmac.digest()`

`hmac.update(data)`
: 使用给定的 `data` 更新 HMAC 内容。

`hmac.digest([encoding])`
: 计算使用 `hmac.update()` 传递的所有数据的 HMAC 摘要。编码可以是 `hex`、`base64` 和 `base64url`。如果未提供编码, 则返回 Buffer 对象 (0.4.4)。

备注

在 0.4.4 版本之前, 返回的是字节字符串而不是 Buffer 对象。

fs

`fs` 模块提供文件系统操作。模块对象使用 `import fs from 'fs'` 导入。

- `fs.accessSync()`
- `fs.appendFileSync()`
- `fs.mkdirSync()`
- `fs.readdirSync()`
- `fs.readFileSync()`
- `fs.realpathSync()`
- `fs.renameSync()`
- `fs.rmdirSync()`
- `fs.symlinkSync()`
- `fs.unlinkSync()`
- `fs.writeFileSync()`
- `fs.promises.readFile()`
- `fs.promises.appendFile()`
- `fs.promises.writeFile()`
- `fs.promises.readdir()`
- `fs.promises.mkdir()`
- `fs.promises.rmdir()`
- `fs.promises.rename()`
- `fs.promises.unlink()`
- `fs.promises.symlink()`
- `fs.promises.access()`
- `fs.promises.realpath()`

有关 fs 方法的详细文档, 请参阅 Node.js fs 文档。

Query String

Query String 模块提供解析和格式化 URL 查询字符串的方法。模块对象使用 `import qs from 'querystring'` 导入。

- `querystring.decode()`
- `querystring.encode()`
- `querystring.escape()`
- `querystring.parse()`
- `querystring.stringify()`
- `querystring.unescape()`

`querystring.decode()`
`querystring.parse()` 的别名。

`querystring.encode()`
`querystring.stringify()` 的别名。

`querystring.escape(string)`
以针对 URL 查询字符串要求优化的方式对 `string` 执行 URL 百分号编码。该方法由 `querystring.stringify()` 使用, 不应直接使用。

`querystring.parse(string[, separator[, equal[, options]]])`
将 `string` 解析为 URL 查询字符串并返回一个对象。可选的 `separator` 参数 (默认值: `&`) 指定用于分隔键值对的子字符串。可选的 `equal` 参数 (默认值: `=`) 指定用于分隔键和值的子字符串。可选的 `options` 参数是一个对象, 可能包含以下属性:

`decodeURIComponent`
解码查询字符串中百分号编码字符时使用的函数, 默认值: `querystring.unescape()`。

`maxKeys`
要解析的最大键数, 默认值: `1000`。值 `0` 会移除对键计数的限制。

示例:

```
>> qs.parse('foo=bar&abc=xyz&abc=123')
{
    foo: 'bar',
    abc: ['xyz', '123']
}
```

`querystring.stringify(object[, separator[, equal[, options]]])`
通过迭代 `object` 的自有属性从中生成 URL 查询字符串。可选的 `separator` 参数 (默认值: `&`) 指定用于分隔键值对的子字符串。可选的 `equal` 参数 (默认值: `=`) 指定用于分隔键和值的子字符串。可选的 `options` 参数是一个对象, 可能包含以下属性:

`encodeURIComponent`

将查询字符串中的 URL 不安全字符转换为百分号编码时使用的函数, 默认值: `querystring.escape()`。

示例:

```
>> qs.stringify({ foo: 'bar', baz: ['qux', 'quux'], corge: '' })
'foo=bar&baz=qux&baz=quux&corge='
```

querystring.unescape(string)

对 `string` 中的 URL 百分号编码字符执行解码。该方法由 `querystring.parse()` 使用, 不应直接使用。

XML

- `xml.parse()`
- `xml.c14n()`
- `xml.exclusiveC14n()`
- `xml.serialize()`
- `xml.serializeToString()`
- `XMLDoc`
- `XMLNode`
- `XMLAttr`

XML 模块允许处理 XML 文档 (自 0.7.10 起)。XML 模块对象使用 `import xml from 'xml'` 导入。

示例:

```
import xml from 'xml';

let data = `<note><to b="bar" a= "foo" >Tove</to><from>Jani</from></note>`;
let doc = xml.parse(data);

console.log(doc.note.to.$text) /* 'Tove' */
console.log(doc.note.to.$attr$b) /* 'bar' */
console.log(doc.note.$tags[1].$text) /* 'Jani' */

let dec = new TextDecoder();
let c14n = dec.decode(xml.exclusiveC14n(doc.note));
console.log(c14n) /* '<note><to a="foo" b="bar">Tove</to><from>Jani</from></note>' */

c14n = dec.decode(xml.exclusiveC14n(doc.note.to));
console.log(c14n) /* '<to a="foo" b="bar">Tove</to>' */

c14n = dec.decode(xml.exclusiveC14n(doc.note, doc.note.to /* excluding 'to' */));
console.log(c14n) /* '<note><from>Jani</from></note>' */
```

`parse(string | Buffer)`
: 解析字符串或 Buffer 中的 XML 文档; 返回一个表示已解析 XML 文档的 `XMLDoc` 包装对象。

`c14n(root_node[, excluding_node])`
: 根据 规范 XML 版本 1.1 规范化 `root_node` 及其子节点。`root_node` 可以是围绕 XML 结构的 `XMLNode` 或 `XMLDoc` 包装对象。返回包含规范化输出的 Buffer 对象。

  `excluding_node`
  : 允许从输出中省略文档的一部分。

`exclusiveC14n(root_node[, excluding_node[, withComments[,prefix_list]]])`
: 根据 独占 XML 规范化版本 1.0 规范化 `root_node` 及其子节点。

  `root_node`
  : 围绕 XML 结构的 `XMLNode` 或 `XMLDoc` 包装对象。

  `excluding_node`
  : 允许从输出中省略与该节点及其子节点对应的文档部分。

  `withComments`
  : 布尔值, 默认为 `false`。如果为 `true`, 规范化对应于 带注释的独占 XML 规范化版本 1.0。返回包含规范化输出的 Buffer 对象。

  `prefix_list`
  : 可选字符串, 包含以空格分隔的命名空间前缀, 这些命名空间也应包含在输出中。

`serialize()`
: 与 `xml.c14n()` 相同 (自 0.7.11 起)。

`serializeToString()`
: 与 `xml.c14n()` 相同, 但以 `string` 形式返回结果 (自 0.7.11 起)。

`XMLDoc`
: 围绕 XML 结构的 XMLDoc 包装对象, 文档的根节点。

  `doc.$root`
  : 按名称获取文档的根节点, 如果不存在则为 undefined。

  `doc.abc`
  : 名为 `abc` 的第一个根标签, 作为 `XMLNode` 包装对象。

`XMLNode`
: 围绕 XML 标签节点的 XMLNode 包装对象。

  `node.abc`
  : 与 `node.$tag$abc` 相同。

  `node.$attr$abc`
  : 节点的 `abc` 属性值, 自 0.7.11 起可写。

  `node.$attr$abc=xyz`
  : 与 `node.setAttribute('abc', xyz)` 相同 (自 0.7.11 起)。

`node.$attrs`
节点所有属性的 `XMLAttr` 包装对象。

`node.$name`
节点的名称。

`node.$ns`
节点的命名空间。

`node.$parent`
当前节点的父节点。

`node.$tag$abc`
节点名为 `abc` 的第一个子标签, 自 0.7.11 起可写。

`node.$tags`
所有子标签的数组。

`node.$tags = [node1, node2, ...]`
与 `node.removeChildren(); node.addChild(node1); node.addChild(node2)` 相同 (自 0.7.11 起)。

`node.$tags$abc`
节点所有名为 `abc` 的子标签, 自 0.7.11 起可写。

`node.$text`
节点的内容, 自 0.7.11 起可写。

`node.$text = 'abc'`
与 `node.setText('abc')` 相同 (自 0.7.11 起)。

`node.addChild(nd)`
将 XMLNode 作为子节点添加到该节点 (自 0.7.11 起)。`nd` 在添加到节点之前会被递归复制。

`node.removeAllAttributes()`
删除节点的所有属性 (自 0.7.11 起)。

`node.removeAttribute(attr_name)`
删除名为 `attr_name` 的属性 (自 0.7.11 起)。

`node.removeChildren(tag_name)`
删除所有名为 `tag_name` 的子标签 (自 0.7.11 起)。如果 `tag_name` 不存在, 则删除所有子标签。

`node.removeText()`
删除节点的文本值 (0.7.11)。

`node.setAttribute(attr_name, value)`
为 `attr_name` 设置值 (自 0.7.11 起)。当值为 `null` 时, 删除名为 `attr_name` 的属性。

`node.setText(value)`
为节点设置文本值 (自 0.7.11 起)。当值为 `null` 时, 删除节点的文本。

`XMLAttr`
: 围绕 XML 节点属性的 XMLAttrs 包装对象。

  `attr.abc`
  : `abc` 的属性值。

zlib

`zlib` 模块 (0.5.2) 使用 zlib 提供压缩和解压缩功能。模块对象使用 `import zlib from 'zlib'` 导入。

- `zlib.constants`
- `zlib.deflateRawSync()`
- `zlib.deflateSync()`
- `zlib.inflateRawSync()`
- `zlib.inflateSync()`

`zlib.constants`
: 返回 zlib 常量字典。

`zlib.deflateRawSync(data[, options])`
: 使用 Deflate 算法压缩 `data`, 不包含 zlib 头。

`zlib.deflateSync(data[, options])`
: 使用 Deflate 算法压缩 `data`。

`zlib.inflateRawSync(data[, options])`
: 使用 Deflate 算法解压缩 `data`, 不包含 zlib 头。

`zlib.inflateSync(data[, options])`
: 使用 Deflate 算法解压缩 `data`。

`options` 参数是一个对象, 可能包含以下属性:

`level`
: 压缩级别 (默认值: `zlib.constants.Z_DEFAULT_COMPRESSION`)。

`memLevel`
: 指定应为压缩状态分配多少内存 (默认值: `zlib.constants.Z_DEFAULT_MEMLEVEL`)。

`strategy`
: 调整压缩算法 (默认值: `zlib.constants.Z_DEFAULT_STRATEGY`)。

`windowBits`
: 设置窗口大小 (默认值: `zlib.constants.Z_DEFAULT_WINDOWBITS`)。

`dictionary`
: 包含预定义压缩字典的 Buffer。

`info`
: 布尔值, 如果为 `true`, 返回包含 buffer 和 engine 的对象。

`chunkSize`
: 压缩的块大小 (默认值: `zlib.constants.Z_DEFAULT_CHUNK`)。

示例:

```
import zlib from 'zlib';

const deflated = zlib.deflateSync('Hello World!');
const inflated = zlib.inflateSync(deflated);

console.log(inflated.toString()); // 'Hello World!'
```

您还可以使用 https://angie.ws/ 上的短链接服务快速查找单个指令和变量:

3.2.4 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。

3.2.4.1 HTTP 和核心指令

在核心设置 和*HTTP* 模块 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。

3.2.4.2 流模块指令

流模块 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。

3.2.4.3 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。

3.2.4.4 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 说明

此处提供了配置 Angie 特定方面的分步说明。

3.3.1 ACME 配置

Angie 中的*ACME* 模块使用 ACME 协议 实现自动证书获取。ACME 协议支持多种域名验证方法 (也称为" 验证"); 本模块实现了*HTTP* 验证、*DNS* 验证、*ALPN* 验证 以及通过自定义外部服务进行的基于钩子的验证。

3.3.1.1 配置步骤

在配置中启用证书请求的常规步骤:

- **配置 ACME 客户端**: 在 `http` 块中使用*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。
- **指定请求证书的域名**: 将为所有 `server` 块中*server_name* 指令列出的所有域名颁发单个证书, 这些块使用指向同一 ACME 客户端的*acme* 指令。
- **设置请求处理和 ACME 回调**: 这是验证域名所有权所必需的。设置取决于所选的域名验证方法:

方法	用户要求	多域名	通配符域名
HTTP 验证	在 Angie 服务器上开放端口 80(或*acme_http_port* 中指定的端口) 以接受传入连接。	✓	
DNS 验证	在 Angie 服务器上开放端口 53(或*acme_dns_port* 中指定的端口) 以接受传入连接。 为 `_acme-challenge.` 子域设置 NS 记录, 指向您的 Angie 服务器。	✓	✓
ALPN 验证	在 Angie 服务器上开放端口 443(或您的服务器使用的 TLS 端口) 以接受传入连接。	✓	
钩子验证	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。	✓	✓

- **使用获取的证书和密钥配置 SSL**: 该模块将证书和密钥作为嵌入式变量 提供, 可在配置 中使用以填充*ssl_certificate* 和*ssl_certificate_key*。

有关 SSL 设置说明, 请参阅*SSL 配置*。

小技巧

证书获取和续订过程依赖于许多服务, 可能需要一些时间。请耐心等待, 如果遇到问题或有疑问, 请检查调试日志。

3.3.1.2 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子目录中:

```
$ ls /var/lib/angie/acme/example/

  account.key   certificate.pem   private.key
```

备注

这些文件在重启之间持久保存在磁盘上。启动时, 客户端会复用仍然有效的已存储证书, 而非请求新证书, 从而避免了签发延迟和向 CA 发送不必要的请求——CA 可能受 频率限制 约束。在容器中使用时, 请将存储目录（默认为 `/var/lib/angie/acme/`）挂载到持久卷上, 以确保已签发的证书在容器重建后仍然存在; 请参阅 在容器中运行 Angie。

ACME 客户端需要在 CA 服务器上拥有一个账户。为了创建和管理此账户, 客户端使用私钥 (`account.key`)。如果不存在密钥, 则在启动时生成。然后客户端使用此密钥向服务器注册账户。

备注

如果您已经拥有账户密钥，请在启动前将其放置在客户端的子目录中以重用该账户。或者，使用*acme_client* 中的 `account_key` 参数指定密钥文件。

ACME 客户端还使用单独的密钥 (`private.key`) 用于证书签名请求 (CSR)。如果需要, 此证书密钥会在启动时自动创建。

启动时, 如果证书不存在, 客户端会请求证书, 签名并向 CA 服务器发送包含其管理的所有域名的 CSR。服务器使用*HTTP* 或*DNS 验证* 验证域名所有权, 并颁发证书, 客户端将其保存在本地 (`certificate.pem`)。

如前所述, 单个证书涵盖同一 ACME 客户端管理的所有域名, 可能产生多域名证书。证书涵盖的所有名称列表可以在获取的证书的 *Subject Alternative Name* (SAN) 部分中找到。要从命令行检查:

```
$ openssl x509 -in certificate.pem -noout -text | grep -A5 "Subject Alternative Name"
```

