Руководство по эксплуатации

Версия 1.0

Аннотация

Настоящий документ содержит информацию, необходимую для эксплуатации программного обеспечения Angie PRO.

Оглавление

1 Общие сведения....4
2 Управление Angie PRO....4
2.1 Изменение конфигурации....5
2.2 Ротация лог-файлов....5
2.3 Обновление исполняемого файла на лету....6
2.4 Управление из командной строки....6
3 Как работает Angie PRO....7
3.1 Методы обработки соединений....7
3.2 Обработка TCP/UDP-сессий....8
3.3 Обработка запросов....8
3.4 Проксирование и балансировка нагрузки....13
3.5 Логирование....15
4 Конфигурация....18
4.1 Наследование....19
4.2 Перезагрузка конфигурации....19
4.3 Единицы измерения....19
4.4 Настройка хэшей....20
4.5 Настройка HTTPS-серверов....20
5 Директивы....26
5.1 Модуль API....26
5.2 Модуль core....45
5.3 Модули http....54
5.4 Модули stream....281
5.5 Модули mail....332

1 Общие сведения

Angie PRO – единственный коммерческий веб-сервер разработка которого локализована в России.

Веб-сервер — это класс программного обеспечения, предоставляющего доступ к сетевым ресурсам по протоколу HTTP конечным пользователям.

Angie PRO, например, может использоваться для работы интернет-сайтов, мобильных приложений, киосков самообслуживания в метрополитене, мультимедиа систем в поездах дальнего следования.

Для начала работы с Angie PRO установите его в своем окружении. Поддерживаемые дистрибутивы и инструкция по установке приведены в руководстве по установке Angie PRO.

2 Управление Angie PRO

Процесс Angie PRO запускается как системный сервис следующей командой:

```
sudo service angie start
```

Рекомендуется перед стартом выполнить проверку синтаксиса конфигурационного файла, команда:

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

Команда для перезагрузки:

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

Команда для остановки процесса Angie:

```
sudo service angie stop
```

После первичной установки можно проверить, что Angie успешно запустился:

```
curl localhost:80
```

У Angie PRO есть один главный и несколько рабочих процессов. Основная задача главного процесса — чтение и проверка конфигурации и управление рабочими процессами. Рабочие процессы выполняют фактическую обработку запросов. Angie PRO использует модель, основанную на событиях, и зависящие от операционной системы механизмы для эффективного распределения запросов между рабочими процессами. Количество рабочих процессов задаётся в конфигурационном файле и может быть фиксированным для данной конфигурации или автоматически устанавливаться равным числу доступных процессорных ядер.

Управлять Angie PRO можно также с помощью сигналов. Номер главного процесса по умолчанию записывается в файл /var/run/angie.pid. Изменить имя этого файла можно при конфигурации сборки или же в angie.conf директивой pid. Главный процесс поддерживает следующие сигналы:

TERM, INT	быстрое завершение
QUIT	плавное завершение
HUP	изменение конфигурации, обновление изменившейся временной зоны (только для FreeBSD и Linux), запуск новых рабочих процессов с новой конфигурацией, плавное завершение старых рабочих процессов
USR1	переоткрытие лог-файлов
USR2	обновление исполняемого файла
WINCH	плавное завершение рабочих процессов

Управлять рабочими процессами по отдельности не нужно. Тем не менее, они тоже поддерживают некоторые сигналы:

TERM, INT	быстрое завершение
QUIT	плавное завершение
USR1	переоткрытие лог-файлов
WINCH	аварийное завершение для отладки (требует включения debug_points)

2.1 Изменение конфигурации

Для того чтобы Angie PRO перечитал файл конфигурации, нужно послать главному процессу сигнал HUP. Главный процесс сначала проверяет синтаксическую правильность конфигурации, а затем пытается применить новую конфигурацию, то есть, открыть лог-файлы и новые слушающие сокеты. Если ему это не удаётся, то он откатывает изменения и продолжает работать со старой конфигурацией. Если же удаётся, то он запускает новые рабочие процессы, а старым шлёт сообщение о плавном выходе. Старые рабочие процессы закрывают слушающие сокеты и продолжают обслуживать старых клиентов. После обслуживания всех клиентов старые рабочие процессы завершаются.

2.2 Ротация лог-файлов

Лог-файлы нужно переименовать, а затем послать сигнал USR1 главному процессу. Он откроет заново все текущие открытые файлы и назначит им в качестве владельца непривилегированного пользователя, под которым работают рабочие процессы. После успешного открытия главный процесс закрывает все открытые файлы и посылает сообщение о переоткрытии файлов рабочим процессам. Они также открывают новые файлы и сразу же закрывают старые. В результате старые файлы практически сразу же готовы для дальнейшей обработки, например, их можно сжимать.

2.3 Обновление исполняемого файла на лету

Для обновления исполняемого файла сервера вначале нужно записать на место старого файла новый. Затем нужно послать сигнал USR2 главному процессу — он переименует свой файл с номером процесса в файл с суффиксом .oldbin, например, /usr/local/angie/logs/angie.pid.oldbin, после чего запустит новый исполняемый файл, а тот в свою очередь — свои рабочие процессы.

Старый процесс не закрывает свои слушающие сокеты и при необходимости ему можно сказать, чтобы он снова запустил свои рабочие процессы. Если работа нового исполняемого файла по каким-то причинам не устраивает, можно проделать одно из следующих действий:

- Послать старому главному процессу сигнал HUP. Старый главный процесс, не перечитывая конфигурации, запустит новые рабочие процессы. После этого можно плавно завершить все новые процессы, послав новому главному процессу сигнал QUIT.
- Послать новому главному процессу сигнал TERM. В ответ на это он пошлёт сообщение о немедленном выходе своим рабочим процессам, и все они практически сразу же завершатся. (Если новые процессы по каким-то причинам не завершаются, нужно послать им сигнал KILL, который заставит их завершиться.) По завершению нового главного процесса старый главный процесс автоматически запустит новые рабочие процессы.

Если новый главный процесс выходит, то старый главный процесс убирает суффикс .oldbin из имени файла с номером процесса.

Если же обновление прошло удачно, то старому процессу нужно послать сигнал QUIT, и останутся только новые процессы.

2.4 Управление из командной строки

-? | -h	вывод справки по параметрам командной строки
-c файл	использование альтернативного конфигурационного файла файл вместо файла по умолчанию.
-e файл	использование альтернативного лог-файла ошибок файл вместо файла по умолчанию. Специальное значение stderr выбирает стандартный файл ошибок
-g директивы	задание глобальных директив конфигурации, например: angie -g "pid /var/run/angie.pid; worker_processes `sysctl -n hw.ncpu`;"
-p префикс	задание префикса пути angie, т.е. каталога, в котором будут находиться файлы сервера (по умолчанию — каталог /usr/local/angie)
-q	вывод только сообщений об ошибках при тестировании конфигурации

-s сигнал	отправка сигнала главному процессу: stop, quit, reopen, reload
-t	тестирование конфигурационного файла: Angie проверяет синтаксическую правильность конфигурации, а затем пытается открыть файлы, описанные в конфигурации
-T	то же, что и -t, а также вывод конфигурационных файлов в стандартный поток вывода
-v	вывод версии Angie PRO
-V	вывод версии Angie PRO, версии компилятора и параметров конфигурации сборки

3 Как работает Angie PRO

3.1 Методы обработки соединений

Angie PRO поддерживает различные методы обработки соединений. Наличие того или иного метода зависит от используемой платформы. Если на платформе доступно сразу несколько методов, Angie PRO обычно сам выбирает наиболее эффективный метод. Однако при необходимости можно явно выбрать метод обработки соединений с помощью директивы use.

Поддерживаются следующие методы обработки соединений:

select

Стандартный метод. Модуль для поддержки этого метода собирается автоматически, если на платформе не обнаружено более эффективного метода. Можно принудительно разрешить или запретить сборку этого модуля с помощью параметров --with-select_module и --without-select_module.

poll

Стандартный метод. Модуль для поддержки этого метода собирается автоматически, если на платформе не обнаружено более эффективного метода. Можно принудительно разрешить или запретить сборку этого модуля с помощью параметров --with-poll_module и --without-poll_module.

kqueue

Эффективный метод, используемый во FreeBSD 4.1+, OpenBSD 2.9+, NetBSD 2.0 и macOS.

epoll

Эффективный метод, используемый в Linux 2.6+.

/dev/poll

Эффективный метод, используемый в Solaris 7 11/99+, HP/UX 11.22+ (eventport), IRIX 6.5.15+ и Tru64 UNIX 5.1A+.

eventport

event ports, метод, используемый в Solaris 10+ (из-за имеющихся проблем вместо него рекомендуется использовать метод /dev/poll).

3.2 Обработка TCP/UDP-сессий

Обработка клиентской TCP/UDP-сессии происходит последовательными фазами:

Post-accept	Первая фаза после принятия клиентского соединения. В этой фазе выполняется модуль stream_realip.
Pre-access	Предварительная проверка доступа. В этой фазе выполняются модули stream_limit_conn и stream_set.
Access	Ограничение доступа для клиента перед обработкой данных. В этой фазе выполняется модуль stream_access.
SSL	Терминирование TLS/SSL. В этой фазе выполняется модуль stream_ssl.
Preread	Чтение первых байт данных в буфер предварительного чтения для анализа, например модулем stream_ssl_preread, перед их обработкой.
Content	Обязательная фаза, в которой происходит обработка данных, как правило проксирование на группу серверов или отправка клиенту заданного значения.
Log	Заключительная фаза, в которой записывается результат обработки клиентской сессии. В этой фазе выполняется модуль stream_log.

3.3 Обработка запросов

Выбор виртуального сервера

Сначала соединение создаётся в контесте сервера по умолчанию. Затем имя сервера может быть определено на следующих стадиях обработки запроса, каждая из которых участвует в выборе конфигурации:

- предварительно во время операции SSL handshake согласно SNI
- после обработки строки запроса
- после обработки поля “Host“ заголовка запроса

Если после обработки строки запроса или поля “Host” заголовка запроса имя сервера не было выбрано, то Angie будет использовать пустое имя в качестве имени сервера.

На каждой из этих стадий могут применяться различные конфигурации сервера. То есть, некоторые директивы следует указывать с осторожностью:

- в случае использования директивы ssl_protocols список протоколов задаётся библиотекой OpenSSL перед применением конфигурации сервера согласно имени, запрашиваемого через SNI. Таким образом, протоколы должны быть заданы только для сервера по умолчанию;
- директивы client_header_buffer_size и merge_slashes задействуются перед чтением строки запроса, они используют конфигурацию сервера по умолчанию или конфигурацию сервера, выбранного через SNI;
- в случае использования директив ignore_invalid_headers, large_client_header_buffers и underscores_in_headers, которые участвуют в обработке полей заголовка запроса, выбор сервера дополнительно зависит от того, была ли обновлена конфигурация сервера согласно строке запроса или полю заголовка “Host”;
- ошибочный ответ будет обработан с помощью директивы error_page в том сервере, который в настоящий момент выполняет запрос.

Определение виртуального сервера по имени

Angie PRO вначале решает, какой из серверов должен обработать запрос. Рассмотрим простую конфигурацию, где все три виртуальных сервера слушают на порту **: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;
#  ...
}
```

В этой конфигурации, чтобы определить, какому серверу следует направить запрос, Angie PRO проверяет только поле “Host” заголовка запроса. Если его значение не соответствует ни одному из имён серверов или в заголовке запроса нет этого поля вовсе, Angie направит запрос в сервер по умолчанию для этого порта. В вышеприведённой конфигурации сервером по умолчанию будет первый сервер, что соответствует стандартному поведению Angie по умолчанию. Сервер по умолчанию можно задать явно с помощью параметра *default_server* в директиве listen:

```
server {
  listen      80 default_server;
```

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

Примечание

Следует иметь в виду, что сервер по умолчанию является свойством слушающего сокета, а не имени сервера.

Как предотвратить обработку запросов без имени сервера

Если запросы без поля “Host” в заголовке не должны обрабатываться, можно определить сервер, который будет их отклонять:

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

Здесь в качестве имени сервера указана пустая строка, которая соответствует запросам без поля “Host” в заголовке, и возвращается специальный код 444, который закрывает соединение.

Определение виртуального сервера по имени и IP-адресу

Рассмотрим более сложную конфигурацию, в которой некоторые виртуальные серверы слушают на разных адресах:

```
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;
#  ...
}
```

В этой конфигурации Angie PRO вначале сопоставляет IP-адрес и порт запроса с директивами listen в блоках server. Затем он сопоставляет значение поля “Host” заголовка запроса с директивами server_name в блоках server, которые соответствуют IP-адресу и порту. Если имя сервера не найдено, запрос будет обработан в сервере по умолчанию.

Например, запрос www.example.com, пришедший на порт 192.168.1.1:80, будет обработан сервером по умолчанию для порта 192.168.1.1:80, т.е. первым сервером, т.к. для этого порта *www.example.com* не указан в списке имён серверов.

Сервер по умолчанию является свойством слушающего сокета, поэтому у разных сокетов могут быть определены свои серверы по умолчанию:

```
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;
#  ...
}
```

Определение location

На примере простого PHP-сайта:

```
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 PRO вначале ищет среди всех префиксных *location*, заданных строками, максимально совпадающий. В вышеприведённой конфигурации указан только один префиксный *location /*, и, поскольку он подходит под любой запрос, он и будет использован,

если других совпадений не будет найдено. Затем Angie PRO проверяет *location*, заданные регулярными выражениями, в порядке их следования в конфигурационном файле. При первом же совпадении поиск прекращается и Angie PRO использует совпавший *location*. Если запросу не соответствует ни одно из регулярных выражений, Angie PRO использует максимально совпавший префиксный *location*, найденный ранее.

Примечание

Следует иметь в виду, что *location* всех типов сопоставляются только с URI-частью строки запроса без аргументов. Так делается потому, что аргументы в строке запроса могут быть заданы различными способами, например:

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

Кроме того, в строке запроса можно запросить что угодно:

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

Теперь посмотрим, как бы обрабатывались запросы в вышеприведённой конфигурации:

Запросу /logo.gif во-первых соответствует префиксный *location /*, а во-вторых — регулярное выражение *.(gif|jpg|png)$*, поэтому он обрабатывается location'ом регулярного выражения. Согласно директиве *root /data/www* запрос отображается в файл /data/www/logo.gif, который и посылается клиенту.

Запросу /index.php также во-первых соответствует префиксный *location /*, а во-вторых — регулярное выражение *.(php)$*. Следовательно, он обрабатывается *location*'ом регулярного выражения и запрос передаётся FastCGI-серверу, слушающему на *localhost:9000*. Директива fastcgi_param устанавливает FastCGI-параметр *SCRIPT_FILENAME* в */data/www/index.php*, и сервер FastCGI выполняет указанный файл. Переменная $document_root равна значению директивы *root*, а переменная $fastcgi_script_name равна URI запроса, т.е. /index.php.

Запросу /about.html соответствует только префиксный *location /*, поэтому запрос обрабатывается в нём. Согласно директиве *root /data/www* запрос отображается в файл /data/www/about.html, который и посылается клиенту.

Обработка запроса / более сложная. Ему соответствует только префиксный *location /*, поэтому запрос обрабатывается в нём. Затем директива index проверяет существование индексных файлов согласно своих параметров и директиве *root /data/www*. Если файл /data/www/index.html не существует, а файл /data/www/index.php существует, то директива делает внутреннее перенаправление на /index.php и Angie снова сопоставляет его с *location*'ами, как если бы такой запрос был послан клиентом. Как мы видели ранее, перенаправленный запрос будет в конечном итоге обработан сервером FastCGI.

3.4 Проксирование и балансировка нагрузки

Одним из частых применений Angie PRO является использование его в качестве прокси-сервера, то есть сервера, который принимает запросы, перенаправляет их на проксируемые сервера, получает ответы от них и отправляет их клиенту.

Простейшая конфигурация прокси-сервера:

```
server {
    location / {
        proxy_pass http://backend:8080;
    }
```

Директива proxy_pass инструктирует Angie PRO передавать клентские запросы на сервер бэкенда *backend:8080* (проксируемый сервер). Проксирование посредством Angie PRO гибко конфигурируется множеством других директив.

Проксирование FastCGI

Angie PRO можно использовать для перенаправления запросов на FastCGI-серверы. На них могут исполняться приложения, созданные с использованием разнообразных фреймворков и языков программирования, например, PHP.

Базовая конфигурация Angie PRO для работы с проксируемым FastCGI-сервером включает в себя использование директивы fastcgi_pass вместо директивы proxy_pass, и директив fastcgi_param для настройки параметров, передаваемых FastCGI-серверу. Представьте, что FastCGI-сервер доступен по адресу *localhost:9000*. В PHP параметр SCRIPT_FILENAME используется для определения имени скрипта, а в параметре QUERY_STRING передаются параметры запроса. Получится следующая конфигурация:

```
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;
    }
}
```

Таким образом будет настроен сервер, который будет перенаправлять все запросы, кроме запросов статических изображений, на проксируемый сервер, работающий по адресу localhost:9000, по протоколу FastCGI.

Проксирование WebSocket

Для превращения соединения между клиентом и сервером из HTTP/1.1 в WebSocket используется доступный в HTTP/1.1 механизм смены протокола.

Но есть сложность: поскольку “Upgrade” является hop-by-hop заголовком, то он не передаётся от клиента к проксируемому серверу. При прямом проксировании клиенты могут использовать метод CONNECT, чтобы обойти эту проблему. Однако при обратном проксировании такой подход не работает, так как клиент ничего о проксирующем сервере не знает, и требуется специальная обработка на проксирующем сервере.

В Angie PRO предусмотрен особый режим работы, который позволяет установить туннель между клиентом и проксируемым сервером, если проксируемый сервер вернул ответ с кодом 101 (Switching Protocols), и клиент попросил сменить протокол с помощью заголовка “Upgrade” в запросе.

Как уже отмечалось выше, hop-by-hop заголовки, включая “Upgrade” и “Connection”, не передаются от клиента к проксируемому серверу, поэтому, для того чтобы проксируемый сервер узнал о намерении клиента сменить протокол на WebSocket, эти заголовки следует передать явно:

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

Более сложный пример, в котором значение поля “Connection” в заголовке запроса к проксируемому серверу зависит от наличия поля “Upgrade” в заголовке запроса клиента:

```
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;
        }
    }
```

По умолчанию соединение будет закрыто, если с проксируемого сервера данные не передавались в течение 60 секунд. Этот таймаут можно увеличить при помощи директивы proxy_read_timeout. Кроме того, на проксируемом сервере можно настроить периодическую отправку WebSocket ping-фреймов для сброса таймаута и проверки работоспособности соединения.

Балансировка нагрузки

Для оптимизации использования ресурсов, повышения устойчивости и скорости обработки запросов широко используется распределение нагрузки клиентского трафика между несколькими серверами приложений. Angie эффективно решает эту задачу.

Простейшая конфигурация для распределения клиентского трафика по трём бэкендам выглядит так:

```
http {
    upstream myapp1 {
        server srv1.example.com;
        server srv2.example.com;
        server srv3.example.com;
    }

    server {
        listen 80;

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

В примере три сервера srv1-srv3 одного приложения объединены в группу *upstream* и при проксировании Angie PRO распределит между ними клиентские запросы. По умолчанию используется алгоритм распределения round-robin. Возможно использование других алгоритмов: weight, least_conn, ip_hash. Для группы серверов upstream по умолчанию работает механизм проверки работоспособности каждого сервера, настраиваемый в контексте upstream параметрами max_fails и fail_timeout директивы server.

3.5 Логирование

Запись в syslog

Директивы error_log и access_log поддерживают запись в *syslog*. Запись в *syslog* настраивается при помощи следующих параметров:

server= *адрес*	Задаёт адрес сервера *syslog*. Адрес может быть указан в виде доменного имени или IP-адреса, и необязательного порта, или в виде пути UNIX-сокета, который указывается после префикса *“unix:”*. Если порт не указан, используется UDP-порт 514. Если доменному имени соответствует несколько IP-адресов, используется только первый адрес.
facility= *строка*	Задаёт категорию сообщений syslog в соответствии с RFC 3164. В качестве категории может быть указано одно из следующих значений: *“kern”*, *“user”*, *“mail”*, *“daemon”*, *“auth”*, *“intern”*, *“lpr”*, *“news”*, *“uucp”*, *“clock”*, *“authpriv”*, *“ftp”*, *“ntp”*, *“audit”*, *“alert”*, *“cron”*, *“local0”..“local7”*. По умолчанию используется *“local7”*.

severity= *строка*	Задаёт важность сообщений *syslog* для access_log в соответствии с RFC 3164. Возможны те же самые значения, что и у второго параметра (уровень) директивы error_log. По умолчанию используется *“info”*. Важность сообщений об ошибках определяется самим Angie Pro, поэтому в директиве error_log параметр игнорируется.
tag= *строка*	Задаёт метку сообщений syslog. По умолчанию используется *“angie”*.
nohostname	Запрещает добавление поля “hostname” в заголовок сообщения syslog

Пример конфигурации syslog:

```
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;
```

Отладочный лог

Для работы отладочного лога Angie PRO должен быть сконфигурирован с поддержкой отладки на этапе сборки:

./configure –with-debug …

Затем нужно задать уровень debug с помощью директивы error_log:

```
error_log /path/to/log debug;
```

Чтобы убедиться, что поддержка отладки сконфигурирована, необходимо выполнить команду *angie -V*:

configure arguments: –with-debug …

Готовые пакеты для Linux по умолчанию предоставляют поддержку отладочного лога при помощи бинарного файла angie-debug. Из пакета бинарные файлы Angie располагаются:

```
$ 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
```

systemd service Angie сконфигурирован:

```
DAEMON=${DAEMON:-/usr/sbin/angie}
```

Поэтому для того, чтобы запустить бинарный файл Angie PRO с поддержкой отладочного лога, нужно переназначить symlink:

```
$ ln -fs /usr/sbin/angie-debug /usr/sbin/angie
```

и выполнить команду

```
$ sudo service angie upgrade
```

которая запустит процедуру обновления исполняемого файла на лету, и затем задать уровень *debug*.

Обратите внимание, что переопределение лога без одновременного указания уровня *debug* отключит отладочный лог. В примере ниже, переопределение лога на уровне server отключает отладочный лог для этого сервера:

```
error_log /path/to/log debug;

http {
   server {
     error_log /path/to/log;
    # ...
```

Чтобы избежать этого, следует либо закомментировать строку, переопределяющую лог, либо добавить определение уровня *debug*:

```
error_log /path/to/log debug;

http {
   server {
     error_log /path/to/log debug;
   #  ...
```

Отладочный лог для определённых клиентов

Можно включить отладочный лог только для определённых клиентских адресов:

```
error_log /path/to/log;

events {
  debug_connection 192.168.1.1;
  debug_connection 192.168.10.0/24;
}
```

Запись в кольцевой буфер в памяти

Отладочный лог можно записывать в кольцевой буфер в памяти:

```
error_log memory:32m debug;
```

Запись в буфер в памяти на уровне *debug* не оказывает существенного влияния на производительность даже при высоких нагрузках. В этом случае лог может быть извлечён при помощи gdb-скрипта, подобного следующему:

```
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 Конфигурация

Angie PRO работает с текстовым конфигурационным файлом. Файл конфигурации *angie.conf* находится по пути conf-path, указанном при компиляции, по умолчанию в */etc/angie*.

Файл конфигурации состоит из контекстов:

- *events* – обработка соединений
- *http* – трафик HTTP
- *mail* – Mail трафик
- *stream* – TCP и UDP трафик

Директивы, расположенные вне этих контекстов, считаются директивами контекста *main*.

```
user angie; # директива в контексте 'main'

events {
    # конфигурация обработки соединений
}

http {
    # Конфигурация трафика HTTP, для всех вложенных виртуальных серверов

    server {
        # конфигурация виртуального HTTP сервера 1
        location /one {
            # конфигурация обработки HTTP запросов с URI, начинающимися с '/one'
        }
        location /two {
            # конфигурация обработки HTTP запросов с URI, начинающимися с '/two'
        }
    }

    server {
        # конфигурация виртуального HTTP сервера 2
    }
}

stream {
    # Конфигурация трафика TCP/UDP, для всех вложенных виртуальных серверов
    server {
        # конфигурация виртуального TCP сервера 1
    }
}
```

Чтобы облегчить управление конфигурацией, рекомендуется использование директивы include в основном файле конфигурации angie.conf для включения в него специализированных файлов.

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

4.1 Наследование

Вложенный контекст — включённый внутри родительского, — наследует директивы из родительского контекста тогда и только тогда, когда такие директивы не описаны в нём самом. При наличии, директива вложенного контекста переопределяет директиву родительского.

4.2 Перезагрузка конфигурации

При любом изменении конфигурации, для применения изменений процесс Angie PRO необходимо:

либо перезапустить полностью, предварительно проверив конфигурацию синтаксически:

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

либо перезагрузить, что позволит не прерывать обработку текущих соединений.