当证书即将过期或域名列表发生变化时, 客户端会签名并向 CA 服务器发送另一个 CSR。服务器重新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>* 获得。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;
}
```

备注

ACME 客户端通过配置的*resolver* 解析 CA 服务器的主机名, 该解析器必须存在于相同上下文中。在没有 IPv6 连接的主机上, 解析器仍可能向 CA 发起 AAAA（IPv6）查询并导致连接失败; 可通过 `ipv6=off` 参数禁用此行为, 例如 `resolver 127.0.0.53 ipv6=off;`。

3.3.1.3 域名收集与证书使用

acme 指令仅用于收集证书请求的域名, 并不控制证书可在何处使用: 任何 `server` 块都可以通过*$acme_cert_<name>* 变量引用获取的证书, 无论该块是否包含 `acme` 指令。

例如, 如果您有一个已涵盖所有子域名的通配符 `server` 块, 那么针对特定子域名的额外 `server` 块不需要 `acme` 指令:

```
http {

    resolver 127.0.0.53;

    acme_client example https://acme-v02.api.letsencrypt.org/directory
        challenge=dns;

    # This block lists the domains for the certificate request
    server {

        listen 443 ssl;

        server_name example.com *.example.com;
        acme example;

        ssl_certificate $acme_cert_example;
        ssl_certificate_key $acme_cert_key_example;
    }

    # This block uses the same certificate but does not
    # add its server_name to the certificate request
    server {
```

```
        listen 443 ssl;

        server_name app.example.com;

        ssl_certificate $acme_cert_example;
        ssl_certificate_key $acme_cert_key_example;
    }
}
```

显式域名列表

为了精确控制证书中的域名集合，而不依赖于从所有 `server` 块自动收集，可以创建一个专用的 `server` 块，其中仅包含*server_name* 和*acme* 指令。为防止此块处理实际流量，可以将其绑定到 Unix 域套接字：

```
# Dedicated block that defines the certificate's domain list
server {

    listen unix:/tmp/acme_example.sock;

    server_name example.com www.example.com;
    acme example;
}
```

其他 `server` 块随后可通过*$acme_cert_<name>* 变量使用该证书，而不会影响请求的域名。

为不同域名申请独立证书

每个*acme_client* 只管理一张证书。若要获取多张独立证书（例如，用于不应共享同一证书的不相关域名），请在 `http` 块中为每个域名单独配置一个*acme_client*，并在每个 `server` 块的*acme* 指令中按名称指向相应的客户端：

```
http {

    resolver 127.0.0.53;

    # 两个独立的客户端，各自管理自己的证书
    acme_client shop https://acme-v02.api.letsencrypt.org/directory
        challenge=http;

    acme_client blog https://acme-v02.api.letsencrypt.org/directory
        challenge=http;

    server {

        listen 443 ssl;

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

```
        acme shop;

        ssl_certificate $acme_cert_shop;
        ssl_certificate_key $acme_cert_key_shop;
    }

    server {

        listen 443 ssl;

        server_name blog.example.com;
        acme blog;

        ssl_certificate $acme_cert_blog;
        ssl_certificate_key $acme_cert_key_blog;
    }
}
```

3.3.1.4 HTTP 验证

验证是自动处理的。该过程涉及 ACME 服务器在收到请求后,'通过 HTTP 检索特殊令牌文件 <https://datatracker.ietf.org/doc/html/rfc8555#section-8.3>'_, 从客户端地址 `/.well-known/acme-challenge/<TOKEN>` 获取。我们的 ACME 模块会自动跟踪和处理此类请求。当收到包含正确内容的预期响应时,ACME 服务器确认该域名属于客户端。

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 (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; # 如果没有服务器监听 HTTP 挑战端口,则可选
                   # (参见 'acme_http_port' 指令)

        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, 必须开放端口 80 以处理 HTTP ACME 调用。如果没有配置服务器监听 HTTP 挑战端口, 该模块会在端口 80(或*acme_http_port* 中设置的端口) 上创建专用监听器。不需要单独的*server* 块。

3.3.1.5 DNS 验证

验证是自动处理的。在处理证书请求时,ACME 服务器会对被验证域名的 `_acme-challenge.` 子域执行特殊的 DNS 查询。一旦收到预期的响应,ACME 服务器确认该域名属于客户端。

我们的 ACME 模块会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。

备注

Angie 服务器必须能够通过互联网在 UDP 53 端口 (或*acme_dns_port* 中指定的端口) 上访问。如果服务器位于防火墙后, 请确保该端口对入站连接开放。

例如, 要使用 IP 地址为 `93.184.215.14` 的 Angie 服务器验证域名 `example.com`, 您的域名 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`, 并通过将 `ns.example.com` 映射到 IP 地址 (`93.184.215.14`) 来确保其可访问。

警告

NS 记录的传播可能需要几分钟到 48 小时不等, 具体取决于 TTL 和 DNS 提供商。建议在请求证书之前验证配置是否正确。

要验证 DNS 是否配置正确, 您可以使用以下命令:

```
$ dig NS _acme-challenge.example.com +short  # 检查 _acme-challenge 子域的 NS 记录

  ns.example.com.

$ dig A ns.example.com +short  # 检查名称服务器的 A 记录

  93.184.215.14

$ nc -zv 93.184.215.14 53  # 检查 DNS 服务器在端口 53 上的可访问性
```

此方法允许请求通配符证书, 例如, 在 *Subject Alternative Name* (SAN) 部分包含 `*.example.com` 条目的证书。要明确请求子域的证书, 例如 `www.example.com`, 您必须使用上述方法单独验证该子域。

警告

此场景的适用性在很大程度上取决于您的 DNS 提供商提供的功能; 某些提供商不允许此类配置。

配置示例

总体而言，配置与上一节中的示例类似。不需要 HTTP 特定的设置；只需为*acme_client* 指令设置 `challenge=dns` 即可。

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;
    }
}
```

3.3.1.6 ALPN 验证

验证是自动处理的。ACME 服务器使用 TLS 连接并通过 ALPN 请求 `acme-tls/1` 协议。该模块为验证请求提供临时证书。

要启用此方法, 请在*acme_client* 指令中配置 `challenge=alpn`, 并确保您的 TLS 监听器在端口 443(或用于 TLS 的端口) 上可访问。

配置示例

配置与前几节类似; 只需为*acme_client* 指令设置 `challenge=alpn`, 并确保 TLS 服务器在端口 443 上可访问即可。

在此示例中, 名为 `example` 的 ACME 客户端管理 `example.com` 和 `www.example.com` 的证书:

```
http {

    resolver 127.0.0.53;
```

```
    acme_client example https://acme-v02.api.letsencrypt.org/directory
        challenge=alpn;

    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;
    }
}
```

3.3.1.7 基于钩子的验证

与之前的方法不同, 此验证需要额外的工作。ACME 服务器执行标准的*HTTP 验证* 或*DNS 验证*, 但它不是直接与 Angie 服务器交互, 而是通过钩子调用 (*acme_hook*) 与 Angie 服务器管理的外部服务通信。该服务配置一个单独的 DNS 或 HTTP 服务器,ACME 服务器向其发送请求。

一旦 ACME 服务器从配置的 DNS 或 HTTP 服务器收到预期的响应, 它就会确认域名所有权。

当证书签发或续订需要进行域名验证时, Angie 会向包含*acme_hook* 指令的命名 `location` 生成一个内部请求。该请求的处理方式完全取决于同一 `location` 中配置的其他指令。

通用模式如下:

1. 创建一个带有*acme_hook* 指令的命名 `location`。
2. 在同一 `location` 中, 使用适合您环境的模块配置请求处理程序: FastCGI 使用*fastcgi_pass*, HTTP 使用*proxy_pass*, CGI 脚本使用 `cgi`, 等等。
3. 使用处理程序支持的机制, 将 ACME 变量 传递给处理程序, 例如 FastCGI 使用 `fastcgi_param`, CGI 使用 `cgi_set_var`。

处理程序必须返回 `2xx` 状态码, 可以通过 `Status` 头发送。任何其他状态码都被视为错误, 证书续订将停止。处理程序的输出将被忽略。

最小配置

无论使用哪种处理程序, 钩子 `location` 都遵循以下结构:

```
location @acme_hook_location {

    acme_hook example;

    # Handler directive (fastcgi_pass, proxy_pass, cgi on, ...)
    # Pass ACME variables using the handler's mechanism:
```

```
    #   ACME_HOOK       — $acme_hook_name ("add" or "remove")
    #   ACME_CHALLENGE  — $acme_hook_challenge ("dns" or "http")
    #   ACME_DOMAIN     — $acme_hook_domain
    #   ACME_TOKEN      — $acme_hook_token
    #   ACME_KEYAUTH    — $acme_hook_keyauth
}
```

对于 DNS 验证, 处理程序必须使用 `ACME_HOOK` 来确定操作: 当其值为 `add` 时, 为 `_acme-challenge.`*`ACME_DOMAIN`* 创建一条值为 `ACME_KEYAUTH` 的 TXT 记录; 当其值为 `remove` 时, 删除该记录。

FastCGI 示例

在此示例中,:ref:*acme_client* 指令通过 `challenge=dns` 参数将 ACME 客户端 `example` 配置为使用 DNS 验证。

`server` 块适用于 `example.com` 的所有子域 (例如 `*.example.com`), 并通过*acme* 指令使用 ACME 客户端 `example` 来管理证书。

一个命名的 `location` 块负责处理钩子调用。*acme_hook* 指令将其与 ACME 客户端 `example` 关联起来。钩子请求通过*fastcgi_pass* 发送到运行在 9000 端口上的本地 FastCGI 服务器。`fastcgi_param` 指令将 ACME 变量传递给外部服务。

```
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 Perl 脚本演示了相应的外部 FastCGI 服务:

```
#!/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,:samp:*DNS_set_TXT_record()* 和 `DNS_clear_TXT_record()` 是假定在外部 DNS 服务器的配置中添加和删除 TXT 记录的函数,ACME 服务器会查询该 DNS 服务器。这些记录必须包含 Angie 服务器提供的数据, 以允许外部 DNS 服务器成功通过验证, 类似于*DNS 验证* 中描述的过程。此类函数的实现细节超出了本指南的范围; 例如, 参数也可以通过请求 URI 传递:

```
# ...

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;
}
```

PHP-FPM 示例

另一个使用 PHP-FPM 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。

3.3.1.8 Stream 模块中的 ACME

ACME stream 模块可为 TCP 流量启用自动化的证书签发和使用。为使其正常工作, 您必须首先配置其 HTTP 对应部分: ACME 客户端必须在 `http` 上下文中声明, 并且 `stream` 块本身必须放置在配置文件中 `http` 块 * 之后 *。

配置示例

默认情况下，使用 HTTP 验证模式来获取证书。如*HTTP 验证* 部分所述，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;
    }
}
```

您也可以使用 DNS 验证，方法是在*acme_client* 指令中配置 `challenge=dns`；在这种情况下，将不需要该服务器。

3.3.1.9 从 certbot 迁移

如果您之前在从 *nginx* 迁移到 *Angie* 之前使用 certbot 从 Let's Encrypt 获取和续订 SSL 证书，请按照以下步骤过渡到使用我们的 ACME 模块。

假设您按如下方式配置了证书:

```
$ sudo certbot --nginx -d example.com -d www.example.com
```

此命令自动创建的配置通常位于 `/etc/nginx/sites-available/example.conf`，看起来类似这样:

```
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;
}
```

在上面的示例中，需要修改高亮显示的行。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;
    }
}
```

请记住在更改后重新加载配置:

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

一旦您验证了此配置可以正常工作, 您可以删除 **certbot** 证书并禁用或从服务器上完全移除 certbot (如果它不再在其他地方使用), 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
```

3.3.2 Angie 集群设置

本指南介绍了创建具有自动配置同步和虚拟 IP 地址故障转移的容错 Angie 集群的过程。

3.3.2.1 准备集群节点以进行同步

第一步是通过配置用户账户并确保服务器之间的安全访问来准备所有集群节点。

配置用户和访问权限

在所有节点上创建一个用户 (例如,:samp:*angie-ha-sync*) 并赋予 sudo 权限:

```
$ sudo adduser angie-ha-sync
```

如有必要, 设置密码:

```
$ sudo passwd angie-ha-sync
```

备注

在某些操作系统中 (例如 Alt Linux), 您应该将用户添加到 `wheel` 组:

```
$ sudo usermod -a -G wheel angie-ha-sync
```

在 Astra Linux 中启用 MAC 时要使用 `rsync`, 请设置正确的完整性级别:

```
$ sudo pdpl-user -i 63 angie-ha-sync
```

配置无密码 sudo:

```
$ echo "angie-ha-sync ALL=(ALL:ALL) NOPASSWD:ALL" | sudo tee -a /etc/sudoers
```

在主节点上, 创建 SSH 密钥并将其复制到备份节点:

```
$ su - angie-ha-sync
$ ssh-keygen -t rsa
$ ssh-copy-id angie-ha-sync@node2_hostname
```

警告

在复制 SSH 密钥之前, 请确保 `/etc/ssh/sshd_config` 文件中有以下选项:

```
PasswordAuthentication yes
```

设置基于密钥的访问后, 将该值设置为 `no` 以提高安全性。

备注

对于 Angie 配置的交叉同步, 请将用户密钥复制到所有节点:

```
$ scp -p ~angie-ha-sync/.ssh/id_rsa angie-ha-sync@node2_hostname:.ssh/
```

3.3.2.2 安装 Angie 和 angie-ha-sync

准备好节点后, 您需要安装主要的集群组件: Angie 和配置同步包。

根据您的系统包的说明在所有节点上配置仓库:

- Angie 说明
- Angie PRO 说明

安装 angie-ha-sync

配置同步模块在以下包中可用:

- Angie:`angie-ha-sync`
- Angie PRO:`angie-pro-ha-sync`

备注

在干净的系统上安装此包时, 相应的 `angie` 或 `angie-pro` 包也将作为依赖项安装。

在所有节点上, 使用您的操作系统包管理器安装该包, 例如:

```
$ sudo {apk|apt|pkg|yum|zypper} {add|install} angie-pro-ha-sync
```

3.3.2.3 配置配置同步

下一步是设置集群节点之间配置文件的自动同步。

备注

同步原则:

- 通过 `rsync` 执行同步。
- 仅在 Angie 服务运行时发生。
- 手动执行 (命令 `angiehasync -Sd`)。
- 单向工作: 从主节点到备份节点。
- `rsync` 以守护进程模式运行。

配置 rsync

在节点上创建 `rsync` 配置 (`/etc/rsyncd.conf`):

```
[angie] # Angie 配置目录
    path = /etc/angie
# 用于同步的用户
    uid = angie-ha-sync
# 用户组
    gid = angie-ha-sync
# 允许连接的 IP 或子网
    hosts allow = 10.21.8.0/24
# 拒绝所有其他
    hosts deny = *
```

根据操作系统, 启动守护进程:

```
$ sudo service rsyncd start # 或 $ sudo service rsync start
```

备注

对于某些系统, 有现成的说明:

- Alt
- Astra

配置同步文件

编辑 /etc/angiehasync/angiehasync.conf:

```
M_NODE="<node1_hostname>"             # 此节点的主机名或 IP
TARGET_HOSTS="<node2_hostname>"       # 用于同步的主机/IP 列表(空格分隔)。
                                      # 在备份节点上可以省略。
SSH_USER="user"                       # 用于同步的用户(具有管理员权限)
SSH_ID="/home/$SSH_USER/.ssh/id_rsa" # 私钥路径
```

备注

对于交叉同步, 在所有节点上填写 TARGET_HOSTS 列表; 但是, 不要在列表中包含当前正在配置的节点。

配置 Angie 的健康检查

在 Angie 配置中添加健康检查块 (/etc/angie/angie.conf):

```
server {

    listen unix:/tmp/angie_hcheck.sock; # 用于检查的 Unix 套接字
    access_log off;
    error_log /dev/null;
    default_type text/plain;
    return 200 'ok\n';
}
```

启动 Angie:

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

启动同步:

```
$ sudo angiehasync -Sd
```

备注

该脚本将自动检查配置, 与所有节点执行同步, 并应用它。

3.3.2.4 配置 Keepalived

为了在集群节点之间实现自动故障转移, 使用 Keepalived —一个用于管理虚拟 IP 地址 (VIP) 的服务。

备注

如果未安装 `keepalived` 包—请安装它:

```
$ sudo {apk|apt|pkg|yum|zypper} {add|install} keepalived
```

要将进程绑定到非本地 IP 地址, 允许系统执行相应的操作:

```
$ sudo sysctl -w net.ipv4.ip_nonlocal_bind=1
```

更多详情: https://www.kernel.org/doc/Documentation/networking/ip-sysctl.txt#ip_nonlocal_bind

假设 VIP `10.21.11.230` 分配给主节点 (`10.21.8.26`) 或备份节点 (`10.21.8.27`)。

如果 Angie 监听此 VIP(`listen 10.21.11.230:80;`) 但地址尚未分配, 没有 `ip_nonlocal_bind` 参数,Angie 将无法启动。

Keepalived 配置

在主节点上 (`/etc/keepalived/keepalived.conf`):

```
global_defs {
    enable_script_security
}

vrrp_script angie_check {
    script "/usr/bin/curl -s --connect-timeout 5 -A 'angie_hcheck_script'
    --no-buffer -XGET --unix-socket /tmp/angie_hcheck.sock http://hcheck/"
    interval 5 user angie
}

vrrp_instance angie {
    state MASTER interface enp0s2 virtual_router_id 254 priority 100
    advert_int 2 unicast_src_ip 10.21.8.26

    unicast_peer {
        10.21.8.27
    }

    virtual_ipaddress {
        10.21.11.230
```

```
    } track_script {
        angie_check
    }
}
```

在备份节点上:

```
global_defs {
    enable_script_security
}

vrrp_script angie_check {
    script "/usr/bin/curl -s --connect-timeout 5 -A 'angie_hcheck_script'
    --no-buffer -XGET --unix-socket /tmp/angie_hcheck.sock http://hcheck/"
    interval 5 user angie
}

vrrp_instance angie {
    state MASTER interface enp0s2 virtual_router_id 254 priority 99
    advert_int 2 unicast_src_ip 10.21.8.27

    unicast_peer {
        10.21.8.26
    }

    virtual_ipaddress {
        10.21.11.230
    } track_script {
        angie_check
    }
}
```

备注

在 `vrrp_instance angie` 部分中, 设置以下值:

- `unicast_src_ip` —当前节点的 IP
- `unicast_peer` —相邻节点的 IP
- `virtual_ipaddress` —虚拟 IP(VIP)
- `interface` —网络接口

启动服务:

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

Keepalived 配置详解

让我们详细研究 Keepalived 配置的主要元素，以了解集群的运行原理。

配置包含两个部分：

- `global_defs` —全局设置
- `vrrp_instance` —VRRP 参数 (VIP 切换)

主要元素：

- `enable_script_security` —允许执行健康检查脚本
- `vrrp_script` —Angie 健康检查脚本
- `state MASTER` —初始节点状态
- `priority` —优先级 (MASTER 角色分配给优先级最高的节点)
- `advert_int` —VRRP 通告间隔
- `unicast_src_ip` —当前节点 IP
- `unicast_peer` —邻居节点 IP
- `virtual_ipaddress` —VIP 地址
- `track_script` —通过健康检查脚本进行可用性监控

备注

如果原主节点恢复，它将重新获得 MASTER 角色 (因为优先级更高)。要禁用故障恢复，请使用 `nopreempt` 参数：

```
vrrp_instance angie {
    ... nopreempt
}
```

3.3.2.5 测试集群运行

完成配置后，需要测试集群运行情况并确保节点之间正确切换。

检查 VIP 状态：

```
$ ip addr show enp0s2 | grep "10.21.11.230"
```

测试容错能力：

在主节点上停止 Angie：

```
$ sudo service angie stop
```

检查 VIP 是否转移到备份节点：

```
$ ip addr show enp0s2 | grep "10.21.11.230"
```

再次在主节点上启动 Angie:

```
$ sudo service angie start
```

此后,VIP 应该返回到主节点。

3.3.3 OIDC 身份验证设置

本指南说明如何使用 Google 作为身份提供商和 Angie Web 服务器通过 Lua 脚本设置 OpenID Connect (OIDC) 身份验证。

该实现使用 OAuth2/OIDC 身份验证保护内部端点, 并演示了一种基于电子邮件域限制访问的方法。这只是一种示例方法; 您可以按照自己喜欢的方式实现访问控制, 例如维护特定用户的允许列表、检查提供商响应中的域成员资格或组属性, 或使用来自您私有 IAM 系统的自定义声明。

小技巧

此 OIDC 实现为身份验证提供了基础, 但应根据生产环境进行调整, 采用适当的安全措施、监控以及符合您组织的安全策略。

3.3.3.1 架构

此处建议的 OIDC 设置包括:

- Angie - 支持 Lua 模块以进行 OIDC 处理
- lua-resty-openidc - 用于 OIDC 身份验证的 OpenResty Lua 库
- Google OAuth2 - 用于用户身份验证的身份提供商
- Docker Compose - 在本示例中仅用于快速启动; 在生产环境中使用您喜欢的任何部署方法

3.3.3.2 前提条件

在配置 OIDC 身份验证之前, 请确保您具备:

1. 支持 Lua 模块的 Angie Web 服务器
2. Docker 和 Docker Compose(用于部署)
3. Google Cloud Console 项目
4. 来自 Google 的 OAuth2 凭据

3.3.3.3 Google OAuth2 设置

要将 Google 配置为您的 OIDC 提供商:

1. 导航到 Google Cloud Console
2. 创建新项目或选择现有项目
3. 为您的项目配置 OAuth 同意屏幕 (外部或内部) 并发布它, 以便用户可以进行身份验证
4. 创建 OAuth2 凭据:
   - 应用类型: Web 应用程序
   - 已授权的重定向 URI: `http://localhost/auth/callback`
5. 保存您的 `client_id` 和 `client_secret` 以供配置使用

备注

标准 Google Identity Services 已经支持 OIDC; 不需要旧版 Google+ API。仅当您的应用程序需要其数据时才启用其他 Google API。

3.3.3.4 配置设置

让我们从 OIDC 设置所需的配置文件开始。

Docker Compose 配置

Docker 部署使用以下配置文件:

Listing 1: docker-compose.yml

```
services:
  angie:
    image: docker.angie.software/angie:templated
    environment:
      ANGIE_LOAD_MODULES: "lua"
    ports:
      - 80:80
    volumes:
      - ./files/etc/angie/http.d:/etc/angie/http.d
```

此配置执行以下操作:

- 使用支持 Lua 模块的 模板化 Angie 镜像
- 加载 Lua 模块以实现 OIDC 功能
- 映射端口 80 以进行 HTTP 访问
- 从本地目录挂载配置文件

对于即插即用的演示，下载 OIDC 快速启动包，在 files/etc/angie/http.d/oidc.lua 中设置您的 client_id 和 client_secret，一切都将开箱即用。

OIDC 身份验证脚本

创建一个 OIDC 身份验证脚本，使用 lua-resty-openidc 库处理身份验证逻辑：

Listing 2: /etc/angie/http.d/oidc.lua

```
access_by_lua_block {
    local res, err = require("resty.openidc").authenticate({
        redirect_uri = "http://localhost/auth/callback",
        discovery = "https://accounts.google.com/.well-known/openid-configuration",
        logout_path = "/auth/logout",
        redirect_after_logout_uri = "/auth/logged-out",
        revoke_tokens_on_logout = true,
        client_id = "YOUR_CLIENT_ID",
        client_secret = "YOUR_CLIENT_SECRET"
    })
}
```

配置参数：

- redirect_uri: 成功身份验证后的回调 URL
- discovery: Google 的 OIDC 发现端点
- logout_path: 用户注销路径
- redirect_after_logout_uri: 注销后的重定向目标
- revoke_tokens_on_logout: 注销时撤销令牌以确保安全
- client_id 和 client_secret: 您的 Google OAuth2 凭据

Angie 配置

使用必要的*location* 块配置 Angie 以进行 OIDC 身份验证。

受保护的资源

要使用 OIDC 身份验证保护资源：

```
location /internal/ {
    include /etc/angie/http.d/oidc.lua;
    proxy_pass http://127.0.0.1/status/;
}
```

此配置执行以下操作：

- 使用 OIDC 身份验证保护 /internal/ 路径

- 将经过身份验证的请求代理到 `/status/` 的内部 *API*

身份验证端点

配置 OAuth2 流程端点:

```
location /auth/callback {
    include /etc/angie/http.d/oidc.lua;
}

location /auth/logout {
    include /etc/angie/http.d/oidc.lua;
}

location /auth/logged-out {
    default_type text/plain;
    return 200 "You have been logged out. Bye!";
}
```

端点功能:

- `/auth/callback`: 处理来自 Google 的 OAuth2 回调
- `/auth/logout`: 启动用户注销
- `/auth/logged-out`: 成功注销后的着陆页

内部 API 访问

配置对内部 API 的受限访问:

```
location /status/ {
    api     /status/;
    allow   127.0.0.1;
    deny    all;
}
```

这提供了:

- 访问 Angie 的状态 *API*
- 仅限本地主机访问 (127.0.0.1)
- 通过 `/internal/` 访问时的 OIDC 保护

3.3.3.5 部署步骤

按照以下步骤部署 OIDC 身份验证:

配置更新

1. 在您的 OIDC Lua 脚本中更新 OAuth2 凭据:

   替换 `oidc.lua` 中的占位符值:

   - 将 `client_id` 替换为您的 Google OAuth2 客户端 ID
   - 将 `client_secret` 替换为您的 Google OAuth2 客户端密钥

服务启动

1. 启动 Docker 服务:

   ```
   $ docker-compose up -d
   ```

2. 验证部署:

   - 导航到 `http://localhost/internal/`
   - 您应该被重定向到 Google 进行身份验证
   - 成功登录后, 您将访问受保护的内容

3.3.3.6 安全配置

电子邮件域限制

实现域验证以按电子邮件域限制访问:

```
if not string.match(res.user.email, "gmail.com$") then
    ngx.exit(ngx.HTTP_FORBIDDEN)
end
```

对于生产环境, 请考虑以下事项:

- 将 `gmail.com` 替换为您组织的域
- 实现允许的电子邮件地址白名单
- 添加基于角色的访问控制

令牌管理

OIDC 实现包括安全功能:

- 注销时自动撤销令牌 (`revoke_tokens_on_logout = true`)
- lua-resty-openidc 库安全地管理会话
- 所有身份验证流程都遵循 OAuth2/OIDC 安全最佳实践

警告

对于生产部署:

- 始终对 OAuth2 回调使用 HTTPS
- 安全地存储客户端密钥, 切勿存储在版本控制中
- 实施适当的会话超时和续订策略
- 监控身份验证日志以发现安全事件

3.3.3.7 身份验证流程

OIDC 身份验证流程遵循标准 OAuth2/OIDC 程序:

1. 用户访问受保护的 URL(例如 `http://localhost/internal/`)
2. 如果未经身份验证, 用户将被重定向到 Google OAuth2
3. 用户使用 Google 凭据进行身份验证
4. Google 使用授权码重定向回 `/auth/callback`
5. 服务器将代码交换为访问令牌和 ID 令牌
6. 验证用户信息 (包括电子邮件域检查)
7. 授予用户访问受保护资源的权限

注销过程确保安全的会话终止:

1. 用户导航到 `http://localhost/auth/logout`
2. 在 Google 的 OAuth2 端点撤销令牌
3. 清除本地会话数据
4. 用户被重定向到 `/auth/logged-out`

3.3.3.8 高级配置

要限制对特定组织域的访问:

```
if not string.match(res.user.email, "yourcompany.com$") then
    ngx.exit(ngx.HTTP_FORBIDDEN)
end
```

对于具有多个允许域的组织:

```
local allowed_domains = {"company1.com", "company2.com", "gmail.com"}
local email_valid = false

for _, domain in ipairs(allowed_domains) do
```

```
    if string.match(res.user.email, domain .. "$") then
        email_valid = true
        break
    end
end

if not email_valid then
    ngx.exit(ngx.HTTP_FORBIDDEN)
end
```

用户信息访问

从 OIDC 提供商访问其他用户声明:

```
-- 从 ID 令牌访问用户信息
local user_name = res.user.name
local user_picture = res.user.picture
local user_locale = res.user.locale
local user_email = res.user.email
```

这些值可用于日志记录、个性化或其他访问控制决策。

3.3.4 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。

3.3.4.1 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;
    #...
```

3.3.4.2 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。

3.3.4.3 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;
#...
}
```

3.3.4.4 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;
#...
}
```

3.3.4.5 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;
#...
}
```

3.3.4.6 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

3.3.5 Console Light Web 监控面板

Angie 提供了广泛的监控工作方式; 除了指标 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。

3.3.5.1 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。该控制台基于 Angie 的*API* 功能; 活动监控数据实时生成。此外, 该控制台允许您在 API 本身提供此功能的情况下动态修改 Angie 配置。

已部署和配置的控制台示例:https://console.angie.software/

3.3.5.2 版本历史

版本	发布日期	变更
1.8.2	23.01.2026	修复了指向 Angie ADC 文档的链接。
1.8.1	08.09.2025	修复了 Settings 和工具提示中的错误术语。
1.8.0	03.07.2025	显示代理 HTTP 和 TCP/UDP 服务器的响应时间指标
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" 选项。
1.7.1	04.04.2025	修复了 HTTP Zones 页面上 HTTP/Location Zones 表中的错误值。
1.7.0	02.04.2025	• 鼠标悬停时显示精确的字节数据量 • 统计 API 中上游对等节点的新 `busy` 状态, 表示对等节点已达到 `max_conns` 参数配置的限制 • 修复了文档链接
1.6.1	27.01.2025	• 修复了拼写错误 • 修复了开发时项目构建问题
1.6.0	23.01.2025	• 国际化支持, 可用语言环境:`en`、`ru`。 • 为表格组件添加了粘性标题功能。 • 支持 pebibytes (PiB) 数据测量单位。 • 修复了主页面*HTTP Upstreams* 小部件中的错误值计数器。 • 现在在响应上下文中的*HTTP Upstreams* 页面上正确使用默认值。
1.5.0		未公开发布。
1.4.0	08.08.2024	在网站图标中添加了监控状态显示。
1.3.0	28.04.2024	添加了在上游上下文中将服务器设置为 `draining` 状态的功能。
1.2.1	26.12.2023	在 `Stream` 上下文中添加了主动健康检查。
1.2.0	25.12.2023	在 `Stream` 上下文中添加了服务器编辑功能。