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

4.3 Единицы измерения

Размеры в конфигурационном файле можно указывать в байтах, килобайтах (суффиксы *k* и *K*) или мегабайтах (суффиксы *m* и *M*), например, *“1024”*, *“8k”*, *“1m”*.

Интервалы времени можно задавать в миллисекундах, секундах, минутах, часах, днях и т.д., используя следующие суффиксы:

ms	миллисекунды
s	секунды
m	минуты
h	часы
d	дни
w	недели
M	месяцы, 30 дней
y	годы, 365 дней

В одном значении можно комбинировать различные единицы, указывая их в порядке от более к менее значащим, и по желанию отделяя их пробелами. Например, *“1h 30m”* задаёт то же время, что и *“90m”* или *“5400s”*. Значение без суффикса задаёт секунды. Рекомендуется всегда указывать суффикс.

Некоторые интервалы времени можно задать лишь с точностью до секунд.

4.4 Настройка хэшей

Для быстрой обработки статических наборов данных, таких как имена серверов, значения директивы map, MIME-типы, имена полей заголовков запросов, Angie PRO использует хэш-таблицы. Во время старта и при каждой переконфигурации Angie подбирает минимально возможный размер хэш-таблиц с учётом того, чтобы размер корзины, куда попадают ключи с совпадающими хэш-значениями, не превышал заданного параметра (*hash bucket size*). Размер таблицы считается в корзинах. Подбор ведётся до тех пор, пока размер таблицы не превысит параметр *hash max size*. Для большинства хэшей есть директивы, которые позволяют менять эти параметры, например, для хэшей имён серверов директивы называются server_names_hash_max_size и server_names_hash_bucket_size.

Параметр *hash bucket size* всегда выравнивается до размера, кратного размеру строки кэша процессора. Это позволяет ускорить поиск ключа в хэше на современных процессорах, уменьшив число обращений к памяти. Если *hash bucket size* равен размеру одной строки кэша процессора, то во время поиска ключа число обращений к памяти в худшем случае будет равно двум — первый раз для определения адреса корзины, а второй — при поиске ключа внутри корзины. Соответственно, если Angie PRO выдал сообщение о необходимости увеличить *hash max size* или *hash bucket size*, то сначала нужно увеличивать первый параметр.

4.5 Настройка HTTPS-серверов

Чтобы настроить HTTPS-сервер, необходимо включить параметр ssl на слушающих сокетах в блоке server, а также указать местоположение файлов с сертификатом сервера и секретным ключом:

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

Сертификат сервера является публичным. Он посылается каждому клиенту, соединяющемуся с сервером. Секретный ключ следует хранить в файле с ограниченным

доступом (права доступа должны позволять главному процессу Angie читать этот файл). Секретный ключ можно также хранить в одном файле с сертификатом:

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

при этом права доступа к файлу следует также ограничить. Несмотря на то, что и сертификат, и ключ хранятся в одном файле, клиенту посылается только сертификат.

С помощью директив ssl_protocols и ssl_ciphers можно ограничить соединения использованием только “сильных” версий и шифров SSL/TLS. По умолчанию Angie использует:

```
ssl_protocols TLSv1 TLSv1.1 TLSv1.2;
ssl_ciphers HIGH:!aNULL:!MD5;
```

поэтому их явная настройка в общем случае не требуется.

Оптимизация HTTPS-сервера

SSL-операции потребляют дополнительные ресурсы процессора. На мультипроцессорных системах следует запускать несколько рабочих процессов, не меньше числа доступных процессорных ядер. Наиболее ресурсоёмкой для процессора является операция SSL handshake, в рамках которой формируются криптографические параметры сессии. Существует два способа уменьшения числа этих операций, производимых для каждого клиента: использование постоянных (keepalive) соединений, позволяющих в рамках одного соединения обрабатывать сразу несколько запросов, и повторное использование параметров SSL-сессии для предотвращения необходимости выполнения SSL handshake для параллельных и последующих соединений. Сессии хранятся в кэше SSL-сессий, разделяемом между рабочими процессами и настраиваемом директивой ssl_session_cache. В 1 мегабайт кэша помещается около 4000 сессий. Таймаут кэша по умолчанию равен 5 минутам. Он может быть увеличен с помощью директивы ssl_session_timeout. Вот пример конфигурации, оптимизированной под многоядерную систему с 10-мегабайтным разделяемым кэшем сессий:

```
worker_processes auto;

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

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

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

```
    #...
```

Цепочки SSL-сертификатов

Некоторые браузеры могут выдавать предупреждение о сертификате, подписанном общеизвестным центром сертификации, в то время как другие браузеры без проблем принимают этот же сертификат. Так происходит потому, что центр, выдавший сертификат, подписал его промежуточным сертификатом, которого нет в базе данных сертификатов общеизвестных доверенных центров сертификации, распространяемой вместе с браузером. В подобном случае центр сертификации предоставляет “связку” сертификатов, которую следует присоединить к сертификату сервера. Сертификат сервера следует разместить перед связкой сертификатов в скомбинированном файле:

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

Полученный файл следует указать в директиве ssl_certificate:

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

Если сертификат сервера и связка сертификатов были соединены в неправильном порядке, Angie PRO не запустится и выдаст сообщение об ошибке:

```
SSL_CTX_use_PrivateKey_file(» … /www.example.com.key») failed
```

(SSL: error:0B080074:x509 certificate routines: X509_check_private_key:key values mismatch)

поскольку Angie попытается использовать секретный ключ с первым сертификатом из связки вместо сертификата сервера.

Браузеры обычно сохраняют полученные промежуточные сертификаты, подписанные доверенными центрами сертификации, поэтому активно используемые браузеры уже могут иметь требуемые промежуточные сертификаты и не выдать предупреждение о сертификате, присланном без связанной с ним цепочки сертификатов. Убедиться в том, что сервер присылает полную цепочку сертификатов, можно при помощи утилиты командной строки openssl, например:

```
$ 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
```

Примечание

При тестировании конфигураций с SNI необходимо указывать опцию *-servername*, так как openssl по умолчанию не использует SNI.

В этом примере субъект (“s”) сертификата №0 сервера www.GoDaddy.com подписан издателем (“i”), который в свою очередь является субъектом сертификата №1, подписанного издателем, который в свою очередь является субъектом сертификата №2, подписанного общеизвестным издателем ValiCert, Inc., чей сертификат хранится во встроенной в браузеры базе данных сертификатов.

Если связку сертификатов не добавили, будет показан только сертификат сервера №0.

Единый HTTP/HTTPS сервер

Можно настроить единый сервер, который обслуживает как HTTP-, так и HTTPS-запросы:

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

Выбор HTTPS-сервера по имени

Типичная проблема возникает при настройке двух и более серверов HTTPS, слушающих на одном и том же IP-адресе:

```
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;
#...
}
```

В такой конфигурации браузер получит сертификат сервера по умолчанию, т.е. www.example.com, независимо от запрашиваемого имени сервера. Это связано с поведением протокола SSL. SSL-соединение устанавливается до того, как браузер посылает HTTP-запрос, и Angie не знает имени запрашиваемого сервера. Следовательно, он лишь может предложить сертификат сервера по умолчанию.

Наиболее старым и надёжным способом решения этой проблемы является назначение каждому HTTPS-серверу своего IP-адреса:

```
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;
#...
}
```

SSL-сертификат с несколькими именами

Существуют и другие способы, которые позволяют использовать один и тот же IP-адрес сразу для нескольких HTTPS-серверов. Все они, однако, имеют свои недостатки. Одним из таких способов является использование сертификата с несколькими именами в поле SubjectAltName сертификата, например www.example.com и www.example.org. Однако, длина поля SubjectAltName ограничена.

Другим способом является использование wildcard-сертификата, например *.example.org. Такой сертификат защищает все поддомены указанного домена, но только на заданном уровне. Под такой сертификат подходит www.example.org, но не подходят example.org и www.sub.example.org. Два вышеуказанных способа можно комбинировать. Сертификат может одновременно содержать и точное, и wildcard имена в поле SubjectAltName, например example.org и *.example.org.

Лучше поместить сведения о файле сертификата с несколькими именами и файле с его секретным ключом на уровне конфигурации http, чтобы все серверы унаследовали их единственную копию в памяти:

```
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;
#...
}
```

Указание имени сервера

Более общее решение для работы нескольких HTTPS-серверов на одном IP-адресе — расширение Server Name Indication протокола TLS (SNI, RFC 6066), которое позволяет браузеру передать запрашиваемое имя сервера во время SSL handshake, а значит сервер будет знать, какой сертификат ему следует использовать для соединения. Сейчас SNI поддерживается большинством современных браузеров, однако может не использоваться некоторыми старыми или специализированными клиентами.

Примечание

В SNI можно передавать только доменные имена, однако некоторые браузеры могут ошибочно передавать IP-адрес сервера в качестве его имени, если в запросе указан IP-адрес. Полагаться на это не следует.

Чтобы использовать SNI в Angie Pro, соответствующая поддержка должна присутствовать как в библиотеке OpenSSL, использованной при сборке бинарного файла Angie Pro, так и в библиотеке, подгружаемой в момент работы. OpenSSL поддерживает SNI начиная с версии 0.9.8f, если она была собрана с опцией конфигурации *--enable-tlsext*. Начиная с OpenSSL 0.9.8j эта опция включена по умолчанию. Если Angie был собран с поддержкой SNI, то при запуске Angie с ключом “-V” об этом сообщается:

```
$ angie -V
...
TLS SNI support enabled
...
```

Однако если Angie Pro, собранный с поддержкой SNI, в процессе работы подгружает библиотеку OpenSSL, в которой нет поддержки SNI, Angie выдаёт предупреждение:

```
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
```

5 Директивы

5.1 Модуль API

Модуль API реализует HTTP RESTful интерфейс для получения базовой информации о веб-сервере в формате JSON, а также статистики по клиентским соединениям, зонам разделяемой памяти, DNS-запросам, HTTP-запросам, кэшу HTTP-ответов, сессиям модуля stream и зонам модулей limit_conn http, limit_conn stream, limit_req и http upstream.

API принимает GET и HEAD HTTP-запросы, запрос другим методом вызовет ошибку:

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

С конфигурацией Angie PRO, включающей location /status/, зоны resolver, http upstream, http server, location, cache, limit_conn в http и limit_req для примера:

```
http {

        #...

        resolver 8.8.8.8 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 64k;
                server backend.example.com service=_http._tcp resolve max_conns=5;
           keepalive 4;
        }

        server {
                server_name www.example.com;
                listen 443 ssl http2;

                status_zone http_server_zone;
                proxy_cache cache_zone;

                access_log /var/log/access.log main;

                location / {
                        root /usr/share/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;
        }

    }
}
```

запрос: curl https://www.example.com/status/ возвращает дерево JSON


```
{
  "status": {
    "angie": {
      "version": "1.0.0",
      "address": "192.168.1.1",
      "generation": 1,
      "load_time": "2022-10-19T12:58:39.789Z"
    },

    "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": 6,
      "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": {
            "192.168.24.1:80": {
              "server": "backend.example.com",
              "service": "_http._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
              }
            }
          },
          "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
        }
      }
    }
  }
```

```
}
```

Набор метрик можно запросить по отдельной ветви JSON, построив соответствующий запрос. Например:

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

Формат Prometheus

Prometheus-совместимый ответ можно получить, добавив аргумент «format=prometheus» в строку HTTP запроса:

```
$ curl https://www.example.com/status/angie?format=prometheus
```

```
generation 1
```

```
$ curl
http://www.example.com/status/http/server_zones/<http_server_zone>?format=Prometheus
```

```
ssl_handshaked 21
ssl_reuses 0
ssl_timedout 0
ssl_failed 0
requests_total 46
requests_processing 1
requests_discarded 0
responses_200 44
responses_404 1
data_received 8634
data_sent 584725.
```

Директива api

Синтаксис api *путь*;

Умолчание —

Контекст location

Включает HTTP RESTful интерфейс в *location*.

Параметр *путь* является обязательным. Подобно директиве alias, задаёт путь для замены указанного в *location*, но по дереву API, а не файловой системы.

Если указан в префиксном *location*:

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

часть URI запроса, совпадающая с префиксом */stats/*, будет заменена на путь, указанный в параметре *путь*: */status/http/server_zones/*. К примеру, по запросу */stats/foo/* будет доступен элемент API /status/http/server_zones/foo/.

Допускается использование переменных: *api /status/$module/server_zones/$name/* и использование внутри regex location:

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

Здесь параметр *путь* определяет полный путь к элементу API, из запроса */api/location/bar/data/* будут определены переменные:

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

и запрос к API будет /status/http/location_zones/bar/data/

Также, параметр *путь* позволяет управлять доступом к API:

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

        allow 127.0.0.1;
        deny  all;
}
```

или

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

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

Метрики

Состояние сервера

/status/angie

```
{
  "version": "1.0.0",
  "address": "192.168.0.1",
  "generation": 1,
  "load_time": "2022-10-24T16:15:43.805Z"
}
```

version	String; версия запущенного сервера Angie PRO
build	String; сборка, если указана при компиляции
address	String; адрес сервера, принявшего запрос к API
generation	Number; количество перезагрузок конфигурации, полное от последнего старта
load_time	String; время последней перезагрузки конфигурации в формате ISO 8601 с миллисекундным разрешением

Соединения

/status/connections

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

accepted	Number; суммарное число принятых клиентских соединений
dropped	Number; суммарное число сброшенных клиентских соединений
active	Number: текущее число активных клиентских соединений
idle	Number; текущее число бездействующих клиентских соединений

Зоны разделяемой памяти, использующие распределение slab

Статистика использования сконфигурированных зон разделяемой памяти, таких как, например: limit_conn, limit_req и HTTP cache

```
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;
```

/status/slabs/<zone>

где <*zone*> — имя любой из сконфигурированных зон разделяемой памяти, использующей распределение slab

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

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

pages	Object; статистика по страницам памяти
used	Number; текущее число используемых страниц памяти
free	Number; текущее число свободных страниц памяти
slots	Object; статистика по слотам памяти, по каждому из размеров. *slots* содержит данные по размеру слота памяти (*8*, *16*, *32*, и т.д., вплоть до половины размера страницы памяти в байтах)
used	Number; текущее число используемых слотов памяти заданного размера
free	Number; текущее число свободных слотов памяти заданного размера
reqs	Number; суммарное число попыток выделения памяти указанного размера
fails	Number; число неудавшихся попыток выделения памяти указанного размера

DNS запросы

Сбор статистики запросов к серверу в зоне разделяемой памяти DNS включается указанием имени в параметре *status_zone*=<*name*> (status_zone) директивы resolver

```
resolver 8.8.8.8 status_zone=resolver_zone;
```

/status/resolvers/<zone>

где <*zone*> — имя зоны разделяемой памяти

```
{
```

```
  "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
  }
}
```

queries	Object; статистика запросов
name	Number; число запросов на преобразование имён в адреса (A и AAAA запросы)
srv	Number; число запросов на преобразование сервисов в адреса (SRV запросы)
addr	Number; число запросов на преобразование адресов в имена (PTR запросы)
responses	Object; статистика ответов
success	Number; число успешных ответов
timedout	Number; число запросов, не дождавшихся ответа
format_error	Number; число ответов с кодом 1 (Format error)
server_failure	Number; число ответов с кодом 2 (Server failure)
not_found	Number; число ответов с кодом 3 (Name Error)
unimplemented	Number; число ответов с кодом 4 (Not Implemented)
refused	Number; число ответов с кодом 5 (Refused)
other	Number; число запросов, завершённых с другим ненулевым кодом

Коды ответов описаны в RFC 1035, часть 4.1.1.

HTTP server и location

Для включения сбора следующих данных необходимо сконфигурировать *status_zone <zone>* (status_zone) в контексте server. Объект *ssl* наполняется данными только в случае, когда *server* сконфигурирован с *listen ssl*

```
server {
        ...
        status_zone http_server_zone;
```

/status/http/server_zones/<zone>

где *<zone>* — имя зоны разделяемой памяти для сбора *server* статистики

```
"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
}
```

ssl	Object; SSL метрики
handshaked	Number; суммарное число успешных SSL handshakes
reuses	Number; суммарное число повторных использований SSL-сессий во время операций SSL handshake
timedout	Number; суммарное число timed out SSL handshakes
failed	Number; суммарное число неуспешных SSL handshakes
requests	Object; метрики запросов
total	Number; суммарное число клиентских запросов

processing	Number; текущее число обслуживаемых клиентских запросов
discarded	Number; суммарное число запросов завершённых без отправки ответа
responses	Object; метрики ответов
<code>	Number; ненулевое число ответов со статусом <code> (100-599)
xxx	Number; ненулевое число ответов с другим кодом статуса
data	Object; метрики данных
received	Number; суммарное количество байт, полученное от клиентов
sent	Number; суммарное количество байт, отправленное клиентам

Директива *status_zone <zone>* (status_zone), определённая в контексте location и if в location, включает сбор статистики в указанной зоне разделяемой памяти. Значение off выключает сбор данных во вложенных блоках location. Набор данных для зоны *location* не включает объект *ssl* и метрику *requests/processing*

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

/status/http/location_zones/<zone>#

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

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

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

Stream server

Для включения сбора следующих данных необходимо сконфигурировать *status_zone <zone>* (status_zone) в контексте server. Объект *ssl* наполняется данными только в случае, когда *server* сконфигурирован с *listen ssl*

```
server {
        ...
        status_zone stream_server_zone;
```

/status/stream/server_zones/<zone>

где <*zone*> — имя зоны разделяемой памяти для сбора *server* статистики

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

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

  "sessions": {
    "success": 24,
    "invalid": 0,
    "forbidden": 0,
    "internal_error": 0,
    "bad_gateway": 0,
    "service_unavailable": 0
  },

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

ssl	Object; SSL метрики
handshaked	Number; суммарное число успешных SSL handshakes
reuses	Number; суммарное число повторных использований SSL-сессий во время операций SSL handshake
timedout	Number; суммарное число timed out SSL handshakes
failed	Number; суммарное число неуспешных SSL handshakes
connections	Object; метрики соединений
total	Number; суммарное число клиентских соединений
processing	Number; текущее число обслуживаемых клиентских соединений
discarded	Number: суммарное число клиентских соединений, завершённых без создания сессии
sessions	Object; метрики сессий

success	Number; число сессий, завершённых с кодом 200, что означает успешное завершение
invalid	Number; число сессий, завершённых с кодом 400, случается, когда сервер не может прочитать данные от клиента, например, заголовок PROXY protocol
forbidden	Number; число сессий, завершённых с кодом 403, когда доступ запрещён, например, ограничен для определённого адреса клиента
internal_error	Number; число сессий, завершённых с кодом 500, внтуренняя ошибка сервера
bad_gateway	Number; число сессий, завершённых с кодом 502, Bad Gateway, если, например, сервер в upstream недоступен или не может быть выбран
service_unavailable	Number; число сессий, завершённых с кодом 503, Service Unavailable, если, например, доступ ограничен числом входящих соединений
data	Object; метрики данных
received	Number; суммарное количество байт, полученное от клиентов
sent	Number; суммарное количество байт, отправленное клиентам

HTTP caches

```
proxy_cache cache_zone;
```

/status/http/caches/<cache>

Для каждой зоны, сконфигурированной в proxy_cache, хранятся следующие данные

```
{
  "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	Number; текущий размер кэша
max_size	Number; ограничение на максимальный размер кэша, если задано в конфигурации
cold	Boolean; true пока процесс *cache loader* подгружает данные с диска
hit	Object; метрики возвращённых из кэша ответов (proxy_cache_valid)
responses	Number; суммарное число ответов, прочитанных из кэша
bytes	Number; суммарное количество байт, прочитанных из кэша
stale	Object; метрики просроченных ответов, возвращённых из кэша (proxy_cache_use_stale)
responses	Number; суммарное число ответов, прочитанных из кэша
bytes	Number; суммарное количество байт, прочитанных из кэша
updating	Object; метрики просроченных ответов, возвращённых из кэша, пока данные в кэше обновляются (proxy_cache_use_stale updating)
responses	Number; суммарное число ответов, прочитанных из кэша

bytes	Number; суммарное количество байт, прочитанных из кэша
revalidated	Object; метрики просроченных и ревалидированных ответов, возвращённых из кэша (proxy_cache_revalidate)
responses	Number; суммарное число ответов, прочитанных из кэша
bytes	Number; суммарное количество байт, прочитанных из кэша
miss	Object; метрики ответов, не найденных в кэше
responses	Number; суммарное число соответствующих ответов
bytes	Number; суммарное количество байт, прочитанных с проксируемого сервера
responses_written	Number; суммарное число ответов, записанных в кэш
bytes_written	Number; суммарное количество байт, записанных в кэш
expired	Object; число ответов, возвращённых не из кэша, т.к. просрочены
responses	Number; суммарное число соответствующих ответов
bytes	Number; суммарное количество байт, прочитанных с проксируемого сервера
responses_written	Number; суммарное число ответов, записанных в кэш
bytes_written	Number; суммарное количество байт, записанных в кэш
bypass	Object; statistics of responses not looked up in the cache (proxy_cache_bypass)
responses	Number; суммарное число соответствующих ответов
bytes	Number; суммарное количество байт, прочитанных с проксируемого сервера
responses_written	Number; суммарное число ответов, записанных в кэш
bytes_written	Number; суммарное количество байт, записанных в кэш

limit_conn

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

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

Каждая из сконфигурированных зон: limit_conn в http или limit_conn в stream содержит следующие данные

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

```
  "exhausted": 0
}
```

passed	Number; суммарное число переданных на проксируемый сервер соединений
skipped	Number; суммарное число соединений, переданных с нулевым или превосходящим 255 байт <key>
rejected	Number; суммарное число соединений сверх сконфигурированного ограничения
exhausted	Number; суммарное число соединений, сброшенных из-за переполнения хранилища зоны

limit_req

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

/status/http/limit_reqs/<zone>

Каждая из сконфигурированных зон limit_req содержит следующие данные

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

passed	Number; суммарное число проксированых соединений
skipped	Number; суммарное число соединений, переданных с нулевым или превосходящим 255 байт <key>
delayed	Number; суммарное число задержанных соединений
rejected	Number; суммарное число сброшенных соединений
exhausted	Number; суммарное число соединений, сброшенных из-за переполнения хранилища зоны

HTTP upstream

Для включения сбора следующих данных необходимо сконфигурировать *zone <zone>* (zone) в контексте upstream

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

```
        server backend1.example.com weight=5;
        server backend2.example.com;
}
```

/status/http/upstreams/<upstream>

где *<upstream>* — имя апстрима, в конфигурации которого указана директива zone.

```
{
  "peers": {
    "192.168.24.1:80": {
      "server": "backend.example.com",
      "service": "_http._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
      }
    }
  },
  "keepalive": 2
}
```

peers	Object; метрики пиров апстрима
address	Object; метрики пира
server	String; сервер, как он указан в директиве server
service	String; имя сервиса, указанное в директиве server, если сконфигурировано
backup	Boolean; true для backup серверов
weight	Number; сконфигурированный weight

state	String; текущее состояние пира, одно из: *up*, *down*, или *unavailable* (когда достигнуто установленное значение max_fails)
selected	Object; статистика выбора пиров
current	Number; текущее число соединений к пиру
total	Number; общее число запросов переданных пиру
last	String; время в формате ISO 8601 последнего выбора пира
max_conns	Number; максимальное число одновременных активных соединений к пиру, если сконфигурировано
responses	Object; статистика ответов
<code>	Number; ненулевое число ответов со статусом <code> (100-599)
xxx	Number; ненулевое число ответов с другим кодом статуса
data	Object; метрики данных
received	Number; суммарное количество байт, полученное от пира
sent	Number; суммарное количество байт, отправленное пиру
health	Object; статистика по состоянию пира
fails	Number; общее количество неудачных попыток работы с пиром
unavailable	Number; столько раз пир становился *unavailable* до достижения значения max_fails
downtime	Number; суммарное время (в миллисекундах), в течение которого пир был недоступен для выбора как *unavailable*
downstart	String; время в формате ISO 8601, когда пир стал *unavailable*
keepalive	Number; текущее число кэшированных соединений

5.2 Модуль core

Пример конфигурации

```
user www www;
worker_processes 2;

error_log /var/log/error.log info;

events {
  use kqueue;
  worker_connections 2048;
}
```

Директивы

accept_mutex

Синтаксис accept_mutex on | off;

Умолчание accept_mutex off;

Контекст events

Если *accept_mutex* включён, рабочие процессы будут принимать новые соединения по очереди. В противном случае о новых соединениях будет сообщаться сразу всем рабочим процессам, и при низкой интенсивности поступления новых соединений часть рабочих процессов может работать вхолостую.

Примечание

Нет необходимости включать *accept_mutex* на системах, поддерживающих флаг *EPOLLEXCLUSIVE*, или при использовании reuseport.

accept_mutex_delay

Синтаксис accept_mutex_delay *время*;

Умолчание accept_mutex_delay 500ms;

Контекст events

При включённом accept_mutex задаёт максимальное время, в течение которого рабочий процесс вновь попытается начать принимать новые соединения, если в настоящий момент новые соединения принимает другой рабочий процесс.

daemon

Синтаксис daemon on | off;

Умолчание daemon on;

Контекст main

Определяет, будет ли Angie запускаться в режиме демона. Используется в основном для разработки.

debug_connection

Синтаксис debug_connection *адрес* | *CIDR* | unix:;

Умолчание —

Контекст events

Включает отладочный лог для отдельных клиентских соединений. Для остальных соединений используется уровень лога, заданный директивой error_log. Отлаживаемые соединения задаются IPv4 или IPv6 адресом или сетью. Соединение может быть также задано при помощи имени хоста. Отладочный лог для соединений через UNIX-сокеты включается параметром “unix:”.

```
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:;
#  ...
}
```

Примечание

Для работы директивы необходимо сконфигурировать Angie с параметром *--with-debug*, см. Отладочный лог.

debug_points

Синтаксис debug_points abort | stop;

Умолчание —

Контекст main

Эта директива используется для отладки.

В случае обнаружения внутренней ошибки, например, утечки сокетов в момент перезапуска рабочих процессов, включение *debug_points* приводит к созданию core-файла (*abort*) или остановке процесса (*stop*) с целью последующей диагностики с помощью системного отладчика.

env

Синтаксис env *переменная* [= *значение*];

Умолчание env TZ;

Контекст main

По умолчанию Angie удаляет все переменные окружения, унаследованные от своего родительского процесса, кроме переменной *TZ*. Эта директива позволяет сохранить часть унаследованных переменных, поменять им значения или же создать новые переменные окружения. Эти переменные затем:

- наследуются во время обновления исполняемого файла на лету;
- используются модулем perl;
- используются рабочими процессами. Следует иметь в виду, что управление поведением системных библиотек подобным образом возможно не всегда, поскольку зачастую библиотеки используют переменные только во время инициализации, то есть ещё до того, как их можно задать с помощью данной директивы. Исключением из этого является упомянутое выше обновление исполняемого файла на лету.

Если переменная *TZ* не описана явно, то она всегда наследуется и всегда доступна модулю perl.

Пример использования:

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

Примечание

Переменная окружения ANGIE используется для внутренних целей Angie и не должна устанавливаться непосредственно самим пользователем.

error_log

Синтаксис error_log *файл* [*уровень*];

Умолчание error_log logs/error.log error;

Контекст main, http, mail, stream, server, location

Конфигурирует запись в лог. На одном уровне конфигурации может использоваться несколько логов. Если на уровне конфигурации *main* запись лога в файл явно не задана, то используется файл по умолчанию.

Первый параметр задаёт файл, который будет хранить лог. Специальное значение *stderr* выбирает стандартный файл ошибок. Запись в syslog настраивается указанием префикса *“syslog:”*. Запись в кольцевой буфер в памяти настраивается указанием префикса *“memory:”* и размера буфера и как правило используется для отладки.

Второй параметр определяет уровень лога и может принимать одно из следующих значений: *debug*, *info*, *notice*, *warn*, *error*, *crit*, *alert* или *emerg*. Уровни лога, указанные выше, перечислены в порядке возрастания важности. При установке определённого уровня в лог попадают все сообщения указанного уровня и уровней большей важности.

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*

Если этот параметр не задан, используется *error*.

Примечание

Для работы уровня лога debug необходимо сконфигурировать Angie с *--with-debug*, см, Отладочный лог.

events

Синтаксис events { … };

Умолчание —

Контекст main

Предоставляет контекст конфигурационного файла, в котором указываются директивы, влияющие на обработку соединений.

include

Синтаксис include *файл* | *маска*;

Умолчание —

Контекст любой

Включает в конфигурацию другой файл или файлы, подходящие под заданную маску. Включаемые файлы должны содержать синтаксически верные директивы и блоки.

Пример использования:

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

load_module

Синтаксис load_module *файл*;

Умолчание —

Контекст main

Загружает динамический модуль.

Пример:

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

lock_file

Синтаксис lock_file *файл*;

Умолчание lock_file logs/angie.lock;

Контекст main

Для реализации accept_mutex и сериализации доступа к разделяемой памяти Angie использует механизм блокировок. На большинстве систем блокировки реализованы с помощью атомарных операций, и эта директива игнорируется. Для остальных систем применяется механизм файлов блокировок. Эта директива задаёт префикс имён файлов блокировок.

master_process

Синтаксис master_process on | off;

Умолчание master_process on;

Контекст main

Определяет, будут ли запускаться рабочие процессы. Эта директива предназначена для разработчиков Angie.

multi_accept

Синтаксис multi_accept on | off;

Умолчание multi_accept off;

Контекст events

on	рабочий процесс за один раз будет принимать сразу все новые соединения
off	рабочий процесс за один раз будет принимать только одно новое соединение

Примечание

Директива игнорируется в случае использования метода обработки соединений kqueue, т.к. данный метод сам сообщает число новых соединений, ожидающих приёма.

pcre_jit

Синтаксис pcre_jit on | off;

Умолчание pcre_jit off;

Контекст main

Разрешает или запрещает использование JIT-компиляции (PCRE JIT) для регулярных выражений, известных на момент парсинга конфигурации.

Использование PCRE JIT способно существенно ускорить обработку регулярных выражений.

Примечание

Для работы JIT необходима библиотека PCRE версии 8.20 или выше, собранная с параметром конфигурации *--enable-jit*. При сборке библиотеки PCRE вместе с Angie (*--with-pcre=*), для включения поддержки JIT необходимо использовать параметр конфигурации *--with-pcre-jit*.

pid

Синтаксис pid *файл*;

Умолчание pid logs/angie.pid;

Контекст main

Задаёт *файл*, в котором будет храниться номер (PID) главного процесса.

ssl_engine

Синтаксис ssl_engine *устройство*;

Умолчание —

Контекст main

Задаёт название аппаратного SSL-акселератора.

thread_pool

Синтаксис thread_pool *имя* threads = *число* [max_queue = *число*];

Умолчание thread_pool default threads=32 max_queue=65536;

Контекст main

Задаёт *имя* и параметры пула потоков, используемого для многопоточной обработки операций чтения и отправки файлов без блокирования рабочего процесса.

Параметр *threads* задаёт число потоков в пуле.

Если все потоки из пула заняты выполнением заданий, новое задание будет ожидать своего выполнения в очереди. Параметр *max_queue* ограничивает число заданий, ожидающих своего выполнения в очереди. По умолчанию в очереди может находиться до 65536 заданий. При переполнении очереди задание завершается с ошибкой.

timer_resolution

Синтаксис timer_resolution *интервал*;

Умолчание —

Контекст main

Уменьшает разрешение таймеров времени в рабочих процессах, за счёт чего уменьшается число системных вызовов *gettimeofday()*. По умолчанию *gettimeofday()* вызывается после каждой операции получения событий из ядра. При уменьшении разрешения *gettimeofday()* вызывается только один раз за указанный интервал.

Пример использования:

```
timer_resolution 100ms;
```

Внутренняя реализация интервала зависит от используемого метода:

- фильтр *EVFILT_TIMER* при использовании kqueue;
- *timer_create()* при использовании eventport;
- *setitimer()* во всех остальных случаях.

use

Синтаксис use *метод*;

Умолчание —

Контекст events

Задаёт *метод*, используемый для обработки соединений. Обычно нет необходимости задавать его явно, поскольку по умолчанию Angie выбирает наиболее эффективный метод.

user

Синтаксис user *пользователь* [*группа*];

Умолчание user nobody nobody;

Контекст main

Задаёт пользователя и группу, с правами которого будут работать рабочие процессы. Если группа не задана, то используется группа, имя которой совпадает с именем пользователя.

worker_aio_requests

Синтаксис worker_aio_requests *число*;

Умолчание worker_aio_requests 32;

Контекст events

При использовании aio совместно с методом обработки соединений epoll, задаёт максимальное число ожидающих обработки операций асинхронного ввода-вывода для одного рабочего процесса.

worker_connections

Синтаксис worker_connections *число*;

Умолчание worker_connections 512;

Контекст events

Задаёт максимальное число соединений, которые одновременно может открыть рабочий процесс.

Следует иметь в виду, что в это число входят все соединения (в том числе, например, соединения с проксируемыми серверами), а не только соединения с клиентами. Стоит также учитывать, что фактическое число одновременных соединений не может превышать действующего ограничения на максимальное число открытых файлов, которое можно изменить с помощью worker_rlimit_nofile.

worker_cpu_affinity

Синтаксис worker_cpu_affinity *маска_CPU* …;
worker_cpu_affinity auto [*маска_CPU*];

Умолчание —

Контекст main

Привязывает рабочие процессы к группам процессоров. Каждая группа процессоров задаётся битовой маской разрешённых к использованию процессоров. Для каждого рабочего процесса должна быть задана отдельная группа. По умолчанию рабочие процессы не привязаны к конкретным процессорам.

Например,

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

привязывает каждый рабочий процесс к отдельному процессору, тогда как

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

привязывает первый рабочий процесс к CPU0/CPU2, а второй — к CPU1/CPU3. Второй пример пригоден для hyper-threading.

Специальное значение *auto* позволяет автоматически привязать рабочие процессы к доступным процессорам:

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

С помощью необязательной маски можно ограничить процессоры, доступные для автоматической привязки:

```
worker_cpu_affinity auto 01010101;
```

worker_priority

Синтаксис worker_priority *число*;

Умолчание worker_priority 0;

Контекст main

Задаёт приоритет планирования рабочих процессов подобно тому, как это делается командой *nice*: отрицательное *число* означает более высокий приоритет. Диапазон возможных значений, как правило, варьируется от -20 до 20.

Пример использования:

```
worker_priority -10;
```

worker_processes

Синтаксис worker_processes *число* | auto;

Умолчание worker_processes 1;

Контекст main

Задаёт число рабочих процессов.

Оптимальное значение зависит от множества факторов, включая (но не ограничиваясь ими) число процессорных ядер, число жёстких дисков с данными и картину нагрузок. Если затрудняетесь в выборе правильного значения, можно начать с установки его равным числу процессорных ядер (значение *auto* пытается определить его автоматически).

worker_rlimit_core

Синтаксис worker_rlimit_core *размер*;

Умолчание —

Контекст main

Изменяет ограничение на наибольший размер core-файла (*RLIMIT_CORE*) для рабочих процессов. Используется для увеличения ограничения без перезапуска главного процесса.

worker_rlimit_nofile

Синтаксис worker_rlimit_nofile *число*;

Умолчание —

Контекст main

Изменяет ограничение на максимальное число открытых файлов (*RLIMIT_NOFILE*) для рабочих процессов. Используется для увеличения ограничения без перезапуска главного процесса.

worker_shutdown_timeout

Синтаксис worker_shutdown_timeout *время*;

Умолчание —

Контекст main

Задаёт таймаут в секундах для плавного завершения рабочих процессов. По истечении указанного времени Angie попытается закрыть все открытые соединения для ускорения завершения.

working_directory

Синтаксис working_directory *каталог*;

Умолчание —

Контекст main

Задаёт каталог, который будет текущим для рабочего процесса. Основное применение — запись core-файла, в этом случае рабочий процесс должен иметь права на запись в этот каталог.

5.3 Модули http

Модуль http_access

Модуль позволяет ограничить доступ для определённых адресов клиентов.

Ограничить доступ можно также по паролю или по результату подзапроса. Одновременное ограничение доступа по адресу и паролю управляется директивой satisfy.

Пример конфигурации

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

Правила проверяются в порядке их записи до первого соответствия. В данном примере доступ разрешён только для IPv4-сетей 10.1.1.0/16 и 192.168.1.0/24, кроме адреса 192.168.1.1, и для IPv6-сети 2001:0db8::/32. Если правил много, то лучше воспользоваться переменными модуля Модуль http_geo.

Директивы

allow

Синтаксис allow *адрес* | *CIDR* | unix: | all;

Умолчание —

Контекст http, server, location, limit_except

Разрешает доступ для указанной сети или адреса. Если указано специальное значение *unix:*, разрешает доступ для всех UNIX-сокетов.

deny

Синтаксис deny *адрес* | *CIDR* | unix: | all;

Умолчание —

Контекст http, server, location, limit_except

Запрещает доступ для указанной сети или адреса. Если указано специальное значение *unix:*, запрещает доступ для всех UNIX-сокетов.

Модуль http_addition

Фильтр, добавляющий текст до и после ответа.

По умолчанию этот модуль не собирается, его сборку необходимо разрешить с помощью конфигурационного параметра --with-http_addition_module.

Пример конфигурации

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

Директивы

add_before_body

Синтаксис add_before_body uri;

Умолчание —

Контекст http, server, location

Добавляет перед телом ответа текст, выдаваемый в результате работы заданного подзапроса. Пустая строка ("") в качестве параметра отменяет добавление, унаследованное с предыдущего уровня конфигурации.

add_after_body

Синтаксис add_after_body uri;

Умолчание —

Контекст http, server, location

Добавляет после тела ответа текст, выдаваемый в результате работы заданного подзапроса. Пустая строка ("") в качестве параметра отменяет добавление, унаследованное с предыдущего уровня конфигурации.

addition_types

Синтаксис addition_types mime-*тип* …;

Умолчание addition_types text/html;

Контекст http, server, location

Разрешает добавлять текст в ответах с указанными MIME-типами в дополнение к “text/html”. Специальное значение “*” соответствует любому MIME-типу.

Модуль http_auth_basic

Позволяет ограничить доступ к ресурсам с проверкой имени и пароля пользователя по протоколу “HTTP Basic Authentication”.

Ограничить доступ можно также по адресу или по результату подзапроса. Одновременное ограничение доступа по адресу и паролю управляется директивой satisfy.

Пример конфигурации

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

```
}
```

Директивы

auth_basic

Синтаксис auth_basic *строка* | off;

Умолчание auth_basic off;

Контекст http, server, location, limit_except

Включает проверку имени и пароля пользователя по протоколу “HTTP Basic Authentication”. Заданный параметр используется в качестве *realm*. В значении параметра допустимо использование переменных.

off	отменяет действие унаследованной с предыдущего уровня конфигурации директивы *auth_basic*

auth_basic_user_file

Синтаксис auth_basic_user_file *файл*;

Умолчание —

Контекст http, server, location, limit_except

Задаёт *файл*, в котором хранятся имена и пароли пользователей. Формат файла следующий:

```
# комментарий
имя1:пароль1
имя2:пароль2:комментарий
имя3:пароль3
```

В имени *файла* можно использовать переменные.

Поддерживаются следующие типы паролей:

- зашифрованные функцией *crypt()*; могут быть созданы с помощью утилиты “*htpasswd*” из дистрибутива HTTP-сервера Apache или команды “*openssl passwd*”;
- хэшированные с помощью алгоритма, основанного на MD5, по версии Apache (apr1); могут быть созданы теми же инструментами;
- заданные согласно синтаксису “*{схема}данные*” как описано в RFC 2307; в настоящий момент реализованы схемы *PLAIN* (в качестве примера, не следует применять), *SHA* (простое SHA-1 хэширование, не следует применять) и *SSHA* (SHA-1 хэширование с солью, используется в некоторых программах, в частности OpenLDAP и Dovecot).

Примечание

Поддержка схемы SHA была добавлена лишь для облегчения процесса миграции файлов паролей с других веб-серверов. Её не следует применять для новых паролей, т.к. используемое при этом SHA-1 хэширование без соли уязвимо к взлому при помощи радужных таблиц.

Модуль http_auth_request

Предоставляет возможность авторизации клиента, основанной на результате подзапроса. Если подзапрос возвращает код ответа 2xx, доступ разрешается. Если 401 или 403 — доступ запрещается с соответствующим кодом ошибки. Любой другой код ответа, возвращаемый подзапросом, считается ошибкой.

При ошибке 401 клиенту также передаётся заголовок “WWW-Authenticate” из ответа подзапроса.

По умолчанию этот модуль не собирается, его сборку необходимо разрешить с помощью конфигурационного параметра --with-http_auth_request_module.

Модуль может быть скомбинирован с другими модулями доступа, такими как http_access и auth_basic с помощью директивы satisfy.

Пример конфигурации

```
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;
}
```

Директивы

auth_request

Синтаксис auth_request uri | off;

Умолчание auth_request off;

Контекст http, server, location

Включает авторизацию, основанную на результате выполнения подзапроса, и задаёт URI, на который будет отправлен подзапрос.

auth_request_set

Синтаксис auth_request_set *$переменная значение*;

Умолчание —

Контекст http, server, location

Устанавливает переменную в запросе в заданное значение после завершения запроса авторизации. Значение может содержать переменные из запроса авторизации, например, *$upstream_http_**.

Модуль http_autoindex

Обслуживает запросы, оканчивающиеся слэшом (‘/’), и выдаёт листинг каталога. Обычно запрос попадает к модулю http_autoindex, когда модуль http_index не нашёл индексный файл.

Пример конфигурации

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

Директивы

autoindex

Синтаксис autoindex on | off;

Умолчание autoindex off;

Контекст http, server, location

Разрешает или запрещает вывод листинга каталога.

autoindex_exact_size

Синтаксис autoindex_exact_size on | off;

Умолчание autoindex_exact_size on;

Контекст http, server, location

Для формата HTML определяет, как выводить размеры файлов в листинге каталога: точно или округляя до килобайт, мегабайт и гигабайт.

autoindex_format

Синтаксис autoindex_format html | xml | json | jsonp;

Умолчание autoindex_format html;

Контекст http, server, location

Задаёт формат вывода листинга каталога.

При использовании формата JSONP имя callback-функции задаётся в аргументе запроса callback. Если аргумент отсутствует или имеет пустое значение, то используется формат JSON.

Вывод в формате XML может быть преобразован при помощи модуля http_xslt.

autoindex_localtime

Синтаксис autoindex_localtime on | off;

Умолчание autoindex_localtime off;

Контекст http, server, location

Для формата HTML определяет, в какой временной зоне выводить время в листинге каталога: в локальной или в UTC.

Модуль http_browser

Создаёт переменные, значения которых зависят от значения поля “User-Agent” в заголовке запроса.

Переменные

$modern_browser

равна значению, заданному директивой modern_browser_value, если браузер опознан как современный;

$ancient_browser

равна значению, заданному директивой ancient_browser_value, если браузер опознан как устаревший;

$msie

равна “1”, если браузер опознан как MSIE любой версии.

Пример конфигурации

Выбор индексного файла:

```
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;
```

Перенаправление для старых браузеров:

```
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;
}
```

Директивы

ancient_browser

Синтаксис ancient_browser *строка* …;

Умолчание —

Контекст http, server, location

Задаёт подстроки, при нахождении которых в поле “User-Agent” заголовка запроса браузер считается устаревшим. Специальная строка “netscape4” соответствует регулярному выражению “^Mozilla/[1-4]”.

ancient_browser_value

Синтаксис ancient_browser_value *строка*;

Умолчание ancient_browser_value 1;

Контекст http, server, location

Задаёт значение для переменных $ancient_browser.

modern_browser

Синтаксис modern_browser *браузер версия*; modern_browser unlisted;

Умолчание —

Контекст http, server, location

Задаёт версию браузера, начиная с которой он считается современным. В качестве браузера можно задать *msie*, *gecko* (браузеры, созданные на основе Mozilla), *opera*, *safari* или *konqueror*.

Версии можно задать в форматах X, X.X, X.X.X или X.X.X.X. Максимальные значения для каждого из форматов соответственно — 4000, 4000.99, 4000.99.99 и 4000.99.99.99.

Специальное значение unlisted указывает считать современным браузер, не описанный директивами *modern_browser* и ancient_browser. В противном случае неперечисленный браузер будет считаться устаревшим. Если в заголовке запроса нет поля “User-Agent”, то браузер считается неперечисленным.

modern_browser_value

Синтаксис modern_browser_value *строка*;

Умолчание modern_browser_value 1;

Контекст http, server, location

Задаёт значение для переменных $modern_browser.

Модуль http_charset

Добавляет указанную кодировку в поле “Content-Type” заголовка ответа. Кроме того, модуль может перекодировать данные из одной кодировки в другую с некоторыми ограничениями:

- перекодирование осуществляется только в одну сторону — от сервера к клиенту,
- перекодироваться могут только однобайтные кодировки
- или однобайтные кодировки в UTF-8 и обратно.

Пример конфигурации

```
include        conf/koi-win;

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

Директивы

charset

Синтаксис set_real_ip_from *кодировка* off;

Умолчание charset off;

Контекст http, server, location, if в location

Добавляет указанную кодировку в поле “Content-Type” заголовка ответа. Если эта кодировка отличается от указанной в директиве source_charset, то выполняется перекодирование.

Параметр off отменяет добавление кодировки в поле “Content-Type” заголовка ответа.

Кодировка может быть задана с помощью переменной:

```
charset $charset;
```

В этом случае необходимо, чтобы все возможные значения переменной присутствовали хотя бы один раз в любом месте конфигурации в виде директив charset_map, *charset* или source_charset. Для кодировок *utf-8*, *windows-1251* и *koi8-r* для этого достаточно включить в конфигурацию файлы *conf/koi-win*, *conf/koi-utf* и *conf/win-utf*. Для других кодировок можно просто сделать фиктивную таблицу перекодировки, например:

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

Кроме того, кодировка может быть задана в поле “X-Accel-Charset” заголовка ответа. Эту возможность можно запретить с помощью директив proxy_ignore_headers, fastcgi_ignore_headers, uwsgi_ignore_headers, scgi_ignore_headers и grpc_ignore_headers.

charset_map

Синтаксис charset_map *кодировка1 кодировка2* { … }

Умолчание —

Контекст http

Описывает таблицу перекодирования из одной кодировки в другую. Таблица для обратного перекодирования строится на основании тех же данных. Коды символов задаются в шестнадцатеричном виде. Неописанные символы в пределах 80-FF заменяются на “?”. При перекодировании из UTF-8 символы, отсутствующие в однобайтной кодировке, заменяются на “&#XXXX;”.

Пример:

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

При описании таблицы перекодирования в UTF-8, коды кодировки UTF-8 должны быть указаны во второй колонке, например:

```
charset_map koi8-r utf-8 {
    C0 D18E ; # small yu
    C1 D0B0 ; # small a
    C2 D0B1 ; # small b
    C3 D186 ; # small ts
}
```

Полные таблицы преобразования из *koi8-r* в *windows-1251* и из *koi8-r* и *windows-1251* в *utf-8* входят в дистрибутив и находятся в файлах *conf/koi-win*, *conf/koi-utf* и *conf/win-utf*.

charset_types

Синтаксис charset_types mime-*тип* …;

Умолчание charset_types text/html text/xml text/plain text/vnd.wap.wml application/javascript application/rss+xml;

Контекст http, server, location

Разрешает работу модуля в ответах с указанными MIME-типами в дополнение к “*text/html*”. Специальное значение “*” соответствует любому MIME-типу.

override_charset

Синтаксис override_charset on | off;

Умолчание override_charset off;

Контекст http, server, location, if в location

Определяет, выполнять ли перекодирование для ответов, полученных от проксированного сервера или от FastCGI/uwsgi/SCGI/gRPC-сервера, если в ответах уже указана кодировка в поле “Content-Type” заголовка ответа. Если перекодирование

разрешено, то в качестве исходной кодировки используется кодировка, указанная в полученном ответе.

Примечание

Если ответ был получен в подзапросе, то, независимо от значения директивы *override_charset*, всегда выполняется перекодирование из кодировки ответа в кодировку основного запроса.

source_charset

Синтаксис source_charset *кодировка*;

Умолчание —

Контекст http, server, location, if в location

Задаёт исходную кодировку ответа. Если эта кодировка отличается от указанной в директиве charset, то выполняется перекодирование.

Модуль http_core

Директивы

absolute_redirect

Синтаксис absolute_redirect on | off;

Умолчание absolute_redirect on;

Контекст http, server, location

Если запрещено, то перенаправления, выдаваемые Angie PRO, будут относительными.

См. также директивы server_name_in_redirect и port_in_redirect.

aio

Синтаксис aio on | off | threads[= *pool*];

Умолчание aio off;

Контекст http, server, location

Разрешает или запрещает использование файлового асинхронного ввода-вывода (AIO) в Linux:

```
location /video/ {
  aio            on;
  output_buffers 1 64k;
}
```

В Linux AIO можно использовать только начиная с версии ядра 2.6.22. Кроме того, необходимо также дополнительно включить directio, иначе чтение будет блокирующимся:

```
location /video/ {
  aio            on;
  directio       512;
```

```
  output_buffers 1 128k;
}
```

В Linux directio можно использовать только для чтения блоков, выравненных на границу 512 байт (или 4К для XFS). Невыравненный конец файла будет читаться блокированно. То же относится к запросам с указанием диапазона запрашиваемых байт (byte-range requests) и к запросам FLV не с начала файла: чтение невыравненных начала и конца ответа будет блокирующимся.