3.3.5.3 安装和配置

Console Light 以 `angie-console-light` (Angie) 和 `angie-pro-console-light` (Angie PRO) 软件包的形式发布在 我们的仓库中, 可以像安装任何其他软件包一样安装; 或者, 您可以从 我们的网站 或 GitHub 下载源代码。

安装后, 通过在服务器配置 的*server* 块内添加以下*location* 来配置控制台 (注意注释):

```
location /console/ {

    # 仅本地访问
```

```
    allow 127.0.0.1;
    deny all;

    auto_redirect on;

    alias /usr/share/angie-console-light/html/;
    # 仅 FreeBSD:
    # alias /usr/local/www/angie-console-light/html/;
    index index.html;

    location /console/api/ {
        api /status/;
    }

    # 为了在身份验证后使编辑功能正常工作(仅 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, 路径为 `location` 指定的路径; 在上面的示例中, 路径设置为 `/console/`。

可以为任何 API 部分启用身份验证, 类似于上面的示例, 例如:

```
location /console/server_zones/ {
    auth_basic           "Protected site";
    auth_basic_user_file conf/htpasswd;
}
```

您还可以限制对已配置控制台 `location` 的任何部分的访问, 例如:

```
location /console/api/resolvers/ {
    deny all;
}
```

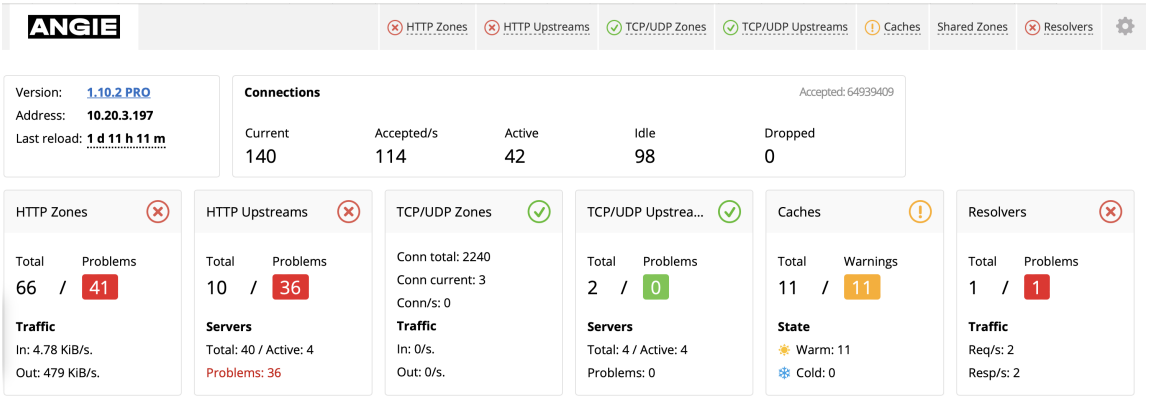
3.3.5.4 界面

控制台是一个带有一组选项卡的单一屏幕，每个选项卡包含多个带有监控数据的小部件。

小技巧

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，其中以摘要形式显示关键的 Angie 监控指标，基于来自多个 API 部分的数据。

备注

如果在*Angie 配置* 中配置了相应的块，则会显示统计小部件。

About 小部件

显示 Angie 版本号及指向相应文档的链接，以及服务器地址和上次配置重新加载 的时间。

此外，如果启用了*api_config_files* 指令，*Configs* 链接会打开服务器上加载的配置文件列表。然后可以以紧凑格式查看每个文件，并带有语法高亮显示。

Connections 小部件

显示基本的服务器连接统计信息，从 `/status/connections/` API 部分生成：

`Current`	当前连接数
`Accepted/s`	每秒接受的连接数
`Active`	活动连接数
`Idle`	空闲连接数
`Dropped`	丢弃的连接数

还可用:

`Accepted`	自上次服务器重新加载以来接受的连接总数

HTTP Zones 小部件

警告

需要在 `server` 或 `location` 上下文中设置*status_zone* 指令。

显示 `http` 上下文的共享内存区域统计信息, 从*/status/http/server_zones/* API 部分生成:

`Total`	区域总数
`Problems`	存在任何问题的区域数
`Traffic`	传入和传出流量总量

HTTP Upstreams 小部件

警告

需要在 `http` 上下文的*upstream* 块中设置*zone* 指令。

显示 `http` 上下文的上游统计信息, 从*/status/http/upstreams/* API 部分生成:

Total	上游总数
Problems	存在任何问题的上游数量
Servers	按状态细分的服务器统计信息

TCP/UDP Zones 小部件

警告

需要设置以下指令:

- 在*server* 或*stream* 上下文中设置 `status_zone`;
- 在*server* 或*stream* 上下文中设置 `limit_conn`;
- 在 `stream` 上下文中设置*limit_conn_zone*。

示例:

```
stream {
```

```
    # ...
    limit_conn_zone $connection zone=limit-conn-stream:10m;

    server {

        # ...
        limit_conn limit-conn-stream 1;
        status_zone foo;
    }
}
```

显示 `stream` 上下文的共享内存区域统计信息, 从*/status/stream/server_zones/* API 部分生成:

Conn total	客户端连接总数
Conn current	当前客户端连接数
Conn/s	每秒处理的连接数

TCP/UDP Upstreams 小部件

警告

需要在 `stream` 上下文的*upstream* 块中设置*zone* 指令。

显示 `stream` 上下文的上游统计信息, 从*/status/stream/upstreams/* API 部分生成:

Total	上游总数
Problems	存在任何问题的上游数量
Servers	按状态细分的服务器统计信息

HTTP Zones 选项卡

警告

需要在 `server` 或 `location` 上下文中设置*status_zone* 指令。

Server Zones 部分

Server Zones

Zone	Requests			Responses						Traffic				SSL			
	Current	Total	Req/s	1xx	2xx	3xx	4xx	5xx	Total	Sent/s	Rcvd/s	Sent	Rcvd	Handshaked	Reuses	Timed out	Failed
Brazil	3	166248	5	0	97821	180	361	67883	166233	127 KiB	460 B	3.02 GiB	12.3 MiB	88627	88087	0	0
Russia	0	164917	2	0	98112	189	353	66263	164906	33.0 KiB	226 B	3.04 GiB	11.8 MiB	91498	90940	0	0
India	2	134435	3	0	80026	154	303	53950	134421	121 KiB	228 B	2.48 GiB	9.70 MiB	71844	71406	0	0
China	5	78305	0	0	45324	92	200	32684	78263	51.3 KiB	170 B	1.40 GiB	5.62 MiB	22108	21853	0	0
South Africa	1	128440	2	0	76669	115	259	51396	128433	107 KiB	253 B	2.36 GiB	10.2 MiB	62442	62061	0	0
Argentina	1	160233	5	0	95301	204	312	64415	160222	93.0 KiB	421 B	2.95 GiB	12.4 MiB	79974	79485	0	0
Egypt	0	160577	2	0	95809	199	339	64230	160562	19.0 KiB	277 B	2.95 GiB	11.4 MiB	76324	75859	0	0
Ethiopia	0	165203	2	0	98405	174	412	66212	165190	66.6 KiB	238 B	3.05 GiB	12.5 MiB	71864	71426	0	0
Iran	2	165417	4	0	98752	191	352	66120	165401	56.7 KiB	274 B	3.05 GiB	11.5 MiB	96761	96170	0	0
Saudi Arabia	1	152814	4	0	90843	167	317	61486	152800	113 KiB	424 B	2.81 GiB	11.8 MiB	71658	71220	0	0
UAE	2	189884	6	0	113091	220	412	76159	189870	83.6 KiB	434 B	3.49 GiB	13.1 MiB	105751	105106	0	0

汇总 `http` 中 `server` 上下文的共享内存区域监控统计信息, 从*/status/http/server_zones/* API 部分生成。为每个区域显示以下数据:

Zone	区域名称 **小技巧** 点击 *Zone* 旁边的箭头, 可按字母顺序或配置顺序对区域进行排序。
Requests	请求总数和每秒请求数
Responses	按状态码细分的响应数量及其总数
Traffic	出站和入站流量速率, 以及出站和入站流量的总量
SSL	汇总计数: 成功的 SSL 握手;SSL 会话重用; 超时过期的 SSL 握手; 不成功的 SSL 握手

Location Zones 部分

Location Zones

Zone	Requests		Responses						Traffic			
	Total	Req/s	1xx	2xx	3xx	4xx	5xx	Total	Sent/s	Rcvd/s	Sent	Rcvd
Brasilia	33832	0	0	19872	41	74	13845	33831	0	0	625 MiB	2.48 MiB
Diadema	33320	5	0	19694	36	76	13514	33316	125 KiB	413 B	623 MiB	2.41 MiB
Porto Alegre	32620	1	0	19121	33	62	13404	32618	4.82 KiB	88.0 B	609 MiB	2.52 MiB
Salvador	33142	1	0	19370	40	60	13672	33139	33.5 KiB	83.0 B	608 MiB	2.43 MiB
Vitoria	33636	1	0	19940	30	89	13577	33634	667 B	82.0 B	628 MiB	2.43 MiB
Lipetsk	33545	0	0	20014	43	78	13410	33544	0	0	638 MiB	2.43 MiB
Moscow	33005	1	0	19665	36	69	13235	33004	334 B	81.0 B	623 MiB	2.36 MiB
Omsk	32186	1	0	19188	26	79	12893	32183	26.4 KiB	79.0 B	609 MiB	2.24 MiB
Sevastopol	33099	1	0	19659	39	65	13336	33095	334 B	84.0 B	616 MiB	2.49 MiB
Ufa	33210	0	0	19659	45	62	13444	33208	0	0	625 MiB	2.28 MiB
Bengaluru	27224	0	0	16091	34	67	11032	27222	0	0	509 MiB	2.00 MiB
Hyderabad	26978	2	0	16090	33	61	10794	26974	668 B	167 B	509 MiB	1.98 MiB
Kolkata	26412	0	0	15809	25	65	10513	26409	0	0	501 MiB	1.89 MiB
Mumbai	26854	0	0	16006	30	58	10760	26853	0	0	505 MiB	1.89 MiB
Vadodara	27141	0	0	16135	32	53	10921	27139	0	0	515 MiB	1.96 MiB

汇总 `http` 中 `location` 上下文的共享内存区域监控统计信息, 从*/status/http/location_zones/* API 部分生成。为每个区域显示以下数据:

Zone	区域名称 **小技巧** 点击 *Zone* 旁边的箭头, 可按字母顺序或配置顺序对区域进行排序。
Requests	请求总数和每秒请求数
Responses	按状态码细分的响应数量及其总数
Traffic	出站和入站流量速率, 以及出站和入站流量的总量

Connection Limit Zones (Limit Conn) 部分

显示 `http` 上下文中 `limit_conn` 区域的统计信息, 从*/status/http/limit_conns/* API 部分生成。为每个区域显示以下数据:

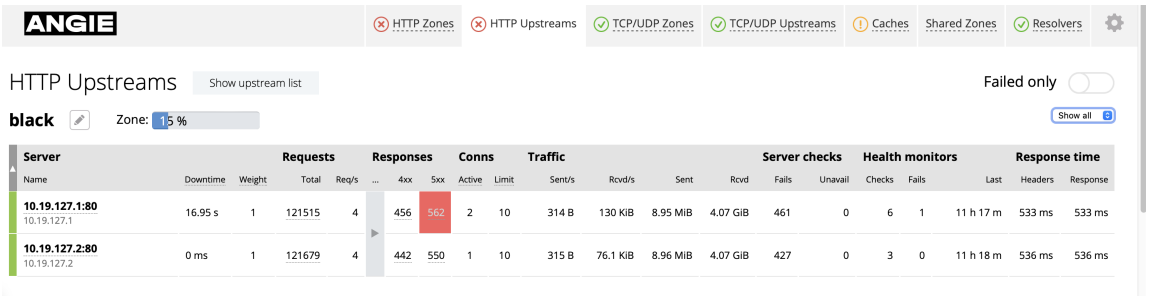
Zone	区域名称 **小技巧** 点击 *Zone* 旁边的图标, 可打开或关闭包含以下指标的图表。
Passed	已代理连接总数
Rejected	已拒绝连接总数
Exhausted	由于区域存储溢出而丢弃的连接总数
Skipped	使用零字节或大于 255 字节密钥通过的连接总数

Request Limit Zones (Limit Req) 部分

显示 `http` 上下文中 `limit_reqs` 区域的统计信息, 从*/status/http/limit_reqs/* API 部分生成。为每个区域显示以下数据:

Zone	区域名称 **小技巧** 点击 *Zone* 旁边的图标，可打开或关闭包含以下指标的图表。
Passed	已代理连接总数
Delayed	已延迟连接总数
Rejected	已拒绝连接总数
Exhausted	由于区域存储溢出而丢弃的连接总数
Skipped	使用零字节或大于 255 字节密钥通过的连接总数

HTTP 上游选项卡



警告

需要在 `http` 上下文的*upstream* 块中设置*zone* 指令。

此选项卡汇总 `http` 上下文的上游监控统计信息，从*/status/http/upstreams/* API 部分生成。在调试模式下，还会显示内存使用百分比。

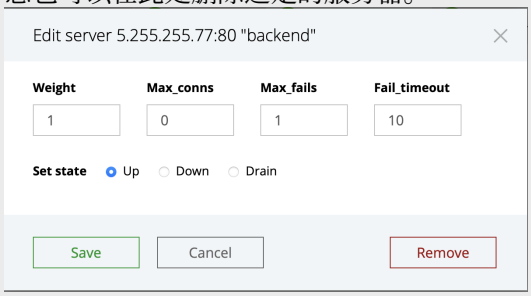
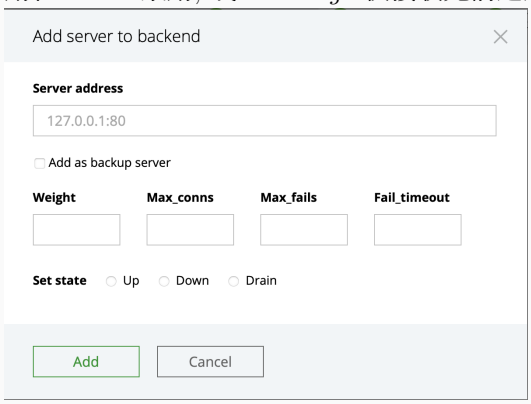
- *Show upstreams list* 按钮可切换显示包含问题上游和对等节点数量的简要列表。
- *Failed only* 开关可切换问题上游统计信息的显示模式。
- 编辑按钮可切换 *upstream editing* (上游编辑) 界面。
- 每个上游表右侧的下拉列表允许您按特定状态 (*Up*、*Failed*、*Checking*、*Down*) 筛选服务器。

对于每个上游，除了其名称和共享内存区域利用率之外，还显示以下数据:

Server	上游服务器的名称、停机时间和权重 **小技巧** 点击 *Server* 旁边的箭头, 可按服务器状态或配置顺序对服务器进行排序。
Requests	请求总数和处理速率
Responses	按状态码细分的响应数量
Connections	活动连接数及其最大限制 (如果已设置)
Traffic	出站和入站流量速率, 以及出站和入站流量的总量
Server checks	联系服务器失败的尝试次数以及服务器被视为不可用的次数 (API 中的 `health` 对象)
Health monitors	服务器检查总数、不成功检查的数量以及最后一次检查的时间
Response time	从请求开始到发送响应的第一个字节的时间; 从请求开始到完成发送整个响应的总时间 (API 中的 `health` 对象)

编辑上游

在 Angie PRO 中, 每个上游旁边都有一个编辑按钮; 点击后, 会显示另外两个按钮:

Edit selected	编辑上游中选定的服务器。允许您一次性为所有服务器设置以下参数: *Weight* (权重)、最大连接数限制 (*Max_conns*)、标记服务器为不可用的最大失败次数限制 (*Max_fails*)、计算最大失败次数限制的失败时间窗口 (*Fail_timeout*)、状态 (*active* –启用、 *down* –禁用, 或 *draining* –仅接收先前通过 *sticky* 绑定的会话请求)。 您也可以在此处删除选定的服务器。 
Add server	向上游添加服务器。允许您设置以下参数: 地址、是否为备份服务器、*Weight* (权重)、最大连接数限制 (*Max_conns*)、标记服务器为不可用的最大失败次数限制 (*Max_fails*)、失败计数时间窗口 (*Fail_timeout*)、状态 (*active* –启用、 *down* –禁用, 或 *draining* –仅接收先前通过 *sticky* 绑定的会话请求)。 

TCP/UDP Zones 选项卡

警告

需要设置以下指令:

- 在*server* 或*stream* 上下文中设置 `status_zone`;
- 在*server* 或*stream* 上下文中设置 `limit_conn`;
- 在 `stream` 上下文中设置*limit_conn_zone*。

示例:

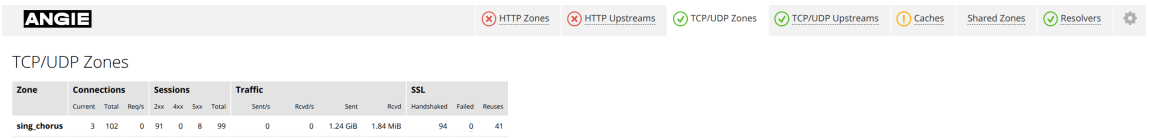
```
stream {