При одновременном включении AIO и sendfile в Linux для файлов, размер которых больше либо равен указанному в директиве directio, будет использоваться AIO, а для файлов меньшего размера или при выключенном directio — sendfile:

```
location /video/ {
  sendfile       on;
  aio            on;
  directio       8m;
}
```

Кроме того, читать и отправлять файлы можно в многопоточном режиме, не блокируя при этом рабочий процесс:

```
location /video/ {
  sendfile       on;
  aio            threads;
}
```

Операции чтения или отправки файлов будут обрабатываться потоками из указанного пула. Если пул потоков не задан явно, используется пул с именем *default*. Имя пула может быть задано при помощи переменных:

aio threads=pool$disk**;**

По умолчанию поддержка многопоточности выключена, её сборку следует разрешить с помощью конфигурационного параметра *–with-threads*. В настоящий момент многопоточность совместима только с методами epoll, kqueue и eventport. Отправка файлов в многопоточном режиме поддерживается только на Linux.

См. также директиву sendfile.

aio_write

Синтаксис aio_write on | off;

Умолчание aio_write off;

Контекст http, server, location

При включённом aio разрешает его использование для записи файлов. В настоящий момент это работает только при использовании *aio threads* и ограничено записью временных файлов с данными, полученными от проксируемых серверов.

alias

Синтаксис alias *путь*;

Умолчание —

Контекст location

Задаёт замену для указанного *location*'а. Например, при такой конфигурации:

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

на запрос /i/top.gif будет отдан файл */data/w3/images/top.gif*.

В значении параметра *путь* можно использовать переменные, кроме $document_root и $realpath_root.

Если *alias* используется внутри *location*'а, заданного регулярным выражением, то регулярное выражение должно содержать выделения, а сам *alias* — ссылки на эти выделения, например:

```
location ~ ^/users/(.+\.(?:gif|jpe?g|png))$ {
  alias /data/w3/images/$1;
}
```

Если *location* и последняя часть значения директивы совпадают:

```
location /images/ {
  alias /data/w3/images/;
}
```

то лучше воспользоваться директивой root:

```
location /images/ {
  root /data/w3;
}
```

auth_delay

Синтаксис auth_delay *время*;

Умолчание auth_delay 0s;

Контекст http, server, location

Задерживает обработку неавторизованных запросов с кодом ответа 401 для предотвращения атак по времени в случае ограничения доступа по паролю или по результату подзапроса.

chunked_transfer_encoding

Синтаксис chunked_transfer_encoding on | off;

Умолчание chunked_transfer_encoding on;

Контекст

http, server, location

Позволяет запретить формат передачи данных частями (chunked transfer encoding) в HTTP/1.1. Это может понадобиться при использовании программ, не поддерживающих chunked encoding, несмотря на требования стандарта.

client_body_buffer_size

Синтаксис client_body_buffer_size *размер;*

Умолчание client_body_buffer_size 8k|16k;

Контекст http, server, location

Задаёт размер буфера для чтения тела запроса клиента. Если тело запроса больше заданного буфера, то всё тело запроса или только его часть записывается во временный файл. По умолчанию размер одного буфера равен двум размерам страницы. На *x86*, других 32-битных платформах и *x86-64* это 8K. На других 64-битных платформах это обычно 16K.

client_body_in_file_only

Синтаксис client_body_in_file_only on | clean | off;

Умолчание client_body_in_file_only off;

Контекст http, server, location

Определяет, сохранять ли всё тело запроса клиента в файл. Директиву можно использовать для отладки и при использовании переменной $request_body_file или метода $r->request_body_file модуля http_perl.

on	временные файлы по окончании обработки запроса не удаляются
clean	разрешает удалять временные файлы, оставшиеся по окончании обработки запроса

client_body_in_single_buffer

Синтаксис client_body_in_single_buffer on | off;

Умолчание client_body_in_single_buffer off;

Контекст http, server, location

Определяет, сохранять ли всё тело запроса клиента в одном буфере. Директива рекомендуется при использовании переменной $request_body для уменьшения требуемого числа операций копирования.

client_body_temp_path

Синтаксис client_body_temp_path *путь* [*уровень1* [*уровень2* [*уровень3*]]];

Умолчание client_body_temp_path client_body_temp;

Контекст http, server, location

Задаёт каталог для хранения временных файлов с телами запросов клиентов. В каталоге может использоваться иерархия подкаталогов до трёх уровней. Например, при такой конфигурации

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

путь к временному файлу будет следующего вида:

```
/spool/angie/client_temp/7/45/00000123457
```

client_body_timeout

Синтаксис client_body_timeout *время;*

Умолчание client_body_timeout 60s;

Контекст http, server, location

Задаёт таймаут при чтении тела запроса клиента. Таймаут устанавливается не на всю передачу тела запроса, а только между двумя последовательными операциями чтения. Если по истечении этого времени клиент ничего не передаст, обработка запроса прекращается с ошибкой 408 (Request Time-out).

client_header_buffer_size

Синтаксис client_header_buffer_size *размер;*

Умолчание client_header_buffer_size 1k;

Контекст http, server

Задаёт размер буфера для чтения заголовка запроса клиента. Для большинства запросов достаточно буфера размером в 1K байт. Однако если в запросе есть длинные cookies, или же запрос пришёл от WAP-клиента, то он может не поместиться в 1K. Поэтому, если строка запроса или поле заголовка запроса не помещаются полностью в этот буфер, то выделяются буферы большего размера, задаваемые директивой large_client_header_buffers.

Если директива указана на уровне server, то может использоваться значение из сервера по умолчанию. Подробнее см. в разделе Выбор виртуального сервера.

client_header_timeout

Синтаксис client_header_timeout *время;*

Умолчание client_header_timeout 60s;

Контекст http, server

Задаёт таймаут при чтении заголовка запроса клиента. Если по истечении этого времени клиент не передаст полностью заголовок, обработка запроса прекращается с ошибкой 408 (Request Time-out).

client_max_body_size

Синтаксис client_max_body_size *размер;*

Умолчание client_max_body_size 1m;

Контекст http, server, location

Задаёт максимально допустимый размер тела запроса клиента. Если размер больше заданного, то клиенту возвращается ошибка 413 (Request Entity Too Large). Следует иметь в виду, что браузеры не умеют корректно показывать эту ошибку.

0	отключает проверку размера тела запроса клиента

connection_pool_size

Синтаксис connection_pool_size *размер;*

Умолчание connection_pool_size 256 | 512;

Контекст http, server, location

Позволяет производить точную настройку выделения памяти под конкретные соединения. Эта директива не оказывает существенного влияния на производительность, и её не следует использовать.
По умолчанию:

256 байт	на 32-битных платформах
512 байт	на 64-битных платформах

default_type

Синтаксис default_type mime-*тип*;

Умолчание default_type text/plain;

Контекст http, server, location

Задаёт MIME-тип ответов по умолчанию. Соответствие расширений имён файлов MIME-типу ответов задаётся с помощью директивы types.

directio

Синтаксис directio *размер* | off;

Умолчание directio off;

Контекст http, server, location

Разрешает использовать флаги *O_DIRECT* (FreeBSD, Linux), *F_NOCACHE* (macOS) или функцию *directio()* (Solaris) при чтении файлов, размер которых больше либо равен указанному. Директива автоматически запрещает использование sendfile для данного запроса. Рекомендуется использовать для больших файлов:

```
directio 4m;
```

или при использовании aio в Linux.

directio_alignment

Синтаксис directio_alignment *размер;*

Умолчание directio_alignment 512;

Контекст http, server, location

Устанавливает выравнивание для directio. В большинстве случаев достаточно 512-байтового выравнивания, однако при использовании XFS под Linux его нужно увеличить до 4K.

disable_symlinks

Синтаксис disable_symlinks off;
disable_symlinks on | if_not_owner [from = *часть*];

Умолчание disable_symlinks off;

Контекст http, server, location

Определяет, как следует поступать с символическими ссылками при открытии файлов:

off	Символические ссылки в пути допускаются и не проверяются. Это стандартное поведение.
on	Если любой компонент пути является символической ссылкой, доступ к файлу запрещается.
if_not_owner	Доступ к файлу запрещается, если любой компонент пути является символической ссылкой, а ссылка и объект, на который она ссылается, имеют разных владельцев.
from= *часть*	При проверке символических ссылок (параметры *on* и *if_not_owner*) обычно проверяются все компоненты пути. Можно не проверять символические ссылки в начальной части пути, указав дополнительно параметр *from=часть*. В этом случае символические ссылки проверяются лишь начиная с компонента пути, который следует за заданной начальной частью. Если значение не является начальной частью проверяемого пути, путь проверяется целиком, как если бы этот параметр не был указан вовсе. Если значение целиком совпадает с именем файла, символические ссылки не проверяются. В значении параметра можно использовать переменные.

Пример:

```
disable_symlinks on from=$document_root;
```

Эта директива доступна только на системах, в которых есть интерфейсы *openat()* и *fstatat()*. К таким системам относятся современные версии FreeBSD, Linux и Solaris.

Предупреждение

Параметры *on* и *if_not_owner* требуют дополнительных затрат на обработку.

На системах, не поддерживающих операцию открытия каталогов только для поиска, для использования этих параметров требуется, чтобы рабочие процессы имели право читать все проверяемые каталоги.

Примечание

Модули http_autoindex, http_random_index и http_dav в настоящий момент игнорируют эту директиву.

error_page

Синтаксис error_page *код* … [=[*ответ*]] uri;

Умолчание —

Контекст http, server, location, if в location

Задаёт URI, который будет показываться для указанных ошибок. В значении *uri* можно использовать переменные.

Пример:

```
error_page 404             /404.html;
error_page 500 502 503 504 /50x.html;
```

При этом делается внутреннее перенаправление на указанный *uri*, а метод запроса клиента меняется на “GET” (для всех методов, отличных от “GET” и “HEAD”).

Кроме того, можно поменять код ответа на другой, используя синтаксис вида *=ответ*, например:

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

Если ошибочный ответ обрабатывается проксированным сервером или FastCGI/uwsgi/SCGI/gRPC-сервером, и этот сервер может вернуть разные коды ответов, например, 200, 302, 401 или 404, то можно выдавать возвращаемый им код:

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

Если при внутреннем перенаправлении не нужно менять URI и метод, то можно передать обработку ошибки в именованный *location*:

```
location / {
  error_page 404 = @fallback;
}

location @fallback {
  proxy_pass http://backend;
}
```

Примечание

Если при обработке uri происходит ошибка, клиенту возвращается ответ с кодом последней случившейся ошибки.

Также существует возможность использовать перенаправления URL для обработки ошибок:

```
error_page 403      http://example.com/forbidden.html;
error_page 404 =301 http://example.com/notfound.html;
```

В этом случае по умолчанию клиенту возвращается код ответа 302. Его можно изменить только на один из кодов ответа, относящихся к перенаправлениям (301, 302, 303, 307 и 308).

etag

Синтаксис etag on | off;

Умолчание etag on;

Контекст http, server, location

Разрешает или запрещает автоматическую генерацию поля “ETag“ заголовка ответа для статических ресурсов.

http

Синтаксис http { … }

Умолчание —

Контекст main

Предоставляет контекст конфигурационного файла, в котором указываются директивы HTTP-сервера.

if_modified_since

Синтаксис if_modified_since off | exact | before;

Умолчание if_modified_since exact;

Контекст http, server, location

Определяет, как сравнивать время модификации ответа с временем в поле “If-Modified-Since“ заголовка запроса:

off	ответ всегда считается изменившимся
exact	точное совпадение
before	время модификации ответа меньше или равно времени, заданному в поле “If-Modified-Since“ заголовка запроса

ignore_invalid_headers

Синтаксис ignore_invalid_headers on | off;

Умолчание ignore_invalid_headers on;

Контекст http, server

Если включено, Angie игнорирует поля заголовка с недопустимыми именами. Допустимыми считаются имена, состоящие из английских букв, цифр, дефисов и возможно знаков подчёркивания (последнее контролируется директивой underscores_in_headers).

Если директива указана на уровне server, то может использоваться значение из сервера по умолчанию.

internal

Синтаксис internal;

Умолчание —

Контекст location

Указывает, что *location* может использоваться только для внутренних запросов. Для внешних запросов клиенту будет возвращаться ошибка 404 (Not Found). Внутренними запросами являются:

- запросы, перенаправленные директивами error_page, index, random_index и try_files;
- запросы, перенаправленные с помощью поля “X-Accel-Redirect“ заголовка ответа вышестоящего сервера;
- подзапросы, формируемые командой *include virtual* модуля Модуль http_ssi, директивами модуля http_addition, а также директивами auth_request и mirror;
- запросы, изменённые директивой rewrite.

Пример:

```
error_page 404 /404.html;

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

Примечание

Для предотвращения зацикливания, которое может возникнуть при использовании некорректных конфигураций, количество внутренних перенаправлений ограничено десятью. По достижении этого ограничения будет возвращена ошибка 500 (Internal Server Error). В таком случае в лог-файле ошибок можно увидеть сообщение *rewrite or internal redirection cycle*.

keepalive_disable

Синтаксис keepalive_disable none | *браузер* …;

Умолчание keepalive_disable msie6;

Контекст http, server, location

Запрещает *keep-alive* соединения с некорректно ведущими себя браузерами. Параметры *браузер* указывают, на какие браузеры это распространяется.

none	разрешает *keep-alive* соединения со всеми браузерами
msie6	запрещает *keep-alive* соединения со старыми версиями MSIE после получения запроса POST
safari	запрещает *keep-alive* соединения с Safari и подобными им браузерами на macOS и подобных ей ОС

keepalive_requests

Синтаксис keepalive_requests *число;*

Умолчание keepalive_requests 1000;

Контекст http, server, location

Задаёт максимальное число запросов, которые можно сделать по одному *keep-alive* соединению. После того, как сделано максимальное число запросов, соединение закрывается.

Периодическое закрытие соединений необходимо для освобождения памяти, выделенной под конкретные соединения. Поэтому использование слишком большого максимального числа запросов может приводить к чрезмерному потреблению памяти и не рекомендуется.

keepalive_time

Синтаксис keepalive_time *время;*

Умолчание keepalive_time 1h;

Контекст http, server, location

Ограничивает максимальное время, в течение которого могут обрабатываться запросы в рамках *keep-alive* соединения. По достижении заданного времени соединение закрывается после обработки очередного запроса.

keepalive_timeout

Синтаксис keepalive_timeout *таймаут* [*заголовок_таймаута*]

Умолчание keepalive_timeout 75s;

Контекст http, server, location

таймаут	задаёт время, в течение которого *keep-alive* соединение с клиентом не будет закрыто со стороны сервера
0	запрещает keep-alive соединения с клиентами

Второй, *необязательный*, параметр задаёт значение в поле “Keep-Alive: timeout=время“ заголовка ответа. Два параметра могут отличаться друг от друга.

Поле “Keep-Alive: timeout=время“ заголовка понимают Mozilla и Konqueror. MSIE сам закрывает keep-alive соединение примерно через 60 секунд.

large_client_header_buffers

Синтаксис large_client_header_buffers *количество размер*;

Умолчание large_client_header_buffers 4 8k;

Контекст http, server

Задаёт максимальное число и размер буферов для чтения большого заголовка запроса клиента. Строка запроса не должна превышать размера одного буфера, иначе клиенту возвращается ошибка 414 (Request-URI Too Large). Поле заголовка запроса также не должно превышать размера одного буфера, иначе клиенту возвращается ошибка 400 (Bad Request). Буферы выделяются только по мере необходимости. По умолчанию размер одного буфера равен 8K байт. Если по окончании обработки запроса соединение переходит в состояние *keep-alive*, эти буферы освобождаются.

Если директива указана на уровне server, то может использоваться значение из сервера по умолчанию.

limit_except

Синтаксис limit_except *метод1* [*метод2*…] { … };

Умолчание —

Контекст location

Ограничивает HTTP-методы, доступные внутри location. Параметр метод может быть одним из *GET*, *HEAD*, *POST*, *PUT*, *DELETE*, *MKCOL*, *COPY*, *MOVE*, *OPTIONS*, *PROPFIND*, *PROPPATCH*, *LOCK*, *UNLOCK* или *PATCH*. Если разрешён метод *GET*, то метод *HEAD* также будет разрешён. Доступ к остальным методам может быть ограничен при помощи директив модулей http_access и http_auth_basic.

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

Примечание

Ограничение в примере действует для всех методов, **кроме** *GET* и *HEAD*.

limit_rate

Синтаксис limit_rate *скорость;*

Умолчание limit_rate 0;

Контекст http, server, location, if в location

Ограничивает скорость передачи ответа клиенту. Скорость задаётся в байтах в секунду. Значение *0* отключает ограничение скорости. Ограничение устанавливается на запрос, поэтому, если клиент одновременно откроет два соединения, суммарная скорость будет вдвое выше заданного ограничения.

В значении параметра можно использовать переменные. Это может быть полезно в случаях, когда скорость нужно ограничивать в зависимости от какого-либо условия:

```
map $slow $rate {
  1     4k;
  2     8k;
}

limit_rate $rate;
```

Ограничение скорости можно также задать в переменной $limit_rate, однако использовать данный метод не рекомендуется:

```
server {

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

}
```

Кроме того, ограничение скорости может быть задано в поле “X-Accel-Limit-Rate“ заголовка ответа проксированного сервера. Эту возможность можно запретить с помощью директив proxy_ignore_headers, fastcgi_ignore_headers, uwsgi_ignore_headers и scgi_ignore_headers.

limit_rate_after

Синтаксис limit_rate_after *размер;*

Умолчание limit_rate_after 0;

Контекст http, server, location, if в location

Задаёт начальный объём данных, после передачи которого начинает ограничиваться скорость передачи ответа клиенту. В значении параметра можно использовать переменные.

Пример:

```
location /flv/ {
 flv;
 limit_rate_after 500k;
 limit_rate       50k;
}
```

lingering_close

Синтаксис lingering_close on | always | off;

Умолчание lingering_close on;

Контекст http, server, location

Управляет закрытием соединений с клиентами.

on	Angie будет ждать и обрабатывать дополнительные данные,

	поступающие от клиента, перед полным закрытием соединения, но только если эвристика указывает на то, что клиент может ещё послать данные.
always	Angie всегда будет ждать и обрабатывать дополнительные данные, поступающие от клиента.
off	Angie не будет ждать поступления дополнительных данных и сразу же закроет соединение. Это поведение нарушает протокол и поэтому не должно использоваться без необходимости.

Для управления закрытием HTTP/2-соединений директива должна быть задана на уровне server.

lingering_time

Синтаксис lingering_time *время;*

Умолчание lingering_time 30s;

Контекст http, server, location

Если действует lingering_close, эта директива задаёт максимальное время, в течение которого Angie будет обрабатывать (читать и игнорировать) дополнительные данные, поступающие от клиента. По прошествии этого времени соединение будет закрыто, даже если будут ещё данные.

lingering_timeout

Синтаксис lingering_timeout *время;*

Умолчание lingering_timeout 5s;

Контекст http, server, location

Если действует lingering_close, эта директива задаёт максимальное время ожидания поступления дополнительных данных от клиента. Если в течение этого времени данные не были получены, соединение закрывается. В противном случае данные читаются и игнорируются, и Angie снова ждёт поступления данных. Цикл “ждать-читать-игнорировать” повторяется, но не дольше чем задано директивой lingering_time.

listen

Синтаксис listen *адрес*[:*порт*] [default_server] [ssl] [http2 | spdy] [proxy_protocol] [setfib = *число*] [fastopen = *число*] [backlog = *число*] [rcvbuf = *размер*] [sndbuf = *размер*] [accept_filter = *фильтр*] [deferred] [bind] [ipv6only=on|off] [reuseport] [so_keepalive=on|off|[keepidle]:[keepintvl]:[keepcnt]];

listen *порт* [default_server] [ssl] [http2 | spdy] [proxy_protocol] [setfib = *число*] [fastopen = *число*] [backlog = *число*] [rcvbuf = *размер*] [sndbuf = *размер*] [accept_filter = *фильтр*] [deferred] [bind] [ipv6only=on|off] [reuseport] [so_keepalive=on|off|[keepidle]:[keepintvl]:[keepcnt]];

listen unix:*путь* [default_server] [ssl] [http2 | spdy] [proxy_protocol] [backlog = *число*] [rcvbuf =

размер] [sndbuf = *размер*] [accept_filter = *фильтр*] [deferred] [bind] [so_keepalive=on|off|[keepidle]:[keepintvl]:[keepcnt]];

Умолчание listen *:80 | *:8000;

Контекст server

Задаёт *адрес* и *порт* для слушающего сокета или путь для UNIX-сокета, на которых сервер будет принимать запросы. Можно указать *адрес* и *порт*, либо только *адрес* или только *порт*. Кроме того, *адрес* может быть именем хоста, например:

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

IPv6-адреса задаются в квадратных скобках:

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

UNIX-сокеты задаются при помощи префикса *unix:*:

```
listen unix:/var/run/angie.sock;
```

Если указан только *адрес*, то используется порт *80*.

Если директива не указана, то используется либо **:80*, если Angie работает с привилегиями суперпользователя, либо **:8000*.

default_server	сервер, в котором указан этот параметр, будет сервером по умолчанию для указанной пары адрес:порт (слушающий сокет). Если же директив с параметром default_server нет, сервером по умолчанию для слущающего сокета будет первый встречающийся в конфигурации сервер, в котором описан этот слушающий сокет.
ssl	указывает на то, что все соединения, принимаемые на данном слушающем сокете, должны работать в режиме SSL. Это позволяет задать компактную конфигурацию для сервера, работающего сразу в двух режимах — HTTP и HTTPS.
http2	позволяет принимать на слушающем сокете HTTP/2-соединения. Обычно, чтобы это работало, следует также указать параметр ssl, однако Angie можно также настроить и на приём HTTP/2-соединений без SSL.
spdy	позволяет принимать на слушающем сокете SPDY-соединения. Обычно, чтобы это работало, следует также указать параметр ssl, однако Angie можно также настроить и на приём SPDY-соединений без SSL.

proxy_protocol	указывает на то, что все соединения, принимаемые на данном слушающем сокете, должны использовать протокол PROXY.

В директиве *listen* можно также указать несколько дополнительных параметров, специфичных для связанных с сокетами системных вызовов. Эти параметры можно задать в любой директиве *listen*, но только один раз для слушающего сокета.

setfib= число	задаёт таблицу маршрутизации, FIB (параметр SO_SETFIB) для слушающего сокета. В настоящий момент это работает только на FreeBSD.
fastopen= число	включает “TCP Fast Open” для слушающего сокета и ограничивает максимальную длину очереди соединений, которые ещё не завершили процесс three-way handshake.

Примечание

Не включайте “TCP Fast Open”, не убедившись, что сервер может адекватно обрабатывать многократное получение одного и того же SYN-пакета с данными.

backlog= число	задаёт параметр backlog в вызове listen(), который ограничивает максимальный размер очереди ожидающих приёма соединений. По умолчанию backlog устанавливается равным -1 для FreeBSD, DragonFly BSD и macOS, и 511 для других платформ.
rcvbuf= размер	задаёт размер буфера приёма (параметр SO_RCVBUF) для слушающего сокета.
sndbuf= размер	задаёт размер буфера передачи (параметр SO_SNDBUF) для слушающего сокета.
accept_filter= фильтр	задаёт название accept-фильтра (параметр SO_ACCEPTFILTER) для слушающего сокета, который включается для фильтрации входящих соединений перед передачей их в accept(). Работает только на FreeBSD и NetBSD 5.0+. Можно использовать два фильтра: dataready и httpready.
deferred	указывает использовать отложенный accept() (параметр TCP_DEFER_ACCEPT сокета) на Linux.
bind	указывает, что для данного слушающего сокета нужно делать bind() отдельно. Это нужно потому, что если описаны несколько директив listen с одинаковым портом, но разными адресами, и одна из директив listen слушает на всех адресах для данного порта (*:порт), то Angie сделает bind() только на *:порт. Необходимо заметить, что в этом случае для определения адреса, на который пришло соединение, делается системный вызов

	getsockname(). Если же используются параметры setfib, fastopen, backlog, rcvbuf, sndbuf, accept_filter, deferred, ipv6only, reuseport или so_keepalive, то для данной пары адрес:порт всегда делается отдельный вызов bind().
ipv6only=on | off	определяет (через параметр сокета IPV6_V6ONLY), будет ли слушающий на wildcard-адресе [::] IPv6-сокет принимать только IPv6-соединения, или же одновременно IPv6- и IPv4-соединения. По умолчанию параметр включён. Установить его можно только один раз на старте.
reuseport	указывает, что нужно создавать отдельный слушающий сокет для каждого рабочего процесса (через параметр сокета SO_REUSEPORT для Linux 3.9+ и DragonFly BSD или SO_REUSEPORT_LB для FreeBSD 12+), позволяя ядру распределять входящие соединения между рабочими процессами. В настоящий момент это работает только на Linux 3.9+, DragonFly BSD и FreeBSD 12+.

Предупреждение

Ненадлежащее использование параметра *reuseport* может быть небезопасно.

so_keepalive=on | off | [keepidle]:[keepintvl]:[keepcnt] конфигурирует для слушающего сокета поведение “TCP keepalive”.

"	если параметр опущен, для сокета будут действовать настройки операционной системы
on	для сокета включается параметр SO_KEEPALIVE
off	для сокета параметр SO_KEEPALIVE выключается

Некоторые операционные системы поддерживают настройку параметров “TCP keepalive” на уровне сокета посредством параметров *TCP_KEEPIDLE*, *TCP_KEEPINTVL* и *TCP_KEEPCNT*. На таких системах их можно сконфигурировать с помощью параметров *keepidle*, *keepintvl* и *keepcnt*. Один или два параметра могут быть опущены, в таком случае для соответствующего параметра сокета будут действовать стандартные системные настройки. Например,

```
so_keepalive=30m::10
```

установит таймаут бездействия (*TCP_KEEPIDLE*) в 30 минут, для интервала проб (*TCP_KEEPINTVL*) будет действовать стандартная системная настройка, а счётчик проб (*TCP_KEEPCNT*) будет равен 10.

Пример:

```
listen 127.0.0.1 default_server accept_filter=dataready backlog=1024;
```

location

Синтаксис location [= | ~ | ~* | ^~] *uri* { … }
location @*имя* { … }

Умолчание —

Контекст server, location

Устанавливает конфигурацию в зависимости от URI запроса.

Для сопоставления используется URI запроса в нормализованном виде, после декодирования текста, заданного в виде *%XX*, преобразования относительных элементов пути «.» и «..» в реальные и возможной замены двух и более подряд идущих слэшей на один.

location можно задать префиксной строкой или регулярным выражением.

Регулярные выражения задаются с модификатором:

~*	для поиска совпадения без учёта регистра символов
~	с учётом регистра

Чтобы найти *location*, соответствующий запросу, вначале проверяются *location*'ы, заданные префиксными строками (префиксные *location*'ы). Среди них ищется *location* с совпадающим префиксом максимальной длины и запоминается. Затем проверяются регулярные выражения, в порядке их следования в конфигурационном файле. Проверка регулярных выражений прекращается после первого же совпадения, и используется соответствующая конфигурация. Если совпадение с регулярным выражением не найдено, то используется конфигурация запомненного ранее префиксного *location*'а.

Блоки *location* могут быть вложенными, с некоторыми исключениями, о которых говорится ниже.

Для операционных систем, нечувствительных к регистру символов, таких как macOS и Cygwin, сравнение с префиксными строками производится без учёта регистра. Однако сравнение ограничено только однобайтными *locale*'ями.

Регулярные выражения могут содержать выделения, которые могут затем использоваться в других директивах.

Если у совпавшего префиксного *location*'а максимальной длины указан модификатор ^~, то регулярные выражения не проверяются.

Кроме того, с помощью модификатора = можно задать точное совпадение URI и *location*. При точном совпадении поиск сразу же прекращается. Например, если запрос / случается часто, то указав location = /, можно ускорить обработку этих запросов, так как поиск прекратится после первого же сравнения. Очевидно, что такой *location* не может иметь вложенные *location*'ы.

Проиллюстрируем вышесказанное примером:

```
location = / {
```

```
    #конфигурация A
}

location / {
    #конфигурация Б
}

location /documents/ {
    #конфигурация В
}

location ^~ /images/ {
    #конфигурация Г
}

location ~* \.(gif|jpg|jpeg)$ {
    #конфигурация Д
}
```

Для запроса / будет выбрана конфигурация А, для запроса /index.html — конфигурация Б, для запроса /documents/document.html — конфигурация В, для запроса /images/1.gif — конфигурация Г, а для запроса /documents/1.jpg — конфигурация Д.

Префикс @ задаёт *именованный location*. Такой *location* не используется при обычной обработке запросов, а предназначен только для перенаправления в него запросов. Такие *location*'ы не могут быть вложенными и не могут содержать вложенные *location*'ы.

Если *location* задан префиксной строкой со слэшом в конце и запросы обрабатываются при помощи *proxy_pass*, *fastcgi_pass*, *uwsgi_pass*, *scgi_pass*, *memcached_pass* или *grpc_pass*, происходит специальная обработка. В ответ на запрос с URI равным этой строке, но без завершающего слэша, будет возвращено постоянное перенаправление с кодом 301 на URI с добавленным в конец слэшом. Если такое поведение нежелательно, можно задать точное совпадение URI и location, например:

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

location = /user {
  proxy_pass http://login.example.com;
}
```

log_not_found

Синтаксис	log_not_found on | off;
Умолчание	log_not_found on;
Контекст	http, server, location

Разрешает или запрещает записывать в error_log ошибки о том, что файл не найден.

log_subrequest

Синтаксис log_subrequest on | off;

Умолчание log_subrequest off;

Контекст http, server, location

Разрешает или запрещает записывать в access_log подзапросы.

max_ranges

Синтаксис max_ranges *число*

Умолчание —

Контекст http, server, location

Ограничивает максимальное допустимое число диапазонов в запросах с указанием диапазона запрашиваемых байт (byte-range requests). Запросы, превышающие указанное ограничение, обрабатываются как если бы они не содержали указания диапазонов. По умолчанию число диапазонов не ограничено.

0	полностью запрещает поддержку диапазонов

merge_slashes

Синтаксис merge_slashes on | off;

Умолчание merge_slashes on;

Контекст http, server

Разрешает или запрещает преобразование URI путём замены двух и более подряд идущих слэшей (“/”) на один.

Необходимо иметь в виду, что это преобразование необходимо для корректной проверки префиксных строк и регулярных выражений. Если его не делать, то запрос //scripts/one.php не попадёт в

```
location /scripts/ { }
```

и может быть обслужен как статический файл. Поэтому он преобразуется к виду /scripts/one.php.

Запрет преобразования может понадобиться, если в URI используются имена, закодированные методом *base64*, в котором задействован символ “/”. Однако из соображений безопасности лучше избегать отключения преобразования.

Если директива указана на уровне server, то может использоваться значение из сервера по умолчанию.

msie_padding

Синтаксис msie_padding on | off;

Умолчание msie_padding on;

Контекст http, server, location

Разрешает или запрещает добавлять в ответы для MSIE со статусом больше 400 комментарий для увеличения размера ответа до 512 байт.

msie_refresh

Синтаксис msie_refresh on | off;

Умолчание msie_refresh off;

Контекст http, server, location

Разрешает или запрещает выдавать для MSIE клиентов refresh'ы вместо перенаправлений.

open_file_cache

Синтаксис open_file_cache off;
open_file_cache max = *N* [inactiv = *время*];

Умолчание open_file_cache off;

Контекст http, server, location

Задаёт кэш, в котором могут храниться:

- дескрипторы открытых файлов, информация об их размерах и времени модификации;
- информация о существовании каталогов;
- информация об ошибках поиска файла — "нет файла", "нет прав на чтение" и тому подобное.

Кэширование ошибок нужно разрешить отдельно директивой open_file_cache_errors.

max	задаёт максимальное число элементов в кэше; при переполнении кэша удаляются наименее востребованные элементы (LRU);
inactive	задаёт время, после которого элемент кэша удаляется, если к нему не было обращений в течение этого времени; по умолчанию 60 секунд;
off	запрещает кэш.

Пример:

```
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

Синтаксис open_file_cache_errors on | off;

Умолчание open_file_cache_errors off;

Контекст http, server, location

Разрешает или запрещает кэширование ошибок поиска файлов в open_file_cache.

open_file_cache_min_uses

Синтаксис open_file_cache_min_uses *число;*

Умолчание open_file_cache_min_uses 1;

Контекст http, server, location

Задаёт минимальное число обращений к файлу в течение времени, заданного параметром inactive директивы open_file_cache, необходимых для того, чтобы дескриптор файла оставался открытым в кэше.

open_file_cache_valid

Синтаксис open_file_cache_valid *время;*

Умолчание open_file_cache_valid 60s;

Контекст http, server, location

Определяет время, через которое следует проверять актуальность информации об элементе в open_file_cache.

output_buffers

Синтаксис output_buffers *число размер;*

Умолчание output_buffers 2 32k;

Контекст http, server, location

Задаёт *число* и *размер* буферов, используемых при чтении ответа с диска.

port_in_redirect

Синтаксис port_in_redirect on | off;

Умолчание port_in_redirect on;

Контекст http, server, location

Разрешает или запрещает указывать порт в абсолютных перенаправлениях, выдаваемых Angie.

Использование в перенаправлениях основного имени сервера управляется директивой server_name_in_redirect.

postpone_output

Синтаксис postpone_output *размер;*

Умолчание postpone_output 1460;

Контекст http, server, location

Если это возможно, то отправка данных клиенту будет отложена пока Angie не накопит по крайней мере указанное количество байт для отправки.

0	запрещает отложенную отправку данных

read_ahead

Синтаксис read_ahead *размер;*

Умолчание read_ahead 0;

Контекст http, server, location

Задаёт ядру размер предчтения при работе с файлами.

На Linux используется системный вызов *posix_fadvise(0, 0, 0, POSIX_FADV_SEQUENTIAL)*, поэтому параметр размер там игнорируется.

recursive_error_pages

Синтаксис recursive_error_pages on | off;

Умолчание recursive_error_pages off;

Контекст http, server, location

Разрешает или запрещает делать несколько перенаправлений через директиву error_page. Число таких перенаправлений ограничено.

request_pool_size

Синтаксис request_pool_size *размер;*

Умолчание request_pool_size 4k;

Контекст http, server

Позволяет производить точную настройку выделений памяти под конкретные запросы. Эта директива не оказывает существенного влияния на производительность, и её не следует использовать.

reset_timedout_connection

Синтаксис reset_timedout_connection on | off;

Умолчание reset_timedout_connection off;

Контекст http, server, location

Разрешает или запрещает сброс соединений по таймауту, а также при закрытии соединений с помощью нестандартного кода 444. Сброс делается следующим образом. Перед закрытием сокета для него задаётся параметр *SO_LINGER* с таймаутом *0*. После этого при закрытии сокета клиенту отсылается *TCP RST*, а вся память, связанная с этим сокетом, освобождается. Это позволяет избежать длительного нахождения уже закрытого сокета в состоянии *FIN_WAIT1* с заполненными буферами.

Примечание

keep-alive соединения по истечении таймаута закрываются обычным образом.

resolver

Синтаксис resolver *адрес* … [valid= *время*] [ipv4=on|off] [ipv6=on|off] [status_zone= *зона*];

Умолчание —

Контекст http, server, location

Задаёт серверы DNS, используемые для преобразования имён вышестоящих серверов в адреса, например:

resolver 127.0.0.1 [::1]:5353**;**

Адрес может быть указан в виде доменного имени или IP-адреса, и необязательного порта. Если порт не указан, используется порт 53. Серверы DNS опрашиваются циклически.

По умолчанию Angie кэширует ответы, используя значение TTL из ответа.

valid	необязательный параметр, позволяет переопределить срок кэширования ответа

```
resolver 127.0.0.1 [::1]:5353 valid=30s;
```

По умолчанию Angie будет искать как IPv4-, так и IPv6-адреса при преобразовании имён в адреса.

ipv4=off	запрещает поиск IPv4-адресов
ipv6=off	запрещает поиск IPv6-адресов
status_zone	*необязательный* параметр, включает сбор информации о запросах и ответах сервера DNS в указанной зоне

Примечание

Для предотвращения DNS-спуфинга рекомендуется использовать DNS-серверы в защищённой доверенной локальной сети.

resolver_timeout

Синтаксис resolver_timeout *время;*

Умолчание resolver_timeout 30s;

Контекст http, server, location

Задаёт таймаут для преобразования имени в адрес, например:

```
resolver_timeout 5s;
```

root

Синтаксис root *путь;*

Умолчание root html;

Контекст http, server, location, if в location

Задаёт корневой каталог для запросов. Например, при такой конфигурации

```
location /i/ {
  root /data/w3;
}
```

в ответ на запрос /i/top.gif будет отдан файл /data/w3/i/top.gif.

В значении параметра путь можно использовать переменные, кроме $document_root и $realpath_root.

Путь к файлу формируется путём простого добавления URI к значению директивы root. Если же URI необходимо поменять, следует воспользоваться директивой alias.

satisfy

Синтаксис satisfy all | any;

Умолчание satisfy all;

Контекст http, server, location

Разрешает доступ, если все (*all*) или хотя бы один (*any*) из модулей http_access, http_auth_basic или http_auth_request.

```
location / {
  satisfy any;

  allow 192.168.1.0/32;
  deny  all;

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

send_lowat

Синтаксис send_lowat *размер;*

Умолчание send_lowat 0;

Контекст http, server, location

При установке этой директивы в ненулевое значение Angie будет пытаться минимизировать число операций отправки на клиентских сокетах либо при помощи флага *NOTE_LOWAT* метода kqueue, либо при помощи параметра сокета *SO_SNDLOWAT*. В обоих случаях будет использован указанный размер.

send_timeout

Синтаксис send_timeout *время;*

Умолчание send_timeout 60s;

Контекст http, server, location

Задаёт таймаут при передаче ответа клиенту. Таймаут устанавливается не на всю передачу ответа, а только между двумя операциями записи. Если по истечении этого времени клиент ничего не примет, соединение будет закрыто.

sendfile

Синтаксис sendfile on | off;

Умолчание sendfile off;

Контекст http, server, location, if в location

Разрешает или запрещает использовать *sendfile()*.

Возможно использование aio для подгрузки данных для *sendfile()*:

```
location /video/ {
  sendfile       on;
  tcp_nopush     on;
  aio            on;
}
```

В такой конфигурации функция *sendfile()* вызывается с флагом *SF_NODISKIO*, в результате чего она не блокируется на диске, а сообщает об отсутствии данных в памяти. После этого Angie инициирует асинхронную подгрузку данных, читая один байт. При этом ядро FreeBSD подгружает в память первые 128K байт файла, однако при последующих чтениях файл подгружается частями только по 16K. Изменить это можно с помощью директивы read_ahead.

sendfile_max_chunk

Синтаксис sendfile_max_chunk *размер;*

Умолчание sendfile_max_chunk 2m;

Контекст http, server, location

Ограничивает объём данных, который может передан за один вызов *sendfile()*. Без этого ограничения одно быстрое соединение может целиком захватить рабочий процесс.

server

Синтаксис server { … }

Умолчание —

Контекст http

Задаёт конфигурацию для виртуального сервера. Чёткого разделения виртуальных серверов на IP-based (на основании IP-адреса) и name-based (на основании поля “Host” заголовка запроса) нет. Вместо этого директивами listen описываются все адреса и порты, на которых нужно принимать соединения для этого сервера, а в директиве server_name указываются все имена серверов.

Подробнее: Как обрабатываются запросы

server_name

Синтаксис server_name *имя* …;

Умолчание server_name "";

Контекст server

Задаёт имена виртуального сервера, например:

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

Первое имя становится основным именем сервера.

В именах серверов можно использовать звёздочку (“*”) для замены первой или последней части имени:

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

Такие имена называются именами с маской.

Два первых вышеприведённых имени можно объединить в одно:

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

В качестве имени сервера можно также использовать регулярное выражение, указав перед ним тильду (“~”):

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

Регулярное выражение может содержать выделения, которые могут затем использоваться в других директивах:

```
server {
  server_name ~^(www\.)?(.+)$;

  location / {
     root /sites/$2;
  }
}

server {
  server_name _;

  location / {
     root /sites/default;
  }
```

```
}
```

Именованные выделения в регулярном выражении создают переменные, которые могут затем использоваться в других директивах:

```
  server {
    server_name ~^(www\.)?(?<domain>.+)$;

    location / {
       root /sites/$domain;
    }
  }

server {
    server_name _;

    location / {
       root /sites/default;
    }
  }
```

Примечание

Если параметр директивы установлен в $hostname, то подставляется имя хоста (*hostname*) машины.

При поиске виртуального сервера по имени, если имени соответствует несколько из указанных вариантов, например, одновременно подходят и имя с маской, и регулярное выражение, будет выбран первый подходящий вариант в следующем порядке приоритета:

- точное имя
- самое длинное имя с маской в начале, например “*.example.com”
- самое длинное имя с маской в конце, например “mail.*”
- первое подходящее регулярное выражение (в порядке следования в конфигурационном файле)

server_name_in_redirect

Синтаксис	server_name_in_redirect on | off;
Умолчание	server_name_in_redirect off;
Контекст	http, server, location

Разрешает или запрещает использовать в абсолютных перенаправлениях, выдаваемых Angie PRO, основное имя сервера, задаваемое директивой server_name.

on	используется основное имя сервера, задаваемое директивой server_name
off	используется имя, указанное в поле “Host” заголовка запроса. Если же этого поля нет, то используется IP-адрес сервера.

Использование в перенаправлениях порта управляется директивой port_in_redirect.

server_names_hash_bucket_size

Синтаксис server_names_hash_bucket_size *размер;*

Умолчание server_names_hash_bucket_size 32 | 64 | 128;

Контекст http

Задаёт размер корзины в хэш-таблицах имён серверов. Значение по умолчанию зависит от размера строки кэша процессора. Подробнее настройка хэш-таблиц обсуждается отдельно.

server_names_hash_max_size

Синтаксис server_names_hash_max_size *размер;*

Умолчание server_names_hash_max_size 512;

Контекст http

Задаёт максимальный размер хэш-таблиц имён серверов. Подробнее настройка хэш-таблиц обсуждается отдельно.

server_tokens

Синтаксис server_tokens on | off | build;

Умолчание server_tokens *on*;

Контекст http, server, location

Разрешает или запрещает выдавать версию Angie на страницах ошибок и в поле “Server” заголовка ответа.

Если указан параметр *build*, то наряду с версией Angie будет также выдаваться имя сборки.

status_zone

Синтаксис status_zone *зона*;

Умолчание —

Конеткст *server*, *location*, *if в location*

Указанная в любом из допустимых контекстов конфигурации, директива включает выделение зоны в разделяемой памяти для сбора метрик, соответствующих контексту. Несколько *server* могут разделять одну зону для сбора данных.

subrequest_output_buffer_size

Синтаксис subrequest_output_buffer_size *размер;*

Умолчание subrequest_output_buffer_size 4k | 8k;

Контекст http, server, location

Задаёт размер буфера, используемого для хранения тела ответа подзапроса. По умолчанию размер одного буфера равен размеру страницы памяти. В зависимости от платформы это или *4K*, или *8K*, однако его можно сделать меньше.

Примечание

Директива применима только для подзапросов, тело ответа которых сохраняется в памяти. Например, подобные подзапросы создаются при помощи SSI.

tcp_nodelay

Синтаксис tcp_nodelay on | off;

Умолчание tcp_nodelay on;

Контекст http, server, location

Разрешает или запрещает использование параметра *TCP_NODELAY*. Параметр включается при переходе соединения в состояние *keep-alive*. Также, он включается на SSL-соединениях, при небуферизованном проксировании и при проксировании WebSocket.

tcp_nopush

Синтаксис tcp_nopush on | off;

Умолчание tcp_nopush off;

Контекст http, server, location

Разрешает или запрещает использование параметра сокета *TCP_NOPUSH* во FreeBSD или *TCP_CORK* в Linux. Параметр включаются только при использовании sendfile. Включение параметра позволяет

- передавать заголовок ответа и начало файла в одном пакете в Linux и во FreeBSD 4.*;
- передавать файл полными пакетами.

try_files

Синтаксис try_files файл … *uri*; try_files файл … = *код*;

Умолчание —

Контекст server, location

Проверяет существование файлов в заданном порядке и использует для обработки запроса первый найденный файл, причём обработка делается в контексте этого же *location*'а. Путь к файлу строится из параметра *файл* в соответствии с директивами root и alias. С помощью слэша в конце имени можно проверить существование каталога, например, *$uri/*. В случае, если ни один файл не найден, то делается внутреннее перенаправление на *uri*, заданный последним параметром. Например:

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

```
}

location = /images/default.gif {
  expires 30s;
}
```

Последний параметр может также указывать на именованный *location*, может также быть кодом:

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

В следующем примере директива *try_files*

```
location / {
  try_files $uri $uri/ @drupal;
}
аналогична директивам
location / {
  error_page 404 = @drupal;
  log_not_found off;
}
```

А здесь

```
location ~ \.php$ {
  try_files $uri @drupal;

  fastcgi_pass ...;

  fastcgi_param SCRIPT_FILENAME /path/to$fastcgi_script_name;

#  ...
}
```

try_files проверяет существование PHP-файла, прежде чем передать запрос FastCGI-серверу.

Пример использования при проксировании Mongrel:

Пример использования вместе с Drupal/FastCGI:

Пример использования вместе с Wordpress и Joomla:

types

Синтаксис	types { … }
Умолчание	types { text/html html; image/gif gif; image/jpeg jpg; }
Контекст	http, server, location

Задаёт соответствие расширений имён файлов и MIME-типов ответов. Расширения нечувствительны к регистру символов. Одному MIME-типу может соответствовать несколько расширений, например:

```
types {
  application/octet-stream bin exe dll;
  application/octet-stream deb;
  application/octet-stream dmg;
}
```

Достаточно полная таблица соответствий входит в дистрибутив Angie и находится в файле conf/mime.types.

Для того чтобы для определённого *location*'а для всех ответов выдавался MIME-тип "application/octet-stream", можно использовать следующее:

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

types_hash_bucket_size

Синтаксис types_hash_bucket_size *размер;*

Умолчание types_hash_bucket_size 64;

Контекст http, server, location

Задаёт размер корзины в хэш-таблицах типов. Подробнее настройка хэш-таблиц обсуждается отдельно.

types_hash_max_size

Синтаксис types_hash_max_size *размер;*

Умолчание types_hash_max_size 1024;

Контекст http, server, location

Задаёт максимальный размер хэш-таблиц типов. Подробнее настройка хэш-таблиц обсуждается отдельно.

underscores_in_headers

Синтаксис underscores_in_headers on | off;

Умолчание underscores_in_headers off;

Контекст http, server

Разрешает или запрещает использование символов подчёркивания в полях заголовка запроса клиента. Если использование символов подчёркивания запрещено, поля заголовка запроса, в именах которых есть подчёркивания, помечаются как недопустимые и подпадают под действие директивы ignore_invalid_headers.

Если директива указана на уровне server, то может использоваться значение из сервера по умолчанию.

variables_hash_bucket_size

Синтаксис variables_hash_bucket_size *размер;*

Умолчание variables_hash_bucket_size 64;

Контекст http

Задаёт размер корзины в хэш-таблице переменных. Подробнее настройка хэш-таблиц обсуждается отдельно.

variables_hash_max_size

Синтаксис variables_hash_max_size *размер;*

Умолчание variables_hash_max_size 1024;

Контекст http

Задаёт максимальный размер хэш-таблиц переменных. Подробнее настройка хэш-таблиц обсуждается отдельно.

Встроенные переменные

Модуль *http_core* поддерживает встроенные переменные, имена которых совпадают с именами переменных веб-сервера Apache. Прежде всего, это переменные, представляющие из себя поля заголовка запроса клиента, такие как *$http_user_agent*, *$http_cookie* и тому подобное. Кроме того, есть и другие переменные:

$angie_version

версия Angie

$arg_ имя

аргумент *имя* в строке запроса

$args

аргументы в строке запроса

$binary_remote_addr

адрес клиента в бинарном виде, длина значения всегда 4 байта для IPv4-адресов или 16 байт для IPv6-адресов

$body_bytes_sent

число байт, переданное клиенту, без учёта заголовка ответа; переменная совместима с параметром “%B” модуля Apache *mod_log_config*

$bytes_sent

число байт, переданных клиенту

$connection

порядковый номер соединения

$connection_requests

текущее число запросов в соединении

$connection_time

время соединения в секундах с точностью до миллисекунд

$content_length

поле “Content-Length” заголовка запроса

$content_type

поле “Content-Type” заголовка запроса

$cookie_ *имя*

cookie *имя*

$document_root

значение директивы root или alias для текущего запроса

$document_uri

то же, что и $uri

$host

в порядке приоритета: имя хоста из строки запроса, или имя хоста из поля “Host” заголовка запроса, или имя сервера, соответствующего запросу

$hostname

имя хоста

$http_ *имя*

произвольное поле заголовка запроса; последняя часть имени переменной соответствует имени поля, приведённому к нижнему регистру, с заменой символов тире на символы подчёркивания

$https

on если соединение работает в режиме SSL, либо пустая строка

$is_args

?, если в строке запроса есть аргументы, и пустая строка, если их нет

$limit_rate

установка этой переменной позволяет ограничивать скорость передачи ответа, см. limit_rate

$msec

текущее время в секундах с точностью до миллисекунд

$pid

номер (PID) рабочего процесса

$pipe

p если запрос был pipelined, иначе *.*

$proxy_protocol_addr

адрес клиента, полученный из заголовка протокола PROXY Протокол PROXY должен быть предварительно включён при помощи установки параметра *proxy_protocol* в директиве listen.

$proxy_protocol_port

порт клиента, полученный из заголовка протокола PROXY Протокол PROXY должен быть предварительно включён при помощи установки параметра *proxy_protocol* в директиве listen.