    # ...
    limit_conn_zone $connection zone=limit-conn-stream:10m;

    server {

        # ...
        limit_conn limit-conn-stream 1;
        status_zone foo;
    }
}
```

TCP/UDP Zones 部分



汇总 `stream` 中 `server` 上下文的共享内存区域监控统计信息, 由*/status/stream/server_zones/* API 部分生成。每个区域显示以下数据:

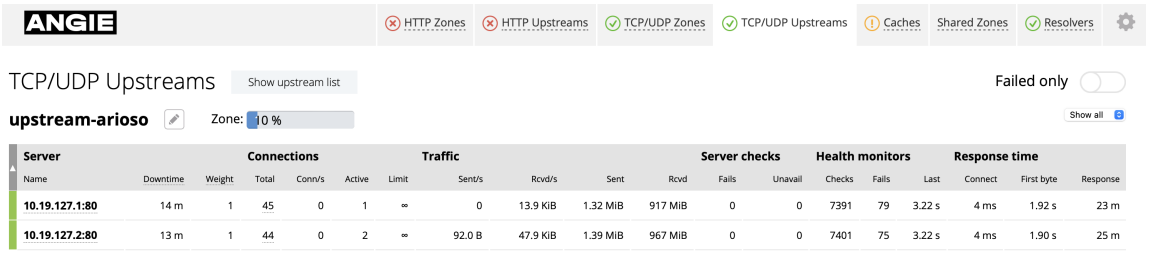
Zone	区域名称
Connections	当前和总连接数, 以及每秒连接数
Sessions	按状态码细分的会话数, 以及它们的总数
Traffic	出站和入站流量速率, 以及出站和入站流量的总量
SSL	汇总计数: 成功的 SSL 握手; 失败的 SSL 握手; SSL 会话重用

Connection Limit Zones (Limit Conn) 部分

显示 `stream` 上下文中 `limit_conn` 区域的统计信息, 由*/status/stream/limit_conns/* API 部分生成。每个区域显示以下数据:

Zone	区域名称 **小技巧** 点击 *Zone* 旁边的图标以打开或关闭包含以下指标的图表。
Passed	代理连接的总数
Rejected	拒绝连接的总数
Exhausted	由于区域存储溢出而丢弃的连接总数
Skipped	使用零字节或大于 255 字节密钥通过的连接总数

TCP/UDP Upstreams 选项卡



警告

需要在 `stream` 上下文的*upstream* 块中设置*zone* 指令。

此选项卡汇总 `stream` 上下文的上游监控统计信息, 由*/status/stream/upstreams/* API 部分生成。在调试模式下, 还会显示内存使用百分比。

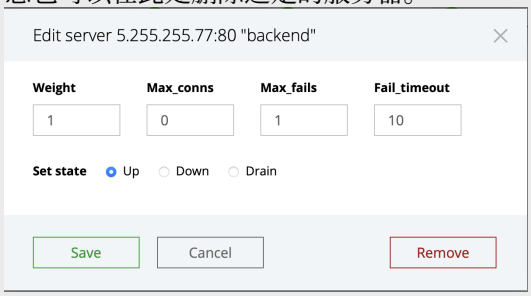
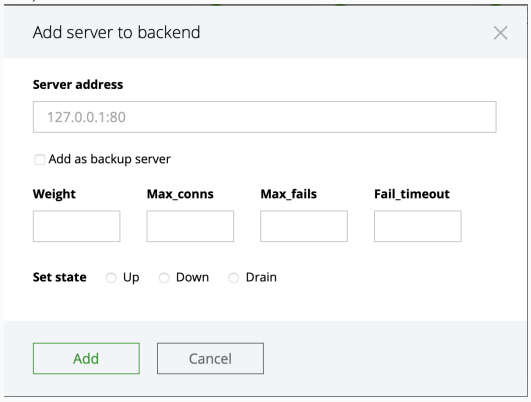
- *Show upstreams list* 按钮切换显示包含问题上游和对等点数量的简要上游列表。
- *Failed only* 开关启用和禁用问题上游统计信息的显示模式。
- 编辑按钮打开 *upstream editing* (上游编辑) 小部件。
- 每个上游表右侧的下拉列表允许您筛选特定状态的服务器 (*Up*、*Failed*、*Checking*、*Down*)。

对于每个上游, 显示以下数据:

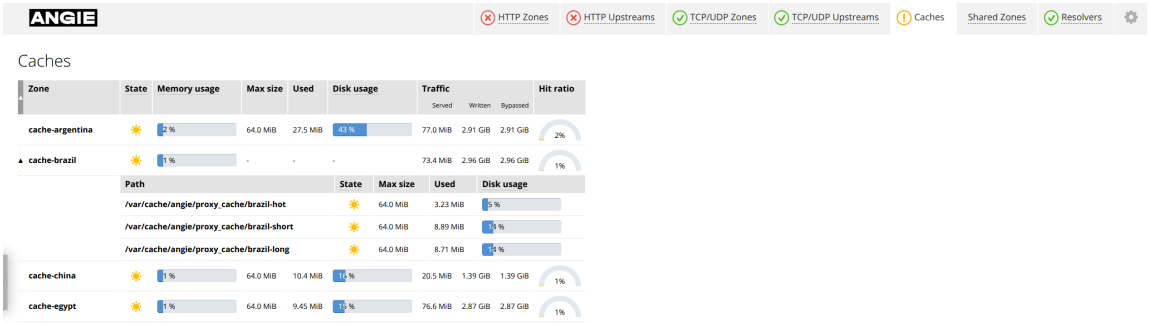
Server	上游服务器的名称、停机时间和权重 **小技巧** 点击 *Server* 旁边的箭头以按服务器状态或配置顺序对服务器进行排序。
Connections	活动连接数及其最大限制 (如果已设置)
Traffic	出站和入站流量速率, 以及出站和入站流量的总量
Server checks	联系服务器失败的尝试次数以及服务器被视为不可用的次数（API 中的 `health` 对象)
Health monitors	服务器检查的总次数、失败检查的次数以及最后一次检查的时间
Response time	建立与后端连接所花费的时间; 从请求开始到接收响应的第一个字节的时间; 从请求开始到接收响应的最后一个字节所经过的总时间（API 中的 `health` 对象)

编辑上游

在 Angie PRO 中, 每个上游旁边都有一个编辑按钮; 点击后, 会显示另外两个按钮:

`Edit selected`	编辑上游中选定的服务器。允许您一次性为所有服务器设置以下参数: `Weight` (权重)、最大连接数限制 (`Max_conns`)、将服务器标记为不可用的最大失败次数限制 (`Max_fails`)、计算最大失败次数限制的失败时间窗口 (`Fail_timeout`)、状态 (`active` –启用、`down` –禁用，或 `draining` –仅接收先前通过 `sticky` 绑定的会话请求)。 您也可以在此处删除选定的服务器。 
`Add server`	向上游添加服务器。允许您设置以下参数: 地址、是否为备份服务器、`Weight` (权重)、最大连接数限制 (`Max_conns`)、将服务器标记为不可用的最大失败次数限制 (`Max_fails`)、失败计数时间窗口 (`Fail_timeout`)、状态 (`active` –启用、`down` –禁用，或 `draining` –仅接收先前通过 `sticky` 绑定的会话请求)。 

`Caches` 选项卡



警告

需要在 `http` 上下文中设置*proxy_cache_path* 指令。

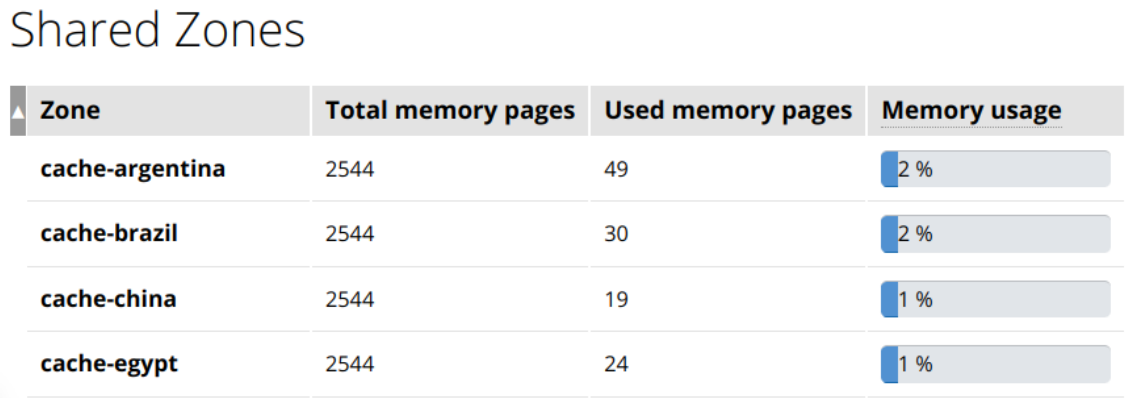
此选项卡汇总了 `http` 上下文中 `proxy_cache` 区域的监控统计信息，这些信息从*/status/http/caches/* API 部分生成。每个区域显示以下数据:

`Zone`	区域名称 **小技巧** 点击 `Zone` 旁边的图标可以打开或关闭所有具有分片 的区域的分片列表。
`State`	缓存状态:cold(元数据正在加载到内存中) 或 hot(元数据已加载)
`Memory usage`	内存利用率
`Max size`	最大内存大小
`Used`	已使用内存大小
`Disk usage`	磁盘利用率
`Traffic`	从缓存提供的流量、写入缓存的流量以及绕过缓存返回的流量
`Hit ratio`	缓存命中率 (从缓存提供的流量与总流量的比率)

如果为某个区域启用了分片, 它将显示为一个下拉列表, 列出各个分片:

`Path`	分片在磁盘上的路径
`State`	分片状态:cold(元数据正在加载到内存中) 或 hot(元数据已加载)
`Max size`	最大内存大小
`Used`	已使用内存大小
`Disk usage`	磁盘利用率

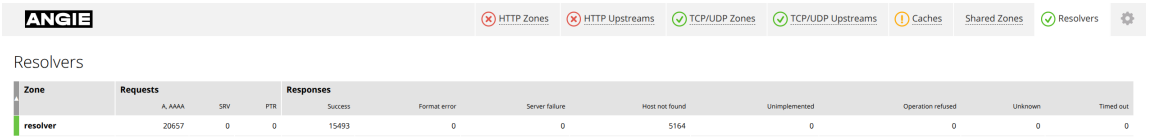
`Shared Zones` 选项卡



此选项卡汇总了 ** 所有 ** 上下文中所有共享内存区域的监控统计信息。每个区域显示以下数据:

Zone	区域名称 **小技巧** 点击 Zone 旁边的箭头可以按大小或配置顺序对区域进行排序。
Total memory pages	内存页总数
Used memory pages	已使用的内存页数
Memory usage	区域的内存利用率

DNS Resolvers 选项卡



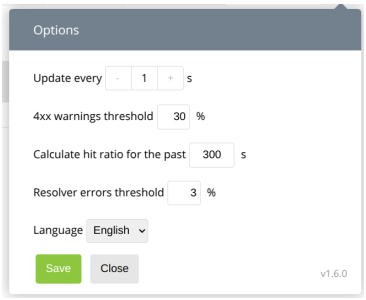
警告

需要在 http 上下文中设置*resolver* 指令。

此选项卡汇总了 DNS 共享内存区域中的查询统计信息, 这些信息从*/status/resolvers/* API 部分生成。每个区域显示以下数据:

Zone	区域名称 **小技巧** 点击 Zone 旁边的箭头可以按状态或配置顺序对区域进行排序。
Requests	A 和 AAAA、SRV、PTR 类型请求的数量
Responses	按相应代码细分的响应数量 (Success、Format error、Server failure、Name error、Not implemented、Refused 及其他)

Settings 小部件



允许您配置控制台的常规参数:

- 数据刷新率。默认值–1 秒。
- 4xx 状态的阈值比率。当达到阈值时, 与服务器响应相关的相应部分将出现" 黄色" 警告。默认值–7%。
- 计算缓存命中率的时间窗口。默认值–300 秒。
- 解析器的错误阈值。当达到阈值时, 解析器将变为" 红色"。默认值–3%。
- 控制台界面语言。可用选项: 英语和俄语。默认情况下, 控制台语言根据浏览器中设置的区域设置选择。

控制台控制面板

在所有选项卡上, 页面左侧中间位置有一个滑出面板, 带有两个按钮 。顶部按钮用于暂停和恢复从 API 更新数据, 底部按钮允许您在更新暂停时手动更新数据。

3.3.6 自定义指标配置

Angie 可以在共享内存中收集自定义数值指标，并通过实时统计 *API* 在 `/status/http/metric_zones/` 公开它们。这由*Metric* 模块提供。

3.3.6.1 配置步骤

1. 在 `http` 块中定义指标区域:
   - *metric_zone* 创建具有单个指标模式的区域。
   - *metric_complex_zone* 创建具有多个命名指标的区域。
2. 使用*metric* 指令在请求处理中更新指标。使用 `key=value` 对（两者都是复杂值），并使用 `on=` 选择更新阶段（`request`、`response` 或 `end`）。
3. 使用 `location` 公开 API:

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

3.3.6.2 示例

统计每个主机的请求数并在 API 中公开指标:

```
http {
    metric_zone requests:128k count;

    server {
        listen 80;
```

```
        location / {
            metric requests $host=1;
        }

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

3.3.6.3 注意事项

- 如果在区域上设置了 `expire=on` 且共享内存已满，则最近最少使用的条目将过期。如果设置了 `expire=off`，新的更新将被丢弃，`discarded` 计数器会增长。
- 如果设置了 `discard_key`，过期条目的指标将在 API 输出中聚合到该键下。
- 键和值限制为 255 字节；较长的键在 API 中会被截断。
- 空值被视为 0，没有前导数字的非空值被视为 1。

3.3.7 从 nginx 迁移到 Angie

如果您正在从 nginx 切换到 Angie, 恭喜您! 我们为您准备了一份指南。

请注意, 本指南针对的是依赖 Angie 软件包版本的基本替换场景。如果您使用的是 容器、虚拟机、自定义路径或 模块, 您将需要进行额外的调整。

3.3.7.1 安装 Angie

我们建议使用来自 我们仓库的官方软件包; 请参阅适用于您的发行版的 Angie 安装步骤。暂时不要启动服务器; 而是使用命令 **`sudo angie -V`** 检查它:

```
$ sudo angie -V

  Angie version: Angie/1.12.0
  nginx version: nginx/1.31.2
  built by gcc 11.4.0
  configure arguments: --prefix=/etc/angie --conf-path=/etc/angie/angie.conf ...
```

如上所示, 当从软件包安装 Angie 时, 配置 位于 `/etc/angie/` 中。

3.3.7.2 更新 Angie 配置

Angie 通常只需要对现有 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 配置存储在 `/etc/nginx/` 中; 如果您的路径不同, 请调整这些步骤。

2. 按照 Angie 的要求重命名主配置文件:

```
$ 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。具体细节取决于 nginx 的安装方式, 但至少需要更新以下内容。

   任何仍然指向 `/etc/nginx/` 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;
```

   PID 文件, 这对 Angie 进程管理很重要:

```
# 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 sites-enabled/ 目录来包含虚拟主机, 也需要更新它:

```
# include /etc/nginx/sites-enabled/*;
include /etc/angie/sites-enabled/*;
```

然后在 /etc/angie/sites-enabled/ 中重新创建符号链接以使一切正常工作。

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; 这里是 /etc/nginx/sites-available/。

如果之前没有将它们复制到 /etc/angie/, 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

为 nginx 配置中引用的所有动态模块查找并 安装 Angie 等效模块, 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 包, 依此类推。

有两种常见的方式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:

```
# Load dynamic modules. See /usr/share/doc/nginx/README.dynamic.
# include /usr/share/nginx/modules/*.conf;

include /usr/share/angie/modules/*.conf;
```

然后复制模块配置文件:

```
$ sudo rsync -a /usr/share/nginx/modules/ /usr/share/angie/modules/
```

最后, 在每个文件中更改*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/`

如果通过 `/etc/nginx/modules-enabled/` 包含动态模块, 更新路径:

```
# include /etc/nginx/modules-enabled/*.conf;
include /etc/angie/modules-enabled/*.conf;
```

然后在 `/etc/angie/modules-enabled/` 中重新创建符号链接以使一切正常工作。

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; 这里是 `/usr/share/nginx/modules-available/`。

将它们复制到 `/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, 您可以更改该指令以指向 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 配置中有*root* 指令, 更改其中指定目录的所有者, 例如:

```
$ sudo chown -R angie:angie /var/www/html/
```

收尾工作

为确保没有遗漏, 在 Angie 配置中查找并修复剩余的 `nginx` 引用:

```
$ grep -rn --include='*.conf' 'nginx' /etc/angie/
```

3.3.7.3 测试和切换

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该命令。

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 服务以在重启后自动启动:

```
$ sudo systemctl enable angie
```

迁移完成! 就是这样; 您真棒。

禁用 nginx

在确认 Angie 稳定运行后, 您可以禁用或删除 nginx 以避免冲突。

您至少可以禁用该服务:

```
$ sudo systemctl disable nginx
```

3.3.7.4 使用 SSL 证书

如果您使用 Certbot 管理 nginx 的 SSL 证书, 它将继续与 Angie 一起工作。

在 Angie 中使用 Certbot

在 Certbot 中支持 Angie 只需最少的工作, 因为 Angie 向后兼容 nginx。要使 Certbot 工作, 只需创建一个符号链接并指定适当的参数:

```
# 创建符号链接以实现 Certbot 兼容性
$ sudo ln -s /etc/angie/angie.conf /etc/angie/nginx.conf

# 为域名获取证书
$ sudo certbot --nginx --nginx-server-root=/etc/angie --nginx-ctl=angie -d example.com -d www.
↪example.com

# 自动续订证书
$ sudo certbot renew

# 检查证书状态
$ sudo certbot certificates
```

迁移到 Angie 后,Certbot 将继续通过配置的 `cron` 作业或 `systemd` 定时器自动续订证书。

从 Certbot 迁移到内置 ACME 模块

Angie 包含内置的*ACME 模块*, 它允许您自动获取和续订 SSL 证书, 而无需使用 Certbot 等外部工具。

内置 ACME 模块的优势:

- 与 Angie 配置完全集成;
- 无需额外服务即可自动续订证书;
- 支持 HTTP 和 DNS 验证;
- 能够获取通配符证书。

有关从 Certbot 迁移到内置 ACME 模块的详细说明, 请参阅从 *certbot* 迁移 部分。

3.3.7.5 适配行为不同的指令

Angie 中有少数 nginx 指令的行为有所不同: 部分已重命名, 部分已弃用, 还有少数已被完全删除。如果您的配置中使用了其中任何指令, 请参阅*nginx 不支持的指令*。

特别需要注意的是,nginx 的 `keepalive_min_timeout` 在 Angie 中命名为*lingering_timeout*; 如果您的配置中出现该指令, 请将其重命名。

3.3.7.6 配置 Angie 功能

可以肯定地说, 您迁移是有原因的。为什么不更进一步, 配置一些 Angie 和 Angie PRO 中可用的附加功能, 这些功能在 nginx 中是没有的?

3.3.8 nginx 不支持的指令

Angie 中的大多数 nginx 指令无需修改即可使用, 因此现有配置通常不需要调整。本页列出了例外情况——Angie 删除、重命名或弃用的指令, 以便您在从 nginx 迁移时调整配置。

nginx 本身未记录文档的底层调优指令 (例如事件方法和 gzip 调优参数) 在 Angie 中仍然有效, 使用 nginx 的默认值, 此处有意省略。

备注

有关端到端迁移流程, 请参阅迁移指南。

3.3.8.1 已删除或省略

以下 nginx 指令在 Angie 中没有对应的等效指令。

nginx 指令	说明
`add_header_inherit`, `add_trailer_inherit`	因设计缺陷而省略; 请参阅 oss_changes。

3.3.8.2 已重命名

nginx 指令	Angie 指令
`keepalive_min_timeout`	*lingering_timeout*

3.3.8.3 已弃用

以下指令仍然有效, 但会发出警告; 请改用现代指令。

nginx 指令	改用
`http2_idle_timeout`	*keepalive_timeout*
`http2_max_requests`	*keepalive_requests*
`http2_recv_timeout`	*client_header_timeout*
`http2_max_field_size`	*large_client_header_buffers*
`http2_max_header_size`	*large_client_header_buffers*
`proxy_downstream_buffer` (stream)	*proxy_buffer_size*
`proxy_upstream_buffer` (stream)	*proxy_buffer_size*

3.3.9 配置 Grafana 仪表板

要在 Grafana 中配置 Angie 的仪表板，请按照以下步骤操作：

1. 使用*Prometheus* 模块，添加以下*include* 指令到配置文件 的 `http` 块中：

```
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 的仪表板 导入到 Grafana 中。

3.4 社区材料

我们收集了社区资源, 这些资源将帮助您更好地理解 Angie 的配置和使用。

3.4.1 文章

- Angie: A New NGINX Fork Developed by Some of Its Former Devs on Linuxiac
- What's New in the Angie 1.9 Web Server (an nginx fork) and What to Expect from 1.10? on Habr

3.4.2 课程

目前没有英文的 Angie 课程。请参阅官方 Angie 文档和下面的实用指南。

3.4.3 实用指南

- Migrating from Nginx to Angie: A Real-World Journey from Certbot to Built-in ACME on DEV Community

3.4.4 访谈和播客

- "NGINX is Dead? // Angie Web Server Migration Guide" by DevOps Toolbox (YouTube, 27.03.2026)
- "Nginx Has a BIG Problem..." by DevOps Toolbox (YouTube, 30.01.2026)

3.5 面向 AI 助手的文档

本站为每个文档页面提供机器可读的副本, 以便 Claude Code、Cursor、ChatGPT 等基于 LLM 的代理工具可以直接获取内容, 而无需抓取渲染后的 HTML。

llms.txt 与 llms-full.txt

每个语言子域名都提供一份 llms.txt 索引文件, 其中包含项目的简短描述以及全部页面的列表 (标题、绝对 URL、摘要)。配套的 `llms-full.txt` 将所有页面的完整 Markdown 内容拼接为一个文件, 适合一次性加载到模型中:

- https://cn.angie.software/llms.txt
- https://cn.angie.software/llms-full.txt

上述 URL 也通过 `Llms:` 指令在 robots.txt 中声明, LLM 爬虫可以自动发现它们。

页面的 Markdown 版本

每个 HTML 页面都有对应的 Markdown 版本。将文档页面 URL 末尾的斜杠替换为 `.md` 即可获取其 Markdown 版本:

- HTML: https://cn.angie.software/angie/docs/configuration/
- Markdown: https://cn.angie.software/angie/docs/configuration.md

Markdown 版本与 HTML 版本由相同的 reStructuredText 源文件构建, 因此内容始终保持同步。

Context7

Angie 文档已收录于 Context7 ——一个通过 MCP 服务器为 AI 代码编辑器提供最新库文档的目录。中文版 Angie 在 Context7 的页面位于 https://context7.com/websites/cn_angie_software_angie。

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技术支持建议进行操作之前，应启用调试日志。

为此，请使用支持调试的可执行文件运行 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
```

Docker

在 模板化 Docker 镜像中，您可以通过覆盖 `ANGIE_BINARY` 环境变量切换到调试版本：

```
$ docker run -it --rm -e ANGIE_BINARY="angie-debug" \
  docker.angie.software/angie:templated
```

从源构建

当从源构建 Angie 时，编译前 启用调试：

```
$ ./configure --with-debug ...
```

安装后，`angie -V` 可以验证是否启用了调试日志：

```
$ angie -V

  ...
  configure arguments: --with-debug ...
```

备注

使用支持调试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
```

在启用调试日志的 模板化 Docker 镜像中，您也可以使用 ANGIE_ERROR_LOG_SEVERITY 环境变量：

```
$ docker run -it --rm -e ANGIE_BINARY="angie-debug" \
-e ANGIE_ERROR_LOG_SEVERITY="debug" \
docker.angie.software/angie:templated
```