$proxy_protocol_server_addr

адрес сервера, полученный из заголовка протокола PROXY Протокол PROXY должен быть предварительно включён при помощи установки параметра *proxy_protocol* в директиве listen.

$proxy_protocol_server_port

порт сервера, полученный из заголовка протокола PROXY Протокол PROXY должен быть предварительно включён при помощи установки параметра *proxy_protocol* в директиве listen.

$proxy_protocol_tlv_ *имя*

TLV, полученный из заголовка протокола PROXY. *Имя* может быть именем типа TLV или его числовым значением. В последнем случае значение задаётся в шестнадцатеричном виде и должно начинаться с *0x*:

```
$proxy_protocol_tlv_alpn
$proxy_protocol_tlv_0x01
```

SSL TLV могут также быть доступны как по имени типа TLV, так и по его числовому значению, оба должны начинаться с *ssl_*:

```
$proxy_protocol_tlv_ssl_version
$proxy_protocol_tlv_ssl_0x21
```

Поддерживаются следующие имена типов TLV:

- *alpn (0x01)* - протокол более высокого уровня, используемый поверх соединения
- *authority (0x02)* - значение имени хоста, передаваемое клиентом
- *unique_id (0x05)* - уникальный идентификатор соединения
- *netns (0x30)* - имя пространства имён
- *ssl (0x20)* - структура SSL TLV в бинарном виде

Поддерживаются следующие имена типов SSL TLV:

- *ssl_version (0x21)* - версия SSL, используемая в клиентском соединении
- *ssl_cn (0x22)* - Common Name сертификата

- *ssl_cipher (0x23)* - имя используемого шифра
- *ssl_sig_alg (0x24)* - алгоритм, используемый для подписи сертификата
- *ssl_key_alg (0x25)* - алгоритм публичного ключа

Также поддерживается следующее специальное имя типа SSL TLV:

- *ssl_verify* - результат проверки клиентского сертификата: *0*, если клиент предоставил сертификат и он был успешно верифицирован, либо ненулевое значение

Протокол PROXY должен быть предварительно включён при помощи установки параметра *proxy_protocol* в директиве listen.

$query_string

то же, что и $args

$realpath_root

абсолютный путь, соответствующий значению директивы root или alias для текущего запроса, в котором все символические ссылки преобразованы в реальные пути

$remote_addr

адрес клиента

$remote_port

порт клиента

$remote_user

имя пользователя, использованное в Basic аутентификации

$request

первоначальная строка запроса целиком

$request_body

тело запроса
Значение переменной *появляется* в *location*'ах, обрабатываемых директивами *proxy_pass*, *fastcgi_pass*, *uwsgi_pass* и *scgi_pass*, когда тело было прочитано в буфер в памяти.

$request_body_file

имя временного файла, в котором хранится тело запроса
По завершении обработки файл необходимо удалить. Для того чтобы тело запроса всегда записывалось в файл, следует включить client_body_in_file_only. При передаче имени временного файла в проксированном запросе или в запросе к FastCGI/uwsgi/SCGI-серверу следует запретить передачу самого тела директивами proxy_pass_request_body off, fastcgi_pass_request_body off, uwsgi_pass_request_body off или scgi_pass_request_body off соответственно.

$request_completion

"OK" если запрос завершился, либо пустая строка

$request_filename

путь к файлу для текущего запроса, формируемый из директив root или alias и URI запроса

$request_id

уникальный идентификатор запроса, сформированный из 16 случайных байт, в шестнадцатеричном виде

$request_length

длина запроса (включая строку запроса, заголовок и тело запроса)

$request_method

метод запроса, обычно “GET” или “POST”

$request_time

время обработки запроса в секундах с точностью до миллисекунд; время, прошедшее с момента чтения первых байт от клиента

$request_uri

первоначальный URI запроса целиком (с аргументами)

$scheme

схема запроса, “http” или “https”

$sent_http_ имя

произвольное поле заголовка ответа; последняя часть имени переменной соответствует имени поля, приведённому к нижнему регистру, с заменой символов тире на символы подчёркивания

$sent_trailer_ *имя*

произвольное поле, отправленное в конце ответа; последняя часть имени переменной соответствует имени поля, приведённому к нижнему регистру, с заменой символов тире на символы подчёркивания

$server_addr

адрес сервера, принявшего запрос
Получение значения этой переменной обычно требует одного системного вызова. Чтобы избежать системного вызова, в директивах listen следует указывать адреса и использовать параметр *bind*.

$server_name

имя сервера, принявшего запрос

$server_port

порт сервера, принявшего запрос

$server_protocol

протокол запроса, обычно “HTTP/1.0”, “HTTP/1.1” или “HTTP/2.0”

$status

статус ответа

$time_iso8601

локальное время в формате по стандарту ISO 8601

$time_local

локальное время в Common Log Format

$tcpinfo_rtt, $tcpinfo_rttvar, $tcpinfo_snd_cwnd, $tcpinfo_rcv_space

информация о клиентском TCP-соединении; доступна на системах, поддерживающих параметр сокета *TCP_INFO*

$uri

текущий URI запроса в нормализованном виде Значение *$uri* может изменяться в процессе обработки запроса, например, при внутренних перенаправлениях или при использовании индексных файлов.

Модуль http_dav

Предназначен для автоматизации задач управления файлами на сервере по протоколу WebDAV. Модуль обрабатывает HTTP- и WebDAV-методы PUT, DELETE, MKCOL, COPY и MOVE.

По умолчанию этот модуль не собирается, его сборку необходимо разрешить с помощью конфигурационного параметра *--with-http_dav_module*.

Предупреждение

WebDAV-клиенты, которые требуют для работы дополнительных WebDAV-методов, не будут работать с этим модулем.

Пример конфигурации

```
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;
    }
}
```

Директивы

create_full_put_path

Синтаксис create_full_put_path on | off;

Умолчание create_full_put_path off;

Контекст http, server, location

По спецификации WebDAV-метод PUT может создавать файл только в уже существующем каталоге. Данная директива разрешает создавать все необходимые промежуточные каталоги.

dav_access

Синтаксис dav_access *пользователи*:*права* …;

Умолчание dav_access user:rw;

Контекст http, server, location

Задаёт права доступа для создаваемых файлов и каталогов, например,

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

Если заданы какие-либо права для group или all, то права для user указывать необязательно:

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

dav_methods

Синтаксис dav_methods off | *метод* …;

Умолчание dav_methods off;

Контекст http, server, location

Разрешает указанные HTTP- и WebDAV-методы. Параметр off запрещает все методы, обрабатываемые данным модулем. Поддерживаются следующие методы: PUT, DELETE, MKCOL, COPY и MOVE.

Файл, загружаемый методом PUT, записывается во временный файл, а потом этот файл переименовывается. Временный файл и его постоянное место хранения могут располагаться на разных файловых системах. Однако нужно учитывать, что в этом случае вместо дешёвой операции переименовывания в пределах одной файловой системы файл копируется с одной файловой системы на другую. Поэтому лучше, если сохраняемые файлы будут находиться на той же файловой системе, что и каталог с временными файлами, задаваемый директивой client_body_temp_path для данного *location*.

При создании файла с помощью метода PUT можно задать дату модификации, передав её в поле заголовка “Date”.

min_delete_depth

Синтаксис min_delete_depth *число*;

Умолчание min_delete_depth 0;

Контекст http, server, location

Разрешает методу DELETE удалять файлы при условии, что число элементов в пути запроса не меньше заданного. Например, директива

```
min_delete_depth 4;
```

разрешает удалять файлы по запросам

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

и запрещает удаление

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

Модуль http_empty_gif

Выдаёт однопиксельный прозрачный GIF.

Пример конфигурации

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

Директивы

empty_gif

Синтаксис empty_gif;

Умолчание —

Контекст location

Разрешает в содержащем location выдавать однопиксельный прозрачный GIF.

Модуль http_fastcgi

Позволяет передавать запросы FastCGI-серверу.

Пример конфигурации

```
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;
}
```

Директивы

fastcgi_bind

Синтаксис fastcgi_bind *адрес* [transparent] | off;

Умолчание —

Контекст http, server, location

Задаёт локальный IP-адрес с необязательным портом, который будет использоваться в исходящих соединениях с FastCGI-сервером. В значении параметра допустимо использование переменных. Специальное значение off отменяет действие унаследованной с предыдущего уровня конфигурации директивы *fastcgi_bind*, позволяя системе самостоятельно выбирать локальный IP-адрес и порт.

Параметр transparent позволяет задать нелокальный IP-адрес, который будет использоваться в исходящих соединениях с FastCGI-сервером, например, реальный IP-адрес клиента:

```
fastcgi_bind $remote_addr transparent;
```

Для работы параметра обычно требуется запустить рабочие процессы Angie с привилегиями суперпользователя. В Linux это не требуется, так как если указан параметр *transparent*, то рабочие процессы наследуют *capability CAP_NET_RAW* из главного процесса.

Внимание

Необходимо настроить таблицу маршрутизации ядра для перехвата сетевого трафика с FastCGI-сервера.

fastcgi_buffer_size

Синтаксис fastcgi_buffer_size *размер*;

Умолчание fastcgi_buffer_size 4k|8k;

Контекст http, server, location

Задаёт размер буфера, в который будет читаться первая часть ответа, получаемого от FastCGI-сервера. В этой части ответа находится, как правило, небольшой заголовок ответа. По умолчанию размер одного буфера равен размеру страницы памяти. В зависимости от платформы это или 4K, или 8K, однако его можно сделать меньше.

fastcgi_buffering

Синтаксис fastcgi_buffering on | off;

Умолчание fastcgi_buffering on;

Контекст http, server, location

Разрешает или запрещает использовать буферизацию ответов FastCGI-сервера.

off	Angie принимает ответ FastCGI-сервера как можно быстрее, сохраняя его в буферы, заданные директивами fastcgi_buffer_size и fastcgi_buffers. Если ответ не вмещается целиком в память, то его часть может быть записана на диск во временный файл. Запись во временные файлы контролируется директивами fastcgi_max_temp_file_size и fastcgi_temp_file_write_size.
on	ответ синхронно передаётся клиенту сразу же по мере его поступления. Angie не пытается считать весь ответ FastCGI-сервера. Максимальный размер данных, который Angie может принять от сервера за один раз, задаётся директивой fastcgi_buffer_size.

Буферизация может быть также включена или выключена путём передачи значения “yes” или “no” в поле “X-Accel-Buffering” заголовка ответа. Эту возможность можно запретить с помощью директивы fastcgi_ignore_headers.

fastcgi_buffers

Синтаксис fastcgi_buffers *число размер*;

Умолчание fastcgi_busy_buffers_size 8 4k|8k;

Контекст http, server, location

Задаёт число и размер буферов для одного соединения, в которые будет читаться ответ, получаемый от FastCGI-сервера.
По умолчанию размер одного буфера равен размеру страницы. В зависимости от платформы это или 4K, или 8K.

fastcgi_busy_buffers_size

Синтаксис fastcgi_busy_buffers_size *размер*;

Умолчание fastcgi_busy_buffers_size 8k|16k;

Контекст http, server, location

При включённой буферизации ответов FastCGI-сервера, ограничивает суммарный размер буферов, которые могут быть заняты для отправки ответа клиенту, пока ответ ещё не прочитан целиком. Оставшиеся буферы тем временем могут использоваться для чтения ответа и, при необходимости, буферизации части ответа во временный файл. По умолчанию размер ограничен двумя буферами, заданными директивами fastcgi_buffer_size и fastcgi_buffers.

fastcgi_cache

Синтаксис fastcgi_cache *зона* | off;

Умолчание fastcgi_cache off;

Контекст http, server, location

Задаёт зону разделяемой памяти, используемой для кэширования. Одна и та же зона может использоваться в нескольких местах. В значении параметра можно использовать переменные. Параметр off запрещает кэширование, унаследованное с предыдущего уровня конфигурации.

fastcgi_cache_background_update

Синтаксис fastcgi_cache_background_update on | off;

Умолчание fastcgi_cache_background_update off;

Контекст http, server, location

Позволяет запустить фоновый подзапрос для обновления просроченного элемента кэша, в то время как клиенту возвращается устаревший закэшированный ответ. Использование устаревшего закэшированного ответа в момент его обновления должно быть разрешено.

fastcgi_cache_bypass

Синтаксис fastcgi_cache_bypass *строка* …;

Умолчание —

Контекст http, server, location

Задаёт условия, при которых ответ не будет браться из кэша. Если значение хотя бы одного из строковых параметров непустое и не равно “0”, то ответ не берётся из кэша:

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

Можно использовать совместно с директивой fastcgi_no_cache.

fastcgi_cache_key

Синтаксис fastcgi_cache_key *строка*;

Умолчание —

Контекст http, server, location

Задаёт ключ для кэширования, например,

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

fastcgi_cache_lock

Синтаксис fastcgi_cache_lock on | off;

Умолчание fastcgi_cache_lock off;

Контекст http, server, location

Если включено, одновременно только одному запросу будет позволено заполнить новый элемент кэша, идентифицируемый согласно директиве fastcgi_cache_key, передав

запрос на FastCGI-сервер. Остальные запросы этого же элемента будут либо ожидать появления ответа в кэше, либо освобождения блокировки этого элемента, в течение времени, заданного директивой fastcgi_cache_lock_timeout.

fastcgi_cache_lock_age

Синтаксис fastcgi_cache_lock_age *время*;

Умолчание fastcgi_cache_lock_age 5s;

Контекст http, server, location

Если последний запрос, переданный на FastCGI-сервер для заполнения нового элемента кэша, не завершился за указанное время, на FastCGI-сервер может быть передан ещё один запрос.

fastcgi_cache_lock_timeout

Синтаксис fastcgi_cache_lock_timeout *время*;

Умолчание fastcgi_cache_lock_timeout 5s;

Контекст http, server, location

Задаёт таймаут для fastcgi_cache_lock. По истечении указанного времени запрос будет передан на FastCGI-сервер, однако ответ не будет закэширован.

fastcgi_cache_max_range_offset

Синтаксис fastcgi_cache_max_range_offset *число*;

Умолчание —

Контекст http, server, location

Задаёт смещение в байтах для запросов с указанием диапазона запрашиваемых байт (byte-range requests). Если диапазон находится за указанным смещением, range-запрос будет передан на FastCGI-сервер и ответ не будет закэширован.

fastcgi_cache_methods

Синтаксис fastcgi_cache_methods GET | HEAD | POST …;

Умолчание fastcgi_cache_methods GET HEAD;

Контекст http, server, location

Если метод запроса клиента указан в этой директиве, то ответ будет закэширован. Методы “GET” и “HEAD” всегда добавляются в список, но тем не менее рекомендуется перечислять их явно. См. также директиву fastcgi_no_cache.

fastcgi_cache_min_uses

Синтаксис fastcgi_cache_min_uses *число*;

Умолчание fastcgi_cache_min_uses 1;

Контекст http, server, location

Задаёт число запросов, после которого ответ будет закэширован.

fastcgi_cache_path

Синтаксис fastcgi_cache_path *путь* [levels = *уровни*] [use_temp_path=on|off] keys_zone = *имя:размер* [inactive = *время*] [max_size = *размер*] [min_free = *размер*] [manager_files = *число*] [manager_sleep = *время*] [manager_threshold = *время*] [loader_files = *число*] [loader_sleep = *время*] [loader_threshold = *время*];;

Умолчание fastcgi_cache_min_uses 1;

Контекст http, server, location

Задаёт путь и другие параметры кэша. Данные кэша хранятся в файлах. Ключом и именем файла в кэше является результат функции MD5 от проксированного URL.

Параметр levels задаёт уровни иерархии кэша: можно задать от 1 до 3 уровней, на каждом уровне допускаются значения 1 или 2. Например, при использовании

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

имена файлов в кэше будут такого вида:

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

Кэшируемый ответ сначала записывается во временный файл, а потом этот файл переименовывается. Временные файлы и кэш могут располагаться на разных файловых системах. Однако нужно учитывать, что в этом случае вместо дешёвой операции переименовывания в пределах одной файловой системы файл копируется с одной файловой системы на другую. Поэтому лучше, если кэш будет находиться на той же файловой системе, что и каталог с временными файлами.

Какой из каталогов будет использоваться для временных файлов определяется параметром use_temp_path.

on	Если параметр не задан или установлен в значение “on”, будет использоваться каталог, задаваемый директивой fastcgi_temp_path для данного location.
off	временные файлы будут располагаться непосредственно в каталоге кэша.

Кроме того, все активные ключи и информация о данных хранятся в зоне разделяемой памяти, имя и размер которой задаются параметром keys_zone. Зоны размером в 1 мегабайт достаточно для хранения около 8 тысяч ключей.

Если к данным кэша не обращаются в течение времени, заданного параметром inactive, то данные удаляются, независимо от их свежести. По умолчанию inactive равен 10 минутам.

Специальный процесс **cache manager** следит за максимальным размером кэша, а также за минимальным объёмом свободного места на файловой системе с кэшем, и удаляет

наименее востребованные данные при превышении максимального размера кэша или недостаточном объёме свободного места. Удаление данных происходит итерациями.

max_size	максимальное пороговое значение размера кэша
min_free	минимальное пороговое значение объёма свободного места на файловой системе с кэшем
manager_files	максимальное количество удаляемых элементов кэша за одну итерацию По умолчанию 100
manager_threshold	ограничивает время работы одной итерации По умолчанию 200 миллисекунд
manager_sleep	время, в течение которого выдерживается пауза между итерациями По умолчанию 50 миллисекунд

Через минуту после старта Angie активируется специальный процесс **cache loader**, который загружает в зону кэша информацию о ранее закэшированных данных, хранящихся на файловой системе. Загрузка также происходит итерациями.

loader_files	максимальное количество элементов кэша к загрузке в одну итерацию По умолчанию *100*
loader_threshold	ограничивает время работы одной итерации По умолчанию *200* миллисекунд
loader_sleep	время, в течение которого выдерживается пауза между итерациями По умолчанию *50* миллисекунд

fastcgi_cache_revalidate

Синтаксис fastcgi_cache_revalidate on | off;

Умолчание fastcgi_cache_revalidate off;

Контекст http, server, location

Разрешает ревалидацию просроченных элементов кэша при помощи условных запросов с полями заголовка “If-Modified-Since” и “If-None-Match”.

fastcgi_cache_use_stale

Синтаксис fastcgi_cache_use_stale error | timeout | invalid_header | updating | http_500 | http_503 | http_403 | http_429 | off …;

Умолчание fastcgi_cache_use_stale off;

Контекст http, server, location

Определяет, в каких случаях можно использовать устаревший закэшированный ответ. Параметры директивы совпадают с параметрами директивы fastcgi_next_upstream.

error	позволяет использовать устаревший закэшированный ответ при невозможности выбора FastCGI-сервера для обработки запроса.
updating	дополнительный параметр, разрешает использовать устаревший закэшированный ответ, если на данный момент он уже обновляется. Это позволяет минимизировать число обращений к FastCGI-серверам при обновлении закэшированных данных.

Использование устаревшего закэшированного ответа может также быть разрешено непосредственно в заголовке ответа на определённое количество секунд после того, как ответ устарел.

- Расширение stale-while-revalidate поля заголовка “Cache-Control” разрешает использовать устаревший закэшированный ответ, если на данный момент он уже обновляется.
- Расширение stale-if-error поля заголовка “Cache-Control” разрешает использовать устаревший закэшированный ответ в случае ошибки.

Примечание

Такой способ менее приоритетен, чем задание параметров директивы.

Чтобы минимизировать число обращений к FastCGI-серверам при заполнении нового элемента кэша, можно воспользоваться директивой fastcgi_cache_lock.

fastcgi_cache_valid

Синтаксис fastcgi_cache_valid [*код* …] *время*;

Умолчание —

Контекст http, server, location

Задаёт время кэширования для разных кодов ответа. Например, директивы

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

задают время кэширования 10 минут для ответов с кодами 200 и 302 и 1 минуту для ответов с кодом 404.

Если указано только время кэширования,

```
fastcgi_cache_valid 5m;
```

то кэшируются только ответы 200, 301 и 302.

Кроме того, можно кэшировать любые ответы с помощью параметра any:

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

Примечание

Параметры кэширования могут также быть заданы непосредственно в заголовке ответа. Такой способ приоритетнее, чем задание времени кэширования с помощью директивы.

- Поле заголовка “X-Accel-Expires” задаёт время кэширования ответа в секундах. Значение *0* запрещает кэшировать ответ. Если значение начинается с префикса @, оно задаёт абсолютное время в секундах с начала эпохи, до которого ответ может быть закэширован.
- Если в заголовке нет поля “X-Accel-Expires”, параметры кэширования определяются по полям заголовка “Expires” или “Cache-Control”.
- Ответ, в заголовке которого есть поле “Set-Cookie”, не будет кэшироваться.
- Ответ, в заголовке которого есть поле “Vary” со специальным значением “*”, не будет кэшироваться. Ответ, в заголовке которого есть поле “Vary” с другим значением, будет закэширован с учётом соответствующих полей заголовка запроса.

Обработка одного или более из этих полей заголовка может быть отключена при помощи директивы fastcgi_ignore_headers.

fastcgi_catch_stderr

Синтаксис fastcgi_catch_stderr *строка*;

Умолчание —

Контекст http, server, location

Задаёт строку для поиска в потоке ошибок ответа, полученного от FastCGI-сервера. Если строка найдена, то считается, что FastCGI-сервер вернул неверный ответ. Это позволяет обрабатывать ошибки приложений в Angie Pro, например:

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

fastcgi_connect_timeout

Синтаксис fastcgi_connect_timeout *время*;

Умолчание fastcgi_connect_timeout 60s;

Контекст http, server, location

Задаёт таймаут для установления соединения с FastCGI-сервером. Необходимо иметь в виду, что этот таймаут обычно не может превышать 75 секунд.

fastcgi_force_ranges

Синтаксис fastcgi_force_ranges on | off;

Умолчание fastcgi_force_ranges off;

Контекст http, server, location

Включает поддержку диапазонов запрашиваемых байт (byte-range) для кэшированных и некэшированных ответов FastCGI-сервера вне зависимости от наличия поля “Accept-Ranges” в заголовках этих ответов.

fastcgi_hide_header

Синтаксис fastcgi_hide_header *поле*;

Умолчание —

Контекст http, server, location

По умолчанию Angie не передаёт клиенту поля заголовка “Status” и “X-Accel-…” из ответа FastCGI-сервера. Директива *fastcgi_hide_header* задаёт дополнительные поля, которые не будут передаваться. Если же передачу полей нужно разрешить, можно воспользоваться директивой fastcgi_pass_header.

fastcgi_ignore_client_abort

Синтаксис fastcgi_ignore_client_abort on | off;

Умолчание fastcgi_ignore_client_abort off;

Контекст http, server, location

Определяет, закрывать ли соединение с FastCGI-сервером в случае, если клиент закрыл соединение, не дождавшись ответа.

fastcgi_ignore_headers

Синтаксис fastcgi_ignore_headers *поле*;

Умолчание —

Контекст http, server, location

Запрещает обработку некоторых полей заголовка из ответа FastCGI-сервера. В директиве можно указать поля “X-Accel-Redirect”, “X-Accel-Expires”, “X-Accel-Limit-Rate”, “X-Accel-Buffering”, “X-Accel-Charset”, “Expires”, “Cache-Control”, “Set-Cookie” и “Vary”.

Если не запрещено, обработка этих полей заголовка заключается в следующем:

- “X-Accel-Expires”, “Expires”, “Cache-Control”, “Set-Cookie” и “Vary” задают параметры кэширования ответа;
- “X-Accel-Redirect” производит внутреннее перенаправление на указанный URI;
- “X-Accel-Limit-Rate” задаёт ограничение скорости передачи ответа клиенту;
- “X-Accel-Buffering” включает или выключает буферизацию ответа;
- “X-Accel-Charset” задаёт желаемую кодировку ответа.

fastcgi_index

Синтаксис fastcgi_index *имя*;

Умолчание —

Контекст http, server, location

Задаёт имя файла, который при создании переменной $fastcgi_script_name будет добавляться после URI, если URI заканчивается слэшом. Например, при таких настройках

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

и запросе /page.php параметр SCRIPT_FILENAME будет равен /home/www/scripts/php/page.php,

а при запросе / - /home/www/scripts/php/index.php.

fastcgi_intercept_errors

Синтаксис fastcgi_intercept_errors on | off;

Умолчание fastcgi_intercept_errors off;

Контекст http, server, location

Определяет, передавать ли клиенту ответы FastCGI-сервера с кодом больше либо равным 300, или же перехватывать их и перенаправлять на обработку Angie с помощью директивы error_page.

fastcgi_keep_conn

Синтаксис fastcgi_keep_conn on | off;

Умолчание fastcgi_keep_conn off;

Контекст http, server, location

По умолчанию FastCGI-сервер будет закрывать соединение сразу же после отправки ответа. При установке значения on Angie указывает FastCGI-серверу оставлять соединения открытыми. Это в частности требуется для функционирования постоянных соединений с FastCGI-серверами.

fastcgi_limit_rate

Синтаксис fastcgi_limit_rate *скорость*;

Умолчание fastcgi_limit_rate 0;

Контекст http, server, location

Ограничивает скорость чтения ответа от проксируемого сервера. *Скорость* задаётся в байтах в секунду.