如果切换到不支持调试的可执行文件但在*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
   --- 2026-07-13 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
```

章节 5

知识产权

Angie PRO 软件产品的文档是 Web Server, LLC 的知识产权。本文档是对 nginx 软件产品文档进行修改 (修订) 的结果。

索引

A

absolute_redirect (*http*), 377
accept_mutex (*core*), 22
accept_mutex_delay (*core*), 22
access_log (*http*), 155
access_log (*stream*), 435
acme (*http*), 34
acme (*stream*), 428
acme_client (*http*), 34
acme_client_path (*http*), 37
acme_dns_port (*http*), 37
acme_dns_ttl (*http*), 37
acme_hook (*http*), 37
acme_http_port (*http*), 38
acme_max_response_size (*http*), 38
add_after_body (*http*), 40
add_before_body (*http*), 40
add_header (*http*), 144
add_trailer (*http*), 144
addition_types (*http*), 40
aio (*http*), 377
aio_write (*http*), 378
alias (*http*), 378
allow (*http*), 32
allow (*stream*), 426
ancient_browser (*http*), 88
ancient_browser_value (*http*), 88
api (*http*), 41
api_config_files (*http*), 43
auth_basic (*http*), 82
auth_basic_user_file (*http*), 83
auth_delay (*http*), 379
auth_http, 529
auth_http_header, 530
auth_http_pass_client_cert, 530
auth_http_timeout, 530
auth_request (*http*), 84
auth_request_set (*http*), 84
auto_redirect (*http*), 379
autoindex (*http*), 85
autoindex_exact_size (*http*), 85
autoindex_format (*http*), 85
autoindex_localtime (*http*), 87

B

backup_switch (*http*), 318
backup_switch (*stream*), 497
bind_conn (*http*), 318
break (*http*), 259

C

certificates (*API*), 72
charset (*http*), 90
charset_map (*http*), 90
charset_types (*http*), 91
chunked_transfer_encoding (*http*), 380
client (*http*), 380
client_body_buffer_size (*http*), 381
client_body_in_file_only (*http*), 382
client_body_in_single_buffer (*http*), 382
client_body_temp_path (*http*), 382
client_body_timeout (*http*), 383
client_header_buffer_size (*http*), 383
client_header_timeout (*http*), 383
client_max_body_size (*http*), 383
connection_pool_size (*http*), 384
create_full_put_path (*http*), 92

D

daemon (*core*), 22
dav_access (*http*), 93
dav_methods (*http*), 93
debug_connection (*core*), 22
debug_points (*core*), 23
default_type (*http*), 384
deny (*http*), 33
deny (*stream*), 426
directio (*http*), 384
directio_alignment (*http*), 384
disable_symlinks (*http*), 385
docker_endpoint (*http*), 97
docker_max_object_size (*http*), 98
doh_bind (*http*), 100
doh_buffer_size (*http*), 100
doh_connect_timeout (*http*), 100
doh_max_size (*http*), 100
doh_next_upstream (*http*), 100
doh_next_upstream_timeout (*http*), 101
doh_next_upstream_tries (*http*), 101
doh_pass (*http*), 101
doh_read_timeout (*http*), 102
doh_send_timeout (*http*), 102
doh_socket_keepalive (*http*), 102
doh_transport (*http*), 102

E

early_hints (*http*), 386
empty_gif (*http*), 103
env (*core*), 23
error_log (*core*), 24
error_log_user_tag (*http*), 386
error_page (*http*), 386
etag (*http*), 387
events (*core*), 25
expires (*http*), 144

F

fastcgi_bind (*http*), 104
fastcgi_buffer_size (*http*), 104
fastcgi_buffering (*http*), 104
fastcgi_buffers (*http*), 105
fastcgi_busy_buffers_size (*http*), 105
fastcgi_cache (*http*), 105
fastcgi_cache_background_update (*http*), 106
fastcgi_cache_bypass (*http*), 106
fastcgi_cache_key (*http*), 106
fastcgi_cache_lock (*http*), 107
fastcgi_cache_lock_age (*http*), 107
fastcgi_cache_lock_timeout (*http*), 107
fastcgi_cache_max_range_offset (*http*), 107
fastcgi_cache_methods (*http*), 108
fastcgi_cache_min_uses (*http*), 108
fastcgi_cache_path (*http*), 108
fastcgi_cache_revalidate (*http*), 111
fastcgi_cache_use_stale (*http*), 111
fastcgi_cache_valid (*http*), 111
fastcgi_catch_stderr (*http*), 112
fastcgi_connect_timeout (*http*), 113
fastcgi_connection_drop (*http*), 113
fastcgi_force_ranges (*http*), 113
fastcgi_hide_header (*http*), 113
fastcgi_ignore_client_abort (*http*), 114
fastcgi_ignore_headers (*http*), 114
fastcgi_index (*http*), 114
fastcgi_intercept_errors (*http*), 115
fastcgi_keep_conn (*http*), 115
fastcgi_limit_rate (*http*), 115
fastcgi_max_temp_file_size (*http*), 115
fastcgi_next_upstream (*http*), 116
fastcgi_next_upstream_timeout (*http*), 117
fastcgi_next_upstream_tries (*http*), 117
fastcgi_no_cache (*http*), 117
fastcgi_param (*http*), 118
fastcgi_pass (*http*), 119
fastcgi_pass_header (*http*), 119
fastcgi_pass_request_body (*http*), 119
fastcgi_pass_request_headers (*http*), 120
fastcgi_read_timeout (*http*), 120
fastcgi_request_buffering (*http*), 120
fastcgi_send_lowat (*http*), 120
fastcgi_send_timeout (*http*), 121
fastcgi_socket_keepalive (*http*), 121
fastcgi_split_path_info (*http*), 121
fastcgi_store (*http*), 122
fastcgi_store_access (*http*), 123
fastcgi_temp_file_write_size (*http*), 123
fastcgi_temp_path (*http*), 123
feedback (*http*), 319

feedback (*stream*), 497
flv (*http*), 125

G

geo (*http*), 125
geo (*stream*), 430
geoip_city (*http*), 127
geoip_city (*stream*), 432
geoip_country (*http*), 128
geoip_country (*stream*), 432
geoip_org (*http*), 128
geoip_org (*stream*), 432
geoip_proxy (*http*), 128
geoip_proxy_recursive (*http*), 129
google_perftools_profiles, 553
goto (*http*), 260
grpc_bind (*http*), 129
grpc_buffer_size (*http*), 130
grpc_connect_timeout (*http*), 130
grpc_connection_drop (*http*), 130
grpc_hide_header (*http*), 131
grpc_ignore_headers (*http*), 131
grpc_intercept_errors (*http*), 131
grpc_next_upstream (*http*), 132
grpc_next_upstream_timeout (*http*), 133
grpc_next_upstream_tries (*http*), 133
grpc_pass (*http*), 133
grpc_pass_header (*http*), 134
grpc_read_timeout (*http*), 134
grpc_send_timeout (*http*), 134
grpc_set_header (*http*), 134
grpc_socket_keepalive (*http*), 135
grpc_ssl_certificate (*http*), 135
grpc_ssl_certificate_cache (*http*), 135
grpc_ssl_certificate_key (*http*), 136
grpc_ssl_ciphers (*http*), 136
grpc_ssl_conf_command (*http*), 137
grpc_ssl_crl (*http*), 137
grpc_ssl_name (*http*), 137
grpc_ssl_password_file (*http*), 138
grpc_ssl_protocols (*http*), 138
grpc_ssl_server_name (*http*), 138
grpc_ssl_session_reuse (*http*), 138
grpc_ssl_trusted_certificate (*http*), 138
grpc_ssl_verify (*http*), 139
grpc_ssl_verify_depth (*http*), 139
gunzip (*http*), 139
gunzip_buffers (*http*), 139
gzip (*http*), 140
gzip_buffers (*http*), 140
gzip_comp_level (*http*), 141
gzip_disable (*http*), 141
gzip_http_version (*http*), 141
gzip_min_length (*http*), 141
gzip_proxied (*http*), 141
gzip_static (*http*), 143
gzip_types (*http*), 142
gzip_vary (*http*), 142

H

hash (*http*), 321
hash (*stream*), 498
http (*http*), 388
http2 (*http*), 368
http2_body_preread_size (*http*), 368
http2_chunk_size (*http*), 368
http2_max_concurrent_pushes (*http*), 368
http2_max_concurrent_streams (*http*), 369
http2_push (*http*), 369
http2_push_preload (*http*), 369
http2_recv_buffer_size (*http*), 370
http3 (*http*), 371
http3_hq (*http*), 372
http3_max_concurrent_streams (*http*), 372
http3_max_table_capacity (*http*), 372
http3_stream_buffer_size (*http*), 372

I

if (*http*), 260
if_modified_since (*http*), 388
ignore_invalid_headers (*http*), 388
image_filter (*http*), 146
image_filter_avif_quality (*http*), 146
image_filter_buffer (*http*), 147
image_filter_heic_quality (*http*), 147
image_filter_interlace (*http*), 147
image_filter_jpeg_quality (*http*), 147
image_filter_sharpen (*http*), 147
image_filter_transparency (*http*), 148
image_filter_webp_quality (*http*), 148

imap_auth, 533
imap_capabilities, 533
imap_client_buffer, 534
include (*core*), 25
index (*http*), 148
internal (*http*), 388
ip_hash (*http*), 321

K

keepalive (*http*), 322
keepalive_disable (*http*), 389
keepalive_requests (*http*), 324, 390
keepalive_time (*http*), 324, 390
keepalive_timeout (*http*), 324, 390

L

large_client_header_buffers (*http*), 391
least_conn (*http*), 324
least_conn (*stream*), 499
least_time (*http*), 325
least_time (*stream*), 499
limit_conn (*http*), 150
limit_conn (*stream*), 433
limit_conn_dry_run (*http*), 150
limit_conn_dry_run (*stream*), 434
limit_conn_log_level (*http*), 151
limit_conn_log_level (*stream*), 434
limit_conn_status (*http*), 151
limit_conn_zone (*http*), 151
limit_conn_zone (*stream*), 434
limit_except (*http*), 391
limit_rate (*http*), 391
limit_rate_after (*http*), 392
limit_req (*http*), 152
limit_req_dry_run (*http*), 153
limit_req_log_level (*http*), 153
limit_req_status (*http*), 153
limit_req_zone (*http*), 153
lingering_close (*http*), 392
lingering_time (*http*), 393
lingering_timeout (*http*), 393
listen, 548
listen (*http*), 393
listen (*stream*), 517
load, 557
load_module (*core*), 25
location (*http*), 397
lock_file (*core*), 26
log_format (*http*), 156
log_format (*stream*), 436
log_not_found (*http*), 399
log_subrequest (*http*), 400

M

mail, 550
map (*http*), 158
map (*stream*), 437
map_hash_bucket_size (*http*), 159
map_hash_bucket_size (*stream*), 439
map_hash_max_size (*http*), 159
map_hash_max_size (*stream*), 439
master_process (*core*), 26
max_commands, 550
max_errors, 550
max_headers (*http*), 400
max_ranges (*http*), 400
memcached_bind (*http*), 160
memcached_buffer_size (*http*), 160
memcached_connect_timeout (*http*), 161
memcached_gzip_flag (*http*), 161
memcached_next_upstream (*http*), 161
memcached_next_upstream_timeout (*http*), 162
memcached_next_upstream_tries (*http*), 162
memcached_pass (*http*), 162
memcached_read_timeout (*http*), 163
memcached_send_timeout (*http*), 163
memcached_socket_keepalive (*http*), 163
merge_slashes (*http*), 400
metric (*http*), 165
metric (*stream*), 441
metric_complex_zone (*http*), 166
metric_complex_zone (*stream*), 441
metric_zone (*http*), 167
metric_zone (*stream*), 442
min_delete_depth (*http*), 93
mirror (*http*), 186
mirror_request_body (*http*), 186
modern_browser (*http*), 89
modern_browser_value (*http*), 89
mp4 (*http*), 188

mp4_buffer_size (*http*), 188
mp4_limit_rate (*http*), 188
mp4_limit_rate_after (*http*), 188
mp4_max_buffer_size (*http*), 189
mp4_start_key_frame (*http*), 189
mqtt_preread (*stream*), 461
msie_padding (*http*), 401
msie_refresh (*http*), 401
multi_accept (*core*), 26

N

no_port (*stream*), 500

O

open_file_cache (*http*), 401
open_file_cache_errors (*http*), 402
open_file_cache_events (*http*), 402
open_file_cache_min_uses (*http*), 402
open_file_cache_valid (*http*), 403
open_log_file_cache (*http*), 156
open_log_file_cache (*stream*), 436
output_buffers (*http*), 403
override_charset (*http*), 91

P

pass (*stream*), 463
pcre_jit (*core*), 26
perl (*http*), 191
perl_modules (*http*), 191
perl_require (*http*), 191
perl_set (*http*), 192
pid (*core*), 27
pop3_auth, 534
pop3_capabilities, 534
port_in_redirect (*http*), 403
postpone_output (*http*), 403
preread_buffer_size (*stream*), 520
preread_timeout (*stream*), 520
prometheus (*http*), 218
prometheus_template (*http*), 218
protocol, 550
proxy_bind (*http*), 221
proxy_bind (*stream*), 464
proxy_buffer, 535
proxy_buffer_size (*http*), 221
proxy_buffer_size (*stream*), 464
proxy_buffering (*http*), 222
proxy_buffers (*http*), 222
proxy_busy_buffers_size (*http*), 222
proxy_cache (*http*), 223
proxy_cache_background_update (*http*), 224
proxy_cache_bypass (*http*), 224
proxy_cache_convert_head (*http*), 225
proxy_cache_key (*http*), 225
proxy_cache_lock (*http*), 225
proxy_cache_lock_age (*http*), 225
proxy_cache_lock_timeout (*http*), 226
proxy_cache_max_range_offset (*http*), 226
proxy_cache_methods (*http*), 226
proxy_cache_min_uses (*http*), 226
proxy_cache_path (*http*), 227
proxy_cache_revalidate (*http*), 229
proxy_cache_use_stale (*http*), 229
proxy_cache_valid (*http*), 230
proxy_connect_timeout (*http*), 231
proxy_connect_timeout (*stream*), 464
proxy_connection_drop (*http*), 231
proxy_connection_drop (*stream*), 464
proxy_cookie_domain (*http*), 231
proxy_cookie_flags (*http*), 232
proxy_cookie_path (*http*), 232
proxy_download_rate (*stream*), 465
proxy_force_ranges (*http*), 233
proxy_half_close (*stream*), 465
proxy_headers_hash_bucket_size (*http*), 233
proxy_headers_hash_max_size (*http*), 233
proxy_hide_header (*http*), 234
proxy_http3_hq (*http*), 234
proxy_http3_max_concurrent_streams (*http*), 234
proxy_http3_max_table_capacity (*http*), 234
proxy_http3_stream_buffer_size (*http*), 235
proxy_http_version (*http*), 235
proxy_ignore_client_abort (*http*), 235
proxy_ignore_headers (*http*), 235
proxy_intercept_errors (*http*), 236
proxy_limit_rate (*http*), 236
proxy_max_temp_file_size (*http*), 237
proxy_method (*http*), 237
proxy_next_upstream (*http*), 237
proxy_next_upstream (*stream*), 466

proxy_next_upstream_timeout (*http*), 238
proxy_next_upstream_timeout (*stream*), 466
proxy_next_upstream_tries (*http*), 239
proxy_next_upstream_tries (*stream*), 466
proxy_no_cache (*http*), 239
proxy_pass (*http*), 239
proxy_pass (*stream*), 466
proxy_pass_error_message, 535
proxy_pass_header (*http*), 241
proxy_pass_request_body (*http*), 241