0	отключает ограничение скорости

Примечание

Ограничение устанавливается на запрос, поэтому, если Angie одновременно откроет два соединения к FastCGI-серверу, суммарная скорость будет вдвое выше заданного

ограничения. Ограничение работает только в случае, если включена буферизация ответов FastCGI-сервера.

fastcgi_max_temp_file_size

Синтаксис fastcgi_max_temp_file_size *размер*;

Умолчание fastcgi_max_temp_file_size 1024m;

Контекст http, server, location

Если включена буферизация ответов FastCGI-сервера, и ответ не вмещается целиком в буферы, заданные директивами fastcgi_buffer_size и fastcgi_buffers, часть ответа может быть записана во временный файл. Эта директива задаёт максимальный размер временного файла. Размер данных, сбрасываемых во временный файл за один раз, задаётся директивой fastcgi_temp_file_write_size.

0	отключает возможность буферизации ответов во временные файлы

Примечание

Данное ограничение не распространяется на ответы, которые будут закэшированы или сохранены на диске.

fastcgi_next_upstream

Синтаксис fastcgi_next_upstream error | timeout | invalid_header | http_500 | http_503 | http_403 | http_404 | http_429 | non_idempotent | off …;

Умолчание fastcgi_next_upstream error timeout;

Контекст http, server, location

Определяет, в каких случаях запрос будет передан следующему серверу:

error	произошла ошибка соединения с сервером, передачи ему запроса или чтения заголовка ответа сервера;
timeout	произошёл таймаут во время соединения с сервером, передачи ему запроса или чтения заголовка ответа сервера;
invalid_header	сервер вернул пустой или неверный ответ;
http_500	сервер вернул ответ с кодом 500;
http_503	сервер вернул ответ с кодом 503;
http_403	сервер вернул ответ с кодом 403;
http_404	сервер вернул ответ с кодом 404;
http_429	сервер вернул ответ с кодом 429;
non_idempotent	обычно запросы с неидемпотентным методом (*POST*, *LOCK*, *PATCH*) не передаются на другой сервер, если запрос серверу группы уже был отправлен; включение параметра явно разрешает

	повторять подобные запросы;
off	запрещает передачу запроса следующему серверу.

Примечание

Необходимо понимать, что передача запроса следующему серверу возможна только при условии, что клиенту ещё ничего не передавалось. То есть, если ошибка или таймаут возникли в середине передачи ответа, то исправить это уже невозможно.

Директива также определяет, что считается неудачной попыткой работы с сервером.

error timeout invalid_header	всегда считаются неудачными попытками, даже если они не указаны в директиве
http_500 http_503 http_429	считаются неудачными попытками, только если они указаны в директиве
http_403 http_404	никогда не считаются неудачными попытками

Передача запроса следующему серверу может быть ограничена по количеству попыток и по времени.

fastcgi_next_upstream_timeout

Синтаксис fastcgi_next_upstream_timeout время;

Умолчание fastcgi_next_upstream_timeout 0;

Контекст http, server, location

Ограничивает время, в течение которого возможна передача запроса *следующему серверу*.

0	отключает это ограничение

fastcgi_next_upstream_tries

Синтаксис fastcgi_next_upstream_tries число;

Умолчание fastcgi_next_upstream_tries 0;

Контекст http, server, location

Ограничивает число допустимых попыток для передачи запроса *следующему серверу*.

0	отключает это ограничение

fastcgi_no_cache

Синтаксис fastcgi_no_cache строка …;

Умолчание —

Контекст http, server, location

Задаёт условия, при которых ответ не будет сохраняться в кэш. Если значение хотя бы одного из строковых параметров непустое и не равно “0”, то ответ не будет сохранён:

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

Можно использовать совместно с директивой *fastcgi_cache_bypass*.

fastcgi_param

Синтаксис fastcgi_param параметр значение [if_not_empty];

Умолчание —

Контекст http, server, location

Задаёт параметр, который будет передаваться FastCGI-серверу. В качестве значения можно использовать текст, переменные и их комбинации. Директивы наследуются с предыдущего уровня конфигурации при условии, что на данном уровне не описаны свои директивы fastcgi_param.

Ниже приведён пример минимально необходимых параметров для PHP:

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

Параметр SCRIPT_FILENAME используется в PHP для определения имени скрипта, а в параметре QUERY_STRING передаются параметры запроса.

Если скрипты обрабатывают запросы POST, то нужны ещё три параметра:

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

Если PHP был собран с параметром конфигурации –enable-force-cgi-redirect, то нужно передавать параметр REDIRECT_STATUS со значением “200”:

```
fastcgi_param REDIRECT_STATUS 200;
```

Если директива указана с if_not_empty, такой параметр с пустым значением передаваться на сервер не будет:

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

fastcgi_pass

Синтаксис fastcgi_pass адрес;

Умолчание —

Контекст location, if в location

Задаёт адрес FastCGI-сервера. Адрес может быть указан в виде доменного имени или IP-адреса, и порта:

```
fastcgi_pass localhost:9000;
```

или в виде пути UNIX-сокета:

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

Если доменному имени соответствует несколько адресов, то все они будут использоваться по очереди (round-robin). И, кроме того, адрес может быть *группой серверов*.

В значении параметра можно использовать переменные. В этом случае, если адрес указан в виде доменного имени, имя ищется среди описанных *групп серверов* и если не найдено, то определяется с помощью *resolver*'а.

fastcgi_pass_header

Синтаксис fastcgi_pass_header поле;

Умолчание —

Контекст http, server, location

Разрешает передавать от FastCGI-сервера клиенту *запрещённые для передачи* поля заголовка.

fastcgi_pass_request_body

Синтаксис fastcgi_pass_request_body on | off;

Умолчание —

Контекст http, server, location

Позволяет запретить передачу исходного тела запроса на FastCGI-сервер. См. также директиву *fastcgi_pass_request_headers*.

fastcgi_pass_request_headers

Синтаксис fastcgi_pass_request_headers on | off;

Умолчание —

Контекст http, server, location

Позволяет запретить передачу полей заголовка исходного запроса на FastCGI-сервер. См. также директиву *fastcgi_pass_request_body*.

fastcgi_read_timeout

Синтаксис fastcgi_read_timeout время;

Умолчание fastcgi_read_timeout 60s;

Контекст http, server, location

Задаёт таймаут при чтении ответа FastCGI-сервера. Таймаут устанавливается не на всю передачу ответа, а только между двумя операциями чтения. Если по истечении этого времени FastCGI-сервер ничего не передаст, соединение закрывается.

fastcgi_request_buffering

Синтаксис fastcgi_request_buffering on | off;

Умолчание fastcgi_request_buffering on;

Контекст http, server, location

Разрешает или запрещает использовать буферизацию тела запроса клиента.

on	тело запроса полностью *читается* от клиента перед отправкой запроса на FastCGI-сервер.
off	тело запроса отправляется на FastCGI-сервер сразу же по мере его поступления. В этом случае запрос не может быть передан *следующему серверу*, если Angie уже начал отправку тела запроса.

fastcgi_send_lowat

Синтаксис fastcgi_send_lowat размер;

Умолчание fastcgi_send_lowat 0;

Контекст http, server, location

При установке директивы в ненулевое значение Angie будет пытаться минимизировать число операций отправки на исходящих соединениях с FastCGI-сервером либо при помощи флага NOTE_LOWAT метода *kqueue*, либо при помощи параметра сокета SO_SNDLOWAT, с указанным размером.

Примечание

Эта директива игнорируется на Linux, Solaris и Windows.

fastcgi_send_timeout

Синтаксис fastcgi_send_timeout время;

Умолчание fastcgi_send_timeout 60s;

Контекст http, server, location

Задаёт таймаут при передаче запроса FastCGI-серверу. Таймаут устанавливается не на всю передачу запроса, а только между двумя операциями записи. Если по истечении этого времени FastCGI-сервер не примет новых данных, соединение закрывается.

fastcgi_socket_keepalive

Синтаксис fastcgi_socket_keepalive on | off;

Умолчание fastcgi_socket_keepalive off;

Контекст http, server, location

Конфигурирует поведение “TCP keepalive” для исходящих соединений к FastCGI-серверу.

""	По умолчанию для сокета действуют настройки операционной системы.
on	для сокета включается параметр SO_KEEPALIVE

fastcgi_split_path_info

Синтаксис fastcgi_split_path_info regex;

Умолчание —

Контекст location

Задаёт регулярное выражение, выделяющее значение для переменной *$fastcgi_path_info*. Регулярное выражение должно иметь два выделения, из которых первое становится значением переменной *$fastcgi_script_name*, а второе - значением переменной *$fastcgi_path_info*. Например, при таких настройках

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

и запросе /show.php/article/0001 параметр SCRIPT_FILENAME будет равен /path/to/php/show.php,

а параметр PATH_INFO - /article/0001.

fastcgi_store

Синтаксис fastcgi_store on | off | строка;

Умолчание fastcgi_store off;

Контекст http, server, location

Разрешает сохранение на диск файлов.

on	сохраняет файлы в соответствии с путями, указанными в директивах *alias* или *root*
off	запрещает сохранение файлов

Кроме того, имя файла можно задать явно с помощью строки с переменными:

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

Время изменения файлов выставляется согласно полученному полю “Last-Modified” в заголовке ответа. Ответ сначала записывается во временный файл, а потом этот файл переименовывается. Временный файл и постоянное место хранения ответа могут располагаться на разных файловых системах. Однако нужно учитывать, что в этом случае вместо дешёвой операции переименовывания в пределах одной файловой системы файл копируется с одной файловой системы на другую. Поэтому лучше, если сохраняемые файлы

будут находиться на той же файловой системе, что и каталог с временными файлами, задаваемый директивой *fastcgi_temp_path* для данного location.

Директиву можно использовать для создания локальных копий статических неизменяемых файлов, например:

```
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

Синтаксис fastcgi_store_access пользователи:права …;

Умолчание fastcgi_store_access user:rw;

Контекст http, server, location

Задаёт права доступа для создаваемых файлов и каталогов, например,

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

Если заданы какие-либо права для group или all, то права для user указывать необязательно:

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

fastcgi_temp_file_write_size

Синтаксис fastcgi_temp_file_write_size размер;

Умолчание fastcgi_temp_file_write_size 8k|16k;

Контекст http, server, location

Ограничивает размер данных, сбрасываемых во временный файл за один раз, при включённой буферизации ответов FastCGI-сервера во временные файлы. По умолчанию размер ограничен двумя буферами, заданными директивами *fastcgi_buffer_size* и *fastcgi_buffers*. Максимальный размер временного файла задаётся директивой *fastcgi_max_temp_file_size*.

fastcgi_temp_path

Синтаксис fastcgi_temp_path путь [уровень1 [уровень2 [уровень3]]]`;

Умолчание fastcgi_temp_path fastcgi_temp;

Контекст http, server, location

Задаёт имя каталога для хранения временных файлов с данными, полученными от FastCGI-серверов. В каталоге может использоваться иерархия подкаталогов до трёх уровней. Например, при такой конфигурации

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

временный файл будет следующего вида:

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

См. также параметр use_temp_path директивы *fastcgi_cache_path*.

Параметры, передаваемые FastCGI-серверу

Поля заголовка HTTP-запроса передаются FastCGI-серверу в виде параметров. В приложениях и скриптах, запущенных в виде FastCGI-сервера, эти параметры обычно доступны в виде переменных среды. Например, поле заголовка “User-Agent” передаётся как параметр HTTP_USER_AGENT. Кроме полей заголовка HTTP-запроса можно передавать произвольные параметры с помощью директивы fastcgi_param.

Встроенные переменные

В модуле *http_fastcgi* есть встроенные переменные, которые можно использовать для формирования параметров с помощью директивы fastcgi_param:

$fastcgi_script_name

URI запроса или же, если URI заканчивается слэшом, то URI запроса, дополненное именем индексного файла, задаваемого директивой fastcgi_index. Эту переменную можно использовать для задания параметров SCRIPT_FILENAME и PATH_TRANSLATED, используемых, в частности, для определения имени скрипта в PHP. Например, для запроса /info/ и при использовании директив

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

параметр SCRIPT_FILENAME будет равен /home/www/scripts/php/info/index.php.

При использовании директивы fastcgi_split_path_info переменная *$fastcgi_script_name* равна значению первого выделения, задаваемого этой директивой.

$fastcgi_path_info

значение второго выделения, задаваемого директивой fastcgi_split_path_info. Эту переменную можно использовать для задания параметра PATH_INFO.

Модуль http_flv

Обеспечивает серверную поддержку псевдо-стриминга для файлов Flash Video (FLV).

Он специальным образом обрабатывает запросы с аргументом start в строке запроса, посылая в ответ содержимое файла с запрошенного смещения в байтах, добавив перед ним FLV-заголовок.

По умолчанию этот модуль не собирается, его сборку необходимо разрешить с помощью конфигурационного параметра --with-http_flv_module.

Пример конфигурации

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

Директивы

flv

Синтаксис flv;

Умолчание —

Контекст location

Включает в содержащем location обработку этим модулем.

Модуль http_geo

Создаёт переменные, значения которых зависят от IP-адреса клиента.

Пример конфигурации

```
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;
}
```

Директивы

geo

Синтаксис geo [*$адрес*] *$переменная* { … }

Умолчание —

Контекст http

Описывает для указанной переменной зависимость значения от IP-адреса клиента. По умолчанию адрес берётся из переменной $remote_addr, но его также можно получить из другой переменной, например:

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

Примечание

Поскольку переменные вычисляются только в момент использования, само по себе наличие даже большого числа объявлений переменных “geo” не влечёт за собой никаких дополнительных расходов на обработку запросов.

Если значение переменной не представляет из себя правильный IP-адрес, то используется адрес “255.255.255.255”.

Адреса задаются либо префиксами в формате CIDR (включая одиночные адреса), либо в виде диапазонов.

Также поддерживаются следующие специальные параметры:

delete	удаляет описанную сеть
default	значение переменной, если адрес клиента не соответствует ни одному из заданных адресов. При задании адресов в формате CIDR вместо *default* можно использовать “*0.0.0.0/0*” и “*::/0*”. Если параметр *default* не указан, значением по умолчанию будет пустая строка.
include	включает файл с адресами и значениями. Включений может быть несколько.
proxy	задаёт доверенные адреса, при запросе с которых будет использоваться адрес в переданном поле заголовка запроса “*X-Forwarded-For*”. В отличие от обычных адресов, доверенные адреса проверяются последовательно.
proxy_recursive	включает рекурсивный поиск адреса. При выключенном рекурсивном поиске вместо исходного адреса клиента, совпадающего с одним из доверенных адресов, будет использоваться последний адрес, переданный в “*X-Forwarded-For*”. При включённом рекурсивном поиске вместо исходного адреса клиента, совпадающего с одним из доверенных адресов, будет использоваться последний не доверенный адрес, переданный в “*X-Forwarded-For*”.
ranges	указывает, что адреса задаются в виде диапазонов. Этот параметр должен быть первым. Для ускорения загрузки гео-базы нужно располагать адреса в порядке возрастания.

Пример:

```
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;
}
```

В файле *conf/geo.conf* могут быть такие строки:

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

В качестве значения выбирается максимальное совпадение, например, для адреса 127.0.0.1 будет выбрано значение “RU”, а не “US”.

Пример описания диапазонов:

```
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;
}
```

Модуль http_geoip

Создаёт переменные, значения которых зависят от IP-адреса клиента, используя готовые базы данных MaxMind.

При использовании баз данных с поддержкой IPv6 IPv4-адреса ищутся отображёнными на IPv6.

По умолчанию этот модуль не собирается, его сборку необходимо разрешить с помощью конфигурационного параметра *--with-http_geoip_module*.

Внимание

Для сборки и работы этого модуля нужна библиотека MaxMind GeoIP.

Пример конфигурации

```
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;
    ...
```

Директивы

geoip_country

Синтаксис geoip_country *файл*;

Умолчание —

Контекст http

Задаёт базу данных для определения страны в зависимости от значения IP-адреса клиента. При использовании этой базы данных доступны следующие переменные:

$geoip_country_code	двухбуквенный код страны, например, “RU”, “US”.
$geoip_country_code3	трёхбуквенный код страны, например, “RUS”, “USA”.
$geoip_country_name	название страны, например, “Russian Federation”, “United States”.

geoip_city

Синтаксис geoip_city *файл*;

Умолчание —

Контекст http

Задаёт базу данных для определения страны, региона и города в зависимости от значения IP-адреса клиента. При использовании этой базы данных доступны следующие переменные:

$geoip_city_continent_code	двухбуквенный код континента, например, “EU”, “NA”.
$geoip_city_country_code	двухбуквенный код страны, например, “RU”, “US”.
$geoip_city_country_code3	трёхбуквенный код страны, например, “RUS”, “USA”.
$geoip_city_country_name	название страны, например, “Russian Federation”, “United States”.
$geoip_dma_code	DMA-код региона в США (также известный как “код агломерации”), согласно геотаргетингу Google AdWords API.
$geoip_latitude	широта.
$geoip_longitude	долгота.
$geoip_region	двухсимвольный код региона страны (область, край, штат, провинция, федеральная земля и тому подобное), например, “48”, “DC”.
$geoip_region_name	название региона страны (область, край, штат, провинция, федеральная земля и тому подобное), например, “Moscow City”, “District of Columbia”.

$geoip_city	название города, например, “Moscow”, “Washington”.
$geoip_postal_code	почтовый индекс.

geoip_org

Синтаксис geoip_org *файл*;

Умолчание —

Контекст http

Задаёт базу данных для определения названия организации в зависимости от значения IP-адреса клиента. При использовании этой базы данных доступна следующая переменная:

$geoip_org	название организации, например, “The University of Melbourne”.

geoip_proxy

Синтаксис geoip_proxy *файл*;

Умолчание —

Контекст http

Задаёт доверенные адреса, при запросе с которых будет использоваться адрес в переданном поле заголовка запроса “*X-Forwarded-For*”.

geoip_proxy_recursive

Синтаксис geoip_proxy_recursive on | off;

Умолчание geoip_proxy_recursive off;

Контекст http

При выключенном рекурсивном поиске вместо исходного адреса клиента, совпадающего с одним из доверенных адресов, будет использоваться последний адрес, переданный в “*X-Forwarded-For*”. При включённом рекурсивном поиске вместо исходного адреса клиента, совпадающего с одним из доверенных адресов, будет использоваться последний не доверенный адрес, переданный в “*X-Forwarded-For*”.

Модуль http_grpc

Позволяет передавать запросы gRPC-серверу.

Примечание

Для работы этого модуля необходим модуль http_v2.

Пример конфигурации

```
server {
    listen 9000 http2;

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

```
}
```

Директивы

grpc_bind

Синтаксис grpc_bind *адрес* [transparent] | off;

Умолчание —

Контекст http, server, location

Задаёт локальный IP-адрес с необязательным портом, который будет использоваться в исходящих соединениях с gRPC-сервером. В значении параметра допустимо использование переменных. Специальное значение off отменяет действие унаследованной с предыдущего уровня конфигурации директивы *grpc_bind*, позволяя системе самостоятельно выбирать локальный IP-адрес и порт.

Параметр transparent позволяет задать нелокальный IP-адрес, который будет использоваться в исходящих соединениях с gRPC-сервером, например, реальный IP-адрес клиента:

```
grpc_bind $remote_addr transparent;
```

Для работы параметра обычно требуется запустить рабочие процессы Angie с привилегиями суперпользователя. В Linux это не требуется, так как если указан параметр *transparent*, то рабочие процессы наследуют *capability CAP_NET_RAW* из главного процесса.

Внимание

Необходимо настроить таблицу маршрутизации ядра для перехвата сетевого трафика с gRPC-сервера.

grpc_buffer_size

Синтаксис grpc_buffer_size *размер*;

Умолчание grpc_buffer_size 4k|8k;

Контекст http, server, location

Задаёт размер буфера, в который будет читаться первая часть ответа, получаемого от gRPC-сервера. Ответ синхронно передаётся клиенту сразу же по мере его поступления.

grpc_connect_timeout

Синтаксис grpc_connect_timeout *время*;

Умолчание grpc_connect_timeout 60s;

Контекст http, server, location

Задаёт таймаут для установления соединения с gRPC-сервером. Необходимо иметь в виду, что этот таймаут обычно не может превышать 75 секунд.

grpc_hide_header

Синтаксис grpc_hide_header *поле*;

Умолчание —

Контекст http, server, location

По умолчанию Angie не передаёт клиенту поля заголовка “Date”, “Server” и “X-Accel-…” из ответа gRPC-сервера. Директива *grpc_hide_header* задаёт дополнительные поля, которые не будут передаваться. Если же передачу полей нужно разрешить, можно воспользоваться директивой grpc_pass_header.

grpc_ignore_headers

Синтаксис grpc_ignore_headers *поле* …;

Умолчание —

Контекст http, server, location

Запрещает обработку некоторых полей заголовка из ответа gRPC-сервера. В директиве можно указать поля “X-Accel-Redirect” и “X-Accel-Charset”.

Если не запрещено, обработка этих полей заголовка заключается в следующем:

- “X-Accel-Redirect” производит внутреннее перенаправление на указанный URI;
- “X-Accel-Charset” задаёт желаемую кодировку ответа.

grpc_intercept_errors

Синтаксис grpc_intercept_errors on | off;

Умолчание grpc_intercept_errors off;

Контекст http, server, location

Определяет, передавать ли клиенту ответы gRPC-сервера с кодом больше либо равным 300, или же перехватывать их и перенаправлять на обработку Angie с помощью директивы error_page.

grpc_next_upstream

Синтаксис grpc_next_upstream error | timeout | invalid_header | http_500 | http_502 | http_503 | http_504 | http_403 | http_404 | http_429 | non_idempotent | off …;

Умолчание grpc_next_upstream error timeout;

Контекст http, server, location

Определяет, в каких случаях запрос будет передан следующему в группе upstream серверу:

error	произошла ошибка соединения с сервером, передачи ему запроса или чтения заголовка ответа сервера;
timeout	произошёл таймаут во время соединения с сервером, передачи ему запроса или чтения заголовка ответа сервера;

invalid_header	сервер вернул пустой или неверный ответ;
http_500	сервер вернул ответ с кодом 500;
http_502	сервер вернул ответ с кодом 502;
http_503	сервер вернул ответ с кодом 503;
http_504	сервер вернул ответ с кодом 504;
http_403	сервер вернул ответ с кодом 403;
http_404	сервер вернул ответ с кодом 404;
http_429	сервер вернул ответ с кодом 429;
non_idempotent	обычно запросы с неидемпотентным методом (*POST*, *LOCK*, *PATCH*) не передаются на другой сервер, если запрос серверу группы уже был отправлен; включение параметра явно разрешает повторять подобные запросы;
off	запрещает передачу запроса следующему серверу.

Примечание

Необходимо понимать, что передача запроса следующему серверу возможна только при условии, что клиенту ещё ничего не передавалось. То есть, если ошибка или таймаут возникли в середине передачи ответа клиенту, то действие директивы на такой запрос не распространяется.

Директива также определяет, что считается неудачной попыткой работы с сервером.

error timeout invalid_header	всегда считаются неудачными попытками, даже если они не указаны в директиве
http_500 http_502 http_503 http_504 http_429	считаются неудачными попытками, только если они указаны в директиве
http_403 http_404	никогда не считаются неудачными попытками

Передача запроса следующему серверу может быть ограничена по количеству попыток и по времени.

grpc_next_upstream_timeout

Синтаксис	grpc_next_upstream_timeout *время*;
Умолчание	grpc_next_upstream_timeout 0;
Контекст	http, server, location

Ограничивает время, в течение которого возможна передача запроса следующему серверу.

0	отключает это ограничение

grpc_next_upstream_tries

Синтаксис grpc_next_upstream_tries *число*;

Умолчание grpc_next_upstream_tries 0;

Контекст http, server, location

Ограничивает число допустимых попыток для передачи запроса следующему серверу.

0	отключает это ограничение

grpc_pass

Синтаксис grpc_pass *адрес*;

Умолчание —

Контекст location, if в location

Задаёт адрес gRPC-сервера. Адрес может быть указан в виде доменного имени или IP-адреса, и порта:

```
grpc_pass localhost:9000;
```

или в виде пути UNIX-сокета:

```
grpc_pass unix:/tmp/grpc.socket;
```

Также может использоваться схема “*grpc://*”:

```
grpc_pass grpc://127.0.0.1:9000;
```

Для использования gRPC по SSL необходимо использовать схему “*grpcs://*”:

```
grpc_pass grpcs://127.0.0.1:443;
```

Если доменному имени соответствует несколько адресов, то все они будут использоваться по очереди (round-robin). Кроме того, в качестве адреса можно указать группу серверов.