proxy_pass_request_headers (*http*), 242
proxy_pass_trailers (*http*), 242
proxy_protocol, 535
proxy_protocol (*stream*), 467
proxy_protocol_timeout (*stream*), 520
proxy_protocol_tlv (*stream*), 467
proxy_protocol_version (*stream*), 467
proxy_quic_active_connection_id_limit (*http*), 242
proxy_quic_gso (*http*), 243
proxy_quic_host_key (*http*), 243
proxy_read_timeout (*http*), 243
proxy_redirect (*http*), 243
proxy_request_buffering (*http*), 245
proxy_requests (*stream*), 467
proxy_responses (*stream*), 468
proxy_send_lowat (*http*), 245
proxy_send_timeout (*http*), 246
proxy_set_body (*http*), 246
proxy_set_header (*http*), 246
proxy_smtp_auth, 535
proxy_socket_keepalive (*http*), 247
proxy_socket_keepalive (*stream*), 468
proxy_ssl (*stream*), 468
proxy_ssl_certificate (*http*), 247
proxy_ssl_certificate (*stream*), 468
proxy_ssl_certificate_cache (*http*), 248
proxy_ssl_certificate_cache (*stream*), 469
proxy_ssl_certificate_key (*http*), 248
proxy_ssl_certificate_key (*stream*), 469
proxy_ssl_ciphers (*http*), 249
proxy_ssl_ciphers (*stream*), 470
proxy_ssl_conf_command (*http*), 249
proxy_ssl_conf_command (*stream*), 471
proxy_ssl_crl (*http*), 250
proxy_ssl_crl (*stream*), 471
proxy_ssl_keylog_file (*http*), 250
proxy_ssl_name (*http*), 251
proxy_ssl_name (*stream*), 471
proxy_ssl_ntls (*http*), 251
proxy_ssl_ntls (*stream*), 471
proxy_ssl_password_file (*http*), 251
proxy_ssl_password_file (*stream*), 472
proxy_ssl_protocols (*http*), 252
proxy_ssl_protocols (*stream*), 472
proxy_ssl_server_name (*http*), 252
proxy_ssl_server_name (*stream*), 472
proxy_ssl_session_reuse (*http*), 252
proxy_ssl_session_reuse (*stream*), 473
proxy_ssl_trusted_certificate (*http*), 252
proxy_ssl_trusted_certificate (*stream*), 473
proxy_ssl_verify (*http*), 252
proxy_ssl_verify (*stream*), 473
proxy_ssl_verify_depth (*http*), 253
proxy_ssl_verify_depth (*stream*), 473
proxy_store (*http*), 253
proxy_store_access (*http*), 254
proxy_temp_file_write_size (*http*), 254
proxy_temp_path (*http*), 255
proxy_timeout, 536
proxy_timeout (*stream*), 473
proxy_upload_rate (*stream*), 474

Q

queue (*http*), 325
quic_active_connection_id_limit (*http*), 373
quic_bpf (*http*), 373
quic_gso (*http*), 373
quic_host_key (*http*), 373
quic_retry (*http*), 374

R

random (*http*), 326
random (*stream*), 501
random_index (*http*), 256
rdp_preread (*stream*), 475
read_ahead (*http*), 403
real_ip_header (*http*), 256
real_ip_recursive (*http*), 257
recursive_error_pages (*http*), 404

referer_hash_bucket_size (*http*), 258
referer_hash_max_size (*http*), 258
request_pool_size (*http*), 404
reset_timedout_connection (*http*), 404
resolver, 551
resolver (*http*), 405
resolver (*stream*), 520
resolver_timeout, 552
resolver_timeout (*http*), 406
resolver_timeout (*stream*), 522
response_time_factor (*http*), 326
response_time_factor (*stream*), 501
return (*http*), 261
return (*stream*), 476
rewrite (*http*), 262
rewrite_log (*http*), 263
root (*http*), 407

S

satisfy (*http*), 407
scgi_bind (*http*), 265
scgi_buffer_size (*http*), 266
scgi_buffering (*http*), 266
scgi_buffers (*http*), 266
scgi_busy_buffers_size (*http*), 267
scgi_cache (*http*), 267
scgi_cache_background_update (*http*), 267
scgi_cache_bypass (*http*), 268
scgi_cache_key (*http*), 268
scgi_cache_lock (*http*), 268
scgi_cache_lock_age (*http*), 269
scgi_cache_lock_timeout (*http*), 269
scgi_cache_max_range_offset (*http*), 269
scgi_cache_methods (*http*), 269
scgi_cache_min_uses (*http*), 270
scgi_cache_path (*http*), 270
scgi_cache_revalidate (*http*), 272
scgi_cache_use_stale (*http*), 272
scgi_cache_valid (*http*), 273
scgi_connect_timeout (*http*), 273
scgi_connection_drop (*http*), 274
scgi_force_ranges (*http*), 274
scgi_hide_header (*http*), 274
scgi_ignore_client_abort (*http*), 274
scgi_ignore_headers (*http*), 275
scgi_intercept_errors (*http*), 275
scgi_limit_rate (*http*), 275
scgi_max_temp_file_size (*http*), 276
scgi_next_upstream (*http*), 276
scgi_next_upstream_timeout (*http*), 277
scgi_next_upstream_tries (*http*), 277
scgi_no_cache (*http*), 278
scgi_param (*http*), 278
scgi_pass (*http*), 279
scgi_pass_header (*http*), 279
scgi_pass_request_body (*http*), 279
scgi_pass_request_headers (*http*), 279
scgi_read_timeout (*http*), 280
scgi_request_buffering (*http*), 280
scgi_send_timeout (*http*), 280
scgi_socket_keepalive (*http*), 280
scgi_store (*http*), 281
scgi_store_access (*http*), 282
scgi_temp_file_write_size (*http*), 282
scgi_temp_path (*http*), 282
secure_link (*http*), 283
secure_link_md5 (*http*), 283
secure_link_secret (*http*), 284
send_lowat (*http*), 407
send_timeout (*http*), 408
sendfile (*http*), 408
sendfile_max_chunk (*http*), 408
server, 552
server (*http*), 327, 408
server (*stream*), 501, 522
server_name, 552
server_name (*http*), 409
server_name (*stream*), 522
server_name_in_redirect (*http*), 411
server_names_hash_bucket_size (*http*), 411
server_names_hash_bucket_size (*stream*), 523
server_names_hash_max_size (*http*), 411
server_names_hash_max_size (*stream*), 524
server_tokens (*http*), 411
set (*http*), 263
set (*stream*), 477
set_real_ip_from, 537
set_real_ip_from (*http*), 257
set_real_ip_from (*stream*), 476
slice (*http*), 286

smtp_auth, 537
smtp_capabilities, 538
smtp_client_buffer, 538
smtp_greeting_delay, 538
source_charset (*http*), 91
split_clients (*http*), 287
split_clients (*stream*), 478
ssi (*http*), 288
ssi_last_modified (*http*), 288
ssi_min_file_chunk (*http*), 288
ssi_silent_errors (*http*), 288
ssi_types (*http*), 289
ssi_value_length (*http*), 289
ssl_alpn (*stream*), 479
ssl_buffer_size (*http*), 294
ssl_certificate, 540
ssl_certificate (*http*), 294
ssl_certificate (*stream*), 480
ssl_certificate_cache (*http*), 295
ssl_certificate_cache (*stream*), 480
ssl_certificate_compression, 540
ssl_certificate_compression (*http*), 296
ssl_certificate_compression (*stream*), 481
ssl_certificate_key, 541
ssl_certificate_key (*http*), 296
ssl_certificate_key (*stream*), 481
ssl_ciphers, 541
ssl_ciphers (*http*), 297
ssl_ciphers (*stream*), 482
ssl_client_certificate, 541
ssl_client_certificate (*http*), 297
ssl_client_certificate (*stream*), 482
ssl_conf_command, 542
ssl_conf_command (*http*), 298
ssl_conf_command (*stream*), 483
ssl_crl, 542
ssl_crl (*http*), 298
ssl_crl (*stream*), 483
ssl_dhparam, 542
ssl_dhparam (*http*), 298
ssl_dhparam (*stream*), 483
ssl_early_data (*http*), 299
ssl_early_data (*stream*), 484
ssl_ecdh_curve, 543
ssl_ecdh_curve (*http*), 299
ssl_ecdh_curve (*stream*), 484
ssl_encrypted_hello_key (*http*), 300
ssl_encrypted_hello_key (*stream*), 485
ssl_engine (*core*), 27
ssl_handshake_timeout (*stream*), 485
ssl_keylog_file (*http*), 300
ssl_ntls (*http*), 300
ssl_ntls (*stream*), 485
ssl_object_cache_inheritable (*core*), 27
ssl_ocsp (*http*), 301
ssl_ocsp (*stream*), 485
ssl_ocsp_cache (*http*), 301
ssl_ocsp_cache (*stream*), 486
ssl_ocsp_responder (*http*), 302
ssl_ocsp_responder (*stream*), 486
ssl_password_file, 543
ssl_password_file (*http*), 302
ssl_password_file (*stream*), 486
ssl_prefer_server_ciphers, 544
ssl_prefer_server_ciphers (*http*), 302
ssl_prefer_server_ciphers (*stream*), 487
ssl_preread (*stream*), 496
ssl_protocols, 544
ssl_protocols (*http*), 303
ssl_protocols (*stream*), 487
ssl_reject_handshake (*http*), 303
ssl_session_cache, 544
ssl_session_cache (*http*), 303
ssl_session_cache (*stream*), 488
ssl_session_ticket_key, 545
ssl_session_ticket_key (*http*), 304
ssl_session_ticket_key (*stream*), 488
ssl_session_tickets, 545
ssl_session_tickets (*http*), 304
ssl_session_tickets (*stream*), 489
ssl_session_timeout, 546
ssl_session_timeout (*http*), 305
ssl_session_timeout (*stream*), 489
ssl_stapling (*http*), 305
ssl_stapling (*stream*), 489
ssl_stapling_file (*http*), 305
ssl_stapling_file (*stream*), 489
ssl_stapling_responder (*http*), 306
ssl_stapling_responder (*stream*), 490
ssl_stapling_verify (*http*), 306

ssl_stapling_verify (*stream*), 490
ssl_trusted_certificate, 546
ssl_trusted_certificate (*http*), 306
ssl_trusted_certificate (*stream*), 490
ssl_verify_client, 546
ssl_verify_client (*http*), 306
ssl_verify_client (*stream*), 490
ssl_verify_depth, 546
ssl_verify_depth (*http*), 307
ssl_verify_depth (*stream*), 491
starttls, 547
state (*http*), 329
state (*stream*), 504
status_zone (*http*), 412
status_zone (*stream*), 524
sticky (*http*), 329
sticky (*stream*), 504
sticky_secret (*http*), 334
sticky_secret (*stream*), 509
sticky_strict (*http*), 335
sticky_strict (*stream*), 509
stream (*stream*), 525
stub_status (*http*), 311
sub_filter (*http*), 313
sub_filter_last_modified (*http*), 313
sub_filter_once (*http*), 313
sub_filter_types (*http*), 313
subrequest_output_buffer_size (*http*), 413

T

tcp_nodelay (*http*), 414
tcp_nodelay (*stream*), 525
tcp_nopush (*http*), 414
thread_pool (*core*), 28
time_format (*http*), 414
timeout, 553
timer_resolution (*core*), 28
try_files (*http*), 415
tunnel_bind (*http*), 314
tunnel_buffer_size (*http*), 315
tunnel_connect_timeout (*http*), 315
tunnel_next_upstream (*http*), 315
tunnel_next_upstream_timeout (*http*), 315
tunnel_next_upstream_tries (*http*), 316
tunnel_pass (*http*), 316
tunnel_read_timeout (*http*), 316
tunnel_send_lowat (*http*), 317
tunnel_send_timeout (*http*), 317
tunnel_socket_keepalive (*http*), 317
types (*http*), 418
types_hash_bucket_size (*http*), 418
types_hash_max_size (*http*), 419

U

underscores_in_headers (*http*), 419
uninitialized_variable_warn (*http*), 263
upstream (*http*), 335
upstream (*stream*), 509
upstream_probe (*http*), 340
upstream_probe (*stream*), 513
upstream_probe_timeout (*stream*), 515
use (*core*), 29
user (*core*), 29
userid (*http*), 343
userid_domain (*http*), 343
userid_expires (*http*), 343
userid_flags (*http*), 343
userid_mark (*http*), 343
userid_name (*http*), 344
userid_p3p (*http*), 344
userid_path (*http*), 344
userid_service (*http*), 344
uwsgi_bind (*http*), 345
uwsgi_buffer_size (*http*), 346
uwsgi_buffering (*http*), 346
uwsgi_buffers (*http*), 347
uwsgi_busy_buffers_size (*http*), 347
uwsgi_cache (*http*), 347
uwsgi_cache_background_update (*http*), 348
uwsgi_cache_bypass (*http*), 348
uwsgi_cache_key (*http*), 348
uwsgi_cache_lock (*http*), 349
uwsgi_cache_lock_age (*http*), 349
uwsgi_cache_lock_timeout (*http*), 349
uwsgi_cache_max_range_offset (*http*), 349
uwsgi_cache_methods (*http*), 349
uwsgi_cache_min_uses (*http*), 350
uwsgi_cache_path (*http*), 350
uwsgi_cache_revalidate (*http*), 352
uwsgi_cache_use_stale (*http*), 352

uwsgi_cache_valid (*http*), 353
uwsgi_connect_timeout (*http*), 354
uwsgi_connection_drop (*http*), 354
uwsgi_force_ranges (*http*), 354
uwsgi_hide_header (*http*), 354
uwsgi_ignore_client_abort (*http*), 354
uwsgi_ignore_headers (*http*), 355
uwsgi_intercept_errors (*http*), 355
uwsgi_limit_rate (*http*), 355
uwsgi_max_temp_file_size (*http*), 356
uwsgi_modifier1 (*http*), 356
uwsgi_modifier2 (*http*), 356
uwsgi_next_upstream (*http*), 357
uwsgi_next_upstream_timeout (*http*), 358
uwsgi_next_upstream_tries (*http*), 358
uwsgi_no_cache (*http*), 358
uwsgi_param (*http*), 359
uwsgi_pass (*http*), 359
uwsgi_pass_header (*http*), 360
uwsgi_pass_request_body (*http*), 360
uwsgi_pass_request_headers (*http*), 360
uwsgi_read_timeout (*http*), 360
uwsgi_request_buffering (*http*), 360
uwsgi_send_timeout (*http*), 361
uwsgi_socket_keepalive (*http*), 361
uwsgi_ssl_certificate (*http*), 361
uwsgi_ssl_certificate_cache (*http*), 362
uwsgi_ssl_certificate_key (*http*), 362
uwsgi_ssl_ciphers (*http*), 362
uwsgi_ssl_conf_command (*http*), 363
uwsgi_ssl_crl (*http*), 363
uwsgi_ssl_name (*http*), 364
uwsgi_ssl_password_file (*http*), 364
uwsgi_ssl_protocols (*http*), 364
uwsgi_ssl_server_name (*http*), 364
uwsgi_ssl_session_reuse (*http*), 365
uwsgi_ssl_trusted_certificate (*http*), 365
uwsgi_ssl_verify (*http*), 365
uwsgi_ssl_verify_depth (*http*), 365
uwsgi_store (*http*), 365
uwsgi_store_access (*http*), 366
uwsgi_temp_file_write_size (*http*), 367
uwsgi_temp_path (*http*), 367

V

valid_referers (*http*), 258
variables_hash_bucket_size (*http*), 419
variables_hash_bucket_size (*stream*), 526
variables_hash_max_size (*http*), 419
variables_hash_max_size (*stream*), 526

W

wamr_global_heap_size, 554
wamr_heap_size, 554
wamr_stack_size, 554
wasm_modules, 557
wasmtime_enable_wasi, 555
wasmtime_stack_size, 555
worker_aio_requests (*core*), 29
worker_connections (*core*), 29
worker_cpu_affinity (*core*), 29
worker_priority (*core*), 30
worker_processes (*core*), 31
worker_rlimit_core (*core*), 31
worker_rlimit_nofile (*core*), 31
worker_shutdown_timeout (*core*), 31
working_directory (*core*), 32

X

xclient, 536
xml_entities (*http*), 375
xslt_last_modified (*http*), 375
xslt_param (*http*), 375
xslt_string_param (*http*), 375
xslt_stylesheet (*http*), 376
xslt_types (*http*), 376

Z

zone (*http*), 336
zone (*stream*), 510