В значении параметра можно использовать переменные. В этом случае, если адрес указан в виде доменного имени, имя ищется среди описанных групп серверов и если не найдено, то определяется с помощью resolver’а.

grpc_pass_header

Синтаксис grpc_pass_header *поле*;

Умолчание —

Контекст http, server, location

Разрешает передавать от gRPC-сервера клиенту запрещённые для передачи поля заголовка.

grpc_read_timeout

Синтаксис grpc_read_timeout *время*;

Умолчание grpc_read_timeout 60s;

Контекст http, server, location

Задаёт таймаут при чтении ответа gRPC-сервера. Таймаут устанавливается не на всю передачу ответа, а только между двумя операциями чтения. Если по истечении этого времени gRPC-сервер ничего не передаст, соединение закрывается.

grpc_send_timeout

Синтаксис grpc_send_timeout *время*;

Умолчание grpc_send_timeout 60s;

Контекст http, server, location

Задаёт таймаут при передаче запроса gRPC-серверу. Таймаут устанавливается не на всю передачу запроса, а только между двумя операциями записи. Если по истечении этого времени gRPC-сервер не примет новых данных, соединение закрывается.

grpc_set_header

Синтаксис grpc_set_header *поле значение*;

Умолчание grpc_set_header Content-Length $content_length;

Контекст http, server, location

Позволяет переопределять или добавлять поля заголовка запроса, передаваемые проксируемому серверу. В качестве *значения* можно использовать текст, переменные и их комбинации. Директивы наследуются с предыдущего уровня конфигурации при условии, что на данном уровне не описаны свои директивы *grpc_set_header*.

Если значение поля заголовка — пустая строка, то поле вообще не будет передаваться gRPC-серверу:

```
grpc_set_header Accept-Encoding "";
```

grpc_socket_keepalive

Синтаксис grpc_socket_keepalive on | off;

Умолчание grpc_socket_keepalive off;

Контекст http, server, location

Конфигурирует поведение “TCP keepalive” для исходящих соединений к проксируемому серверу.

""	По умолчанию для сокета действуют настройки операционной

	системы.
on	для сокета включается параметр *SO_KEEPALIVE*

grpc_ssl_certificate

Синтаксис grpc_ssl_certificate *файл*;

Умолчание —

Контекст http, server, location

Задаёт файл с сертификатом в формате PEM для аутентификации на gRPC SSL-сервере. В имени файла можно использовать переменные.

grpc_ssl_certificate_key

Синтаксис grpc_ssl_certificate_key *файл*;

Умолчание —

Контекст http, server, location

Задаёт файл с секретным ключом в формате PEM для аутентификации на gRPC SSL-сервере.

Вместо файла можно указать значение “engine:*имя*:id“, которое загружает ключ с указанным *id* из OpenSSL engine с заданным именем.

В имени файла можно использовать переменные.

grpc_ssl_ciphers

Синтаксис grpc_ssl_ciphers *шифры*;

Умолчание grpc_ssl_ciphers DEFAULT;

Контекст http, server, location

Описывает разрешённые шифры для запросов к gRPC SSL-серверу. Шифры задаются в формате, поддерживаемом библиотекой OpenSSL.

Полный список можно посмотреть с помощью команды “*openssl ciphers*”.

grpc_ssl_conf_command

Синтаксис grpc_ssl_conf_command *имя значение*;

Умолчание —

Контекст http, server, location

Задаёт произвольные конфигурационные команды OpenSSL при установлении соединения с gRPC SSL-сервером.

Примечание

Директива поддерживается при использовании OpenSSL 1.0.2 и выше.

На одном уровне может быть указано несколько директив *grpc_ssl_conf_command*. Директивы наследуются с предыдущего уровня конфигурации при условии, что на данном уровне не описаны свои директивы *grpc_ssl_conf_command*.

Внимание

Следует учитывать, что изменение настроек OpenSSL напрямую может привести к неожиданному поведению.

grpc_ssl_crl

Синтаксис grpc_ssl_crl *файл*;

Умолчание —

Контекст http, server, location

Указывает файл с отозванными сертификатами (CRL) в формате PEM, используемыми при проверке сертификата gRPC SSL-сервера.

grpc_ssl_name

Синтаксис grpc_ssl_name *имя*;

Умолчание grpc_ssl_name `имя хоста из grpc_pass;`

Контекст http, server, location

Позволяет переопределить имя сервера, используемое при проверке сертификата gRPC SSl-сервера, а также для передачи его через SNI при установлении соединения с gRPC SSL-сервером.

По умолчанию используется имя хоста из grpc_pass.

grpc_ssl_password_file

Синтаксис grpc_ssl_password_file *файл*;

Умолчание —

Контекст http, server, location

Задаёт файл с паролями от секретных ключей, где каждый пароль указан на отдельной строке. Пароли применяются по очереди в момент загрузки ключа.

grpc_ssl_protocols

Синтаксис grpc_ssl_protocols [SSLv2] [SSLv3] [TLSv1] [TLSv1.1] [TLSv1.2] [TLSv1.3];

Умолчание grpc_ssl_protocols TLSv1 TLSv1.1 TLSv1.2;

Контекст http, server, location

Разрешает указанные протоколы для запросов к gRPC SSL-серверу.

grpc_ssl_server_name

Синтаксис grpc_ssl_server_name on | off;

Умолчание grpc_ssl_server_name off;

Контекст http, server, location

Разрешает или запрещает передачу имени сервера через расширение Server Name Indication протокола TLS (SNI, RFC 6066) при установлении соединения с gRPC SSL-сервером.

grpc_ssl_session_reuse

Синтаксис grpc_ssl_session_reuse on | off;

Умолчание grpc_ssl_session_reuse on;

Контекст http, server, location

Определяет, использовать ли повторно SSL-сессии при работе с gRPC-сервером. Если в логах появляются ошибки “*SSL3_GET_FINISHED:digest check failed*”, то можно попробовать выключить повторное использование сессий.

grpc_ssl_trusted_certificate

Синтаксис grpc_ssl_trusted_certificate *файл*;

Умолчание —

Контекст http, server, location

Задаёт файл с доверенными сертификатами CA в формате PEM, используемыми при проверке сертификата gRPC SSL-сервера.

grpc_ssl_verify

Синтаксис grpc_ssl_verify on | off;

Умолчание grpc_ssl_verify off;

Контекст http, server, location

Разрешает или запрещает проверку сертификата gRPC SSL-сервера.

grpc_ssl_verify_depth

Синтаксис grpc_ssl_verify_depth *число*;

Умолчание grpc_ssl_verify_depth 1;

Контекст http, server, location

Устанавливает глубину проверки в цепочке сертификатов gRPC SSL-сервера.

Модуль http_gunzip

Фильтр, распаковывающий ответы с “Content-Encoding: gzip” для тех клиентов, которые не поддерживают метод сжатия “gzip”. Модуль будет полезен, когда данные желательно хранить сжатыми для экономии места и сокращения затрат на ввод-вывод.

По умолчанию этот модуль не собирается, его сборку необходимо разрешить с помощью конфигурационного параметра --with-http_gunzip_module.

Пример конфигурации

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

Директивы

gunzip

Синтаксис gunzip on | off;

Умолчание gunzip off;

Контекст http, server, location

Разрешает или запрещает распаковку ответов, сжатых методом gzip, для тех клиентов, которые его не поддерживают. Если разрешено, то для определения, поддерживает ли клиент gzip, также учитываются следующие директивы: gzip_http_version, gzip_proxied и gzip_disable. См. также директиву gzip_vary.

gunzip_buffers

Синтаксис gunzip_buffers *число размер*;

Умолчание gunzip_buffers 32 4k|16 8k;

Контекст http, server, location

Задаёт число и размер буферов, в которые будет разжиматься ответ. По умолчанию размер одного буфера равен размеру страницы. В зависимости от платформы это или 4K, или 8K.

Модуль http_gzip

Фильтр, сжимающий ответ методом gzip, что позволяет уменьшить размер передаваемых данных в 2 и более раз.

Примечание

При использовании протокола SSL/TLS сжатые ответы могут быть подвержены атакам BREACH.

Пример конфигурации

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

Директивы

gzip

Синтаксис gzip on | off;

Умолчание gzip off;

Контекст http, server, location, if в location

Разрешает или запрещает сжатие ответа методом gzip.

gzip_buffers

Синтаксис gzip_buffers *число размер*;

Умолчание gzip_buffers 32 4k|16 8k;

Контекст http, server, location

Задаёт число и размер буферов, в которые будет сжиматься ответ. По умолчанию размер одного буфера равен размеру страницы. В зависимости от платформы это или 4K, или 8K.

gzip_comp_level

Синтаксис gzip_comp_level *степень*;

Умолчание gzip_comp_level 1;

Контекст http, server, location

Устанавливает степень сжатия ответа методом gzip. Допустимые значения находятся в диапазоне от 1 до 9.

gzip_disable

Синтаксис gzip_disable *regex* …;

Умолчание —

Контекст http, server, location

Запрещает сжатие ответа методом gzip для запросов с полями заголовка “User-Agent”, совпадающими с заданными регулярными выражениями.

Специальная маска “*msie6*” соответствует регулярному выражению “*MSIE [4-6].*”, но работает быстрее. Из этой маски исключается “*MSIE 6.0; … SV1*”.

gzip_http_version

Синтаксис gzip_http_version 1.0 | 1.1;

Умолчание gzip_http_version 1.1;

Контекст http, server, location

Устанавливает минимальную HTTP-версию запроса, необходимую для сжатия ответа.

gzip_min_length

Синтаксис gzip_min_length *длина*;

Умолчание gzip_min_length 20;

Контекст http, server, location

Устанавливает минимальную длину ответа, который будет сжиматься методом gzip. Длина определяется только из поля “Content-Length” заголовка ответа.

gzip_proxied

Синтаксис gzip_proxied off | expired | no-cache | no-store | private | no_last_modified | no_etag | auth | any …;

Умолчание gzip_proxied off;

Контекст http, server, location

Разрешает или запрещает сжатие ответа методом gzip для проксированных запросов в зависимости от запроса и ответа. То, что запрос проксированный, определяется на основании наличия поля “Via” в заголовке запроса. В директиве можно указать одновременно несколько параметров:

off	запрещает сжатие для всех проксированных запросов, игнорируя остальные параметры;
expired	разрешает сжатие, если в заголовке ответа есть поле “Expires” со значением, запрещающим кэширование;
no-cache	разрешает сжатие, если в заголовке ответа есть поле “Cache-Control” с параметром “no-cache”;
no-store	разрешает сжатие, если в заголовке ответа есть поле “Cache-Control” с параметром “no-store”;
private	разрешает сжатие, если в заголовке ответа есть поле “Cache-Control” с параметром “private”;
no_last_modified	разрешает сжатие, если в заголовке ответа нет поля “Last-Modified”;
no_etag	разрешает сжатие, если в заголовке ответа нет поля “ETag”;
auth	разрешает сжатие, если в заголовке запроса есть поле “Authorization”;
any	разрешает сжатие для всех проксированных запросов.

gzip_types

Синтаксис gzip_types mime-*тип* …;

Умолчание gzip_types text/html;

Контекст http, server, location

Разрешает сжатие ответа методом gzip для указанных MIME-типов в дополнение к “*text/html*”. Специальное значение “*” соответствует любому MIME-типу. Ответы с типом “*text/html*” сжимаются всегда.

gzip_vary

Синтаксис gzip_vary on | off;

Умолчание gzip_vary off;

Контекст http, server, location

Разрешает или запрещает выдавать в ответе поле заголовка “Vary: Accept-Encoding”, если активны директивы gzip, gzip_static или gunzip.

Встроенные переменные

$gzip_ratio

достигнутый коэффициент сжатия — отношение размера исходного ответа к размеру сжатого.

Модуль http_gzip_static

Позволяет отдавать вместо обычного файла предварительно сжатый файл с таким же именем и с расширением “.gz”.

По умолчанию этот модуль не собирается, его сборку необходимо разрешить с помощью конфигурационного параметра *--with-http_gzip_static_module*.

Пример конфигурации

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

Директивы

gzip_static

Синтаксис gzip_static on | off | always;

Умолчание gzip_static off;

Контекст http, server, location

Разрешает (on) или запрещает (off) проверку готового сжатого файла. При использовании также учитываются директивы gzip_http_version, gzip_proxied, gzip_disable и gzip_vary.

Со значением always во всех случаях будет использоваться сжатый файл, без проверки поддержки на стороне клиента. Это полезно, если на диске всё равно нет несжатых файлов, или используется модуль http_gunzip.
Сжимать файлы можно с помощью программы gzip или совместимой с ней. Желательно, чтобы дата и время модификации исходного и сжатого файлов совпадали.

Модуль http_headers

Позволяет выдавать поля заголовка “Expires” и “Cache-Control”, а также добавлять произвольные поля в заголовок ответа.

Пример конфигурации

```
expires    24h;
expires    modified +24h;
```

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

Директивы

add_header

Синтаксис add_header *имя значение* [always];

Умолчание —

Контекст http, server, location, if в location

Добавляет указанное поле в заголовок ответа при условии, что код ответа равен 200, 201 (1.3.10), 204, 206, 301, 302, 303, 304, 307 или 308. В значении параметра можно использовать переменные.

Директив *add_header* может быть несколько. Директивы наследуются с предыдущего уровня конфигурации при условии, что на данном уровне не описаны свои директивы *add_header*.

Если указан параметр always, то поле заголовка будет добавлено независимо от кода ответа.

add_trailer

Синтаксис add_trailer *имя значение* [always];

Умолчание —

Контекст http, server, location, if в location

Добавляет указанное поле в конец ответа при условии, что код ответа равен 200, 201, 206, 301, 302, 303, 307 или 308. В значении можно использовать переменные.

Директив *add_trailer* может быть несколько. Директивы наследуются с предыдущего уровня конфигурации при условии, что на данном уровне не описаны свои директивы *add_trailer*.

Если указан параметр always, то указанное поле будет добавлено независимо от кода ответа.

expires

Синтаксис expires [modified] *время*;
expires epoch | max | off;

Умолчание expires off;

Контекст http, server, location, if в location

Разрешает или запрещает добавлять или менять поля “Expires” и “Cache-Control” в заголовке ответа при условии, что код ответа равен 200, 201, 204, 206, 301, 302, 303, 304, 307 или 308. В качестве параметра можно задать положительное или отрицательное время.

Время в поле “Expires” получается как сумма текущего времени и времени, заданного в директиве. Если используется параметр modified, то время получается как сумма времени модификации файла и времени, заданного в директиве.

Кроме того, с помощью префикса “@” можно задать время суток:

```
expires @15h30m;
```

Содержимое поля “Cache-Control” зависит от знака заданного времени:

- отрицательное время — “Cache-Control: no-cache”.
- положительное или равное нулю время — “Cache-Control: max-age=`t`”, где *t* это время в секундах, заданное в директиве.

epoch	задаёт время “Thu, 01 Jan 1970 00:00:01 GMT” (1 января 1970 00:00:01 GMT) для поля “Expires” и “no-cache” для поля “Cache-Control”.
max	задаёт время “Thu, 31 Dec 2037 23:55:55 GMT” (31 декабря 2037 23:55:55 GMT) для поля “Expires” и 10 лет для поля “Cache-Control”.
off	запрещает добавлять или менять поля “Expires” и “Cache-Control” в заголовке ответа.

В значении последнего параметра можно использовать переменные:

```
map $sent_http_content_type $expires {
    default          off;
    application/pdf 42d;
    ~image/          max;
}

expires $expires;
```

Модуль http_image_filter

Фильтр для преобразования изображений в форматах JPEG, GIF, PNG и WebP.

По умолчанию этот модуль не собирается, его сборку необходимо разрешить с помощью конфигурационного параметра --with-http_image_filter_module.

Примечание

Для сборки и работы этого модуля необходима библиотека libgd. Рекомендуется использовать самую последнюю версию библиотеки.

Для преобразования изображений в формате WebP библиотека libgd должна быть собрана с поддержкой WebP.

Пример конфигурации

```
location /img/ {
    proxy_pass   http://backend;
    image_filter resize 150 100;
    image_filter rotate 90;
    error_page   415 = /empty;
}

location = /empty {
    empty_gif;
}
```

Директивы

image_filter

Синтаксис image_filter off;
image_filter test;
image_filter size;
image_filter rotate 90 | 180 | 270;
image_filter resize *ширина высота*;
image_filter crop *ширина высота*;

Умолчание image_filter off;

Контекст location

Задаёт тип преобразования изображения:

off	отключает обработку данным модулем во вложенном location.
test	проверяет, что ответ действительно является изображением в формате JPEG, GIF, PNG или WebP. В противном случае возвращается ошибка 415 (Unsupported Media Type).
size	**выдаёт информацию об изображении в формате JSON, например:** { «img» : { «width»: 100, «height»: 100, «type»: «gif» } } **В случае ошибки выдаётся** {}
rotate 90|180|270	поворачивает изображение против часовой стрелки на указанное число градусов. В значении параметра допустимо использование переменных. Можно использовать как отдельно, так и совместно с преобразованиями *resize* и *crop*.
resize *ширина высота*	пропорционально уменьшает изображение до указанных размеров. Если требуется уменьшить только по одному измерению, то в качестве второго можно указать “-”. В случае ошибки сервер возвращает код 415 (Unsupported Media Type). В

	значениях параметров допустимо использование переменных. При использовании совместно с *rotate*, поворот изображения происходит **после** уменьшения размеров изображения.
crop *ширина высота*	пропорционально уменьшает изображение до размера большей стороны и обрезает лишние края по другой стороне. Если требуется уменьшить только по одному измерению, то в качестве второго можно указать “-”. В случае ошибки сервер возвращает код 415 (Unsupported Media Type). В значениях параметров допустимо использование переменных. При использовании совместно с rotate, поворот изображения происходит **до** уменьшения размеров изображения.

image_filter_buffer

Синтаксис image_filter_buffer *размер*;

Умолчание image_filter_buffer 1M;

Контекст http, server, location

Задаёт максимальный размер буфера для чтения изображения. При превышении размера сервер вернёт ошибку 415 (Unsupported Media Type).

image_filter_interlace

Синтаксис image_filter_interlace on | off;

Умолчание image_filter_interlace off;

Контекст http, server, location

Если включено, то итоговые изображения будут с чересстрочностью. В случае JPEG итоговые изображения будут в формате “progressive JPEG”.

image_filter_jpeg_quality

Синтаксис image_filter_jpeg_quality *качество*;

Умолчание image_filter_jpeg_quality 75;

Контекст http, server, location

Задаёт желаемое качество преобразованного изображения в формате JPEG. Допустимые значения находятся в диапазоне от 1 до 100. Меньшим значениям обычно соответствует худшее качество изображения и меньший объём передаваемых данных. Максимальное рекомендуемое значение — 95. В значении параметра допустимо использование переменных.

image_filter_sharpen

Синтаксис image_filter_sharpen *процент*;

Умолчание image_filter_sharpen 0;

Контекст http, server, location

Повышает резкость итогового изображения. Процент резкости может быть больше 100. Значение 0 отключает повышение резкости. В значении параметра допустимо использование переменных.

image_filter_transparency

Синтаксис image_filter_transparency on | off;

Умолчание image_filter_transparency on;

Контекст http, server, location

Определяет, сохранять ли прозрачность при обработке изображений в формате GIF и в формате PNG с цветами, заданными палитрой. Потеря прозрачности позволяет получить более качественное изображение. Прозрачность альфа-канала в формате PNG сохраняется всегда.

image_filter_webp_quality

Синтаксис image_filter_webp_quality *качество*;

Умолчание image_filter_webp_quality 80;

Контекст http, server, location

Задаёт желаемое качество преобразованного изображения в формате WebP. Допустимые значения находятся в диапазоне от 1 до 100. Меньшим значениям обычно соответствует худшее качество изображения и меньший объём передаваемых данных. В значении параметра допустимо использование переменных.

Модуль http_random_index

Обслуживает запросы, оканчивающиеся слэшом (‘/’). Такие запросы также могут обслуживаться модулями Модуль http_autoindex и Модуль http_random_index.

Пример конфигурации

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

Директивы

index

Синтаксис index *файл* …;

Умолчание index index.html;

Контекст http, server, location

Определяет файлы, которые будут использоваться в качестве индекса. В имени файла можно использовать переменные. Наличие файлов проверяется в порядке их перечисления. В конце списка может стоять файл с абсолютным путём. Пример:

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

Необходимо иметь в виду, что при использовании индексного файла делается внутреннее перенаправление и запрос может быть обработан уже в другом location'е. Например, в такой конфигурации:

```
location = / {
    index index.html;
}

location / {
#    ...
}
```

запрос "/" будет фактически обработан во втором location'е как "*/index.html*".

Модуль http_js

Позволяет задавать обработчики location и переменных на njs — подмножестве языка JavaScript.

Пример конфигурации

```
http {
    js_import http.js;

    js_set $foo     http.foo;
    js_set $summary http.summary;
    js_set $hash    http.hash;

    resolver 10.0.0.1;

    server {
        listen 8000;

        location / {
            add_header X-Foo $foo;
            js_content http.baz;
        }

        location = /summary {
            return 200 $summary;
        }

        location = /hello {
            js_content http.hello;
        }

        location = /fetch {
            js_content                   http.fetch;
            js_fetch_trusted_certificate /path/to/ISRG_Root_X1.pem;
        }

        location = /crypto {
            add_header Hash $hash;
            return     200;
```

```
        }
    }
}
```

Файл http.js:

```
function foo(r) {
    r.log("hello from foo() handler");
    return "foo";
}

function summary(r) {
    var a, s, h;

    s = "JS summary\n\n";

    s += "Method: " + r.method + "\n";
    s += "HTTP version: " + r.httpVersion + "\n";
    s += "Host: " + r.headersIn.host + "\n";
    s += "Remote Address: " + r.remoteAddress + "\n";
    s += "URI: " + r.uri + "\n";

    s += "Headers:\n";
    for (h in r.headersIn) {
        s += "  header '" + h + "' is '" + r.headersIn[h] + "'\n";
    }

    s += "Args:\n";
    for (a in r.args) {
        s += "  arg '" + a + "' is '" + r.args[a] + "'\n";
    }

    return s;
}

function baz(r) {
    r.status = 200;
    r.headersOut.foo = 1234;
    r.headersOut['Content-Type'] = "text/plain; charset=utf-8";
    r.headersOut['Content-Length'] = 15;
    r.sendHeader();
    r.send("nginx");
    r.send("java");
    r.send("script");

    r.finish();
}

function hello(r) {
    r.return(200, "Hello world!");
}

async function fetch(r) {
    let results = await Promise.all([ngx.fetch('https://google.com/'),
```

```
                                    ngx.fetch('https://google.ru/')]);

    r.return(200, JSON.stringify(results, undefined, 4));
}

async function hash(r) {
    let hash = await crypto.subtle.digest('SHA-512', r.headersIn.host);
    r.setReturnValue(Buffer.from(hash).toString('hex'));
}

export default {foo, summary, baz, hello, fetch, hash};
```

Директивы

js_body_filter

Синтаксис js_body_filter *функция* | *модуль.функция* [buffer_type = *строка* | *буфер*];

Умолчание —

Контекст location, if in location, limit_except

Задаёт функцию njs в качестве фильтра тела ответа. Функция фильтра вызывается для каждого блока данных тела ответа со следующими аргументами:

r	объект HTTP request
data	входящий блок данных может быть строкой или буфером в зависимости от значения *buffer_type*, по умолчанию является строкой.
flags	объект со следующими свойствами: last — логическое значение true — если данные являются последним буфером.

Функция фильтра может передавать свою модифицированную версию входящего блока данных следующему фильтру тела ответа при помощи вызова r.sendBuffer(). Пример преобразования букв в нижний регистр в теле ответа:

```
function filter(r, data, flags) {
    r.sendBuffer(data.toLowerCase(), flags);
}
```

Для отмены фильтра (блоки данных будут передаваться клиенту без вызова *js_body_filter*), можно использовать r.done().

Если функция фильтра изменяет длину тела ответа, то необходимо очистить заголовок ответа “Content-Length” (если присутствует) в js_header_filter, чтобы применить поблочное кодирование.

Примечание

Так как обработчик *js_body_filter* должен сразу возвращать результат, то поддерживаются только синхронные операции, Таким образом, асинхронные операции, например r.subrequest() или setTimeout(), не поддерживаются.

js_content

Синтаксис js_content *функция* | *модуль.функция*;

Умолчание —

Контекст location, if in location, limit_except

Задаёт функцию njs в качестве обработчика содержимого location. Можно ссылаться на функцию модуля.

js_fetch_buffer_size

Синтаксис js_fetch_buffer_size *размер*;

Умолчание js_fetch_buffer_size 16k;

Контекст http, server, location

Задаёт размер буфера, который будет использоваться для чтения и записи для Fetch API.

js_fetch_ciphers

Синтаксис js_fetch_ciphers *шифры*;

Умолчание js_fetch_ciphers HIGH:!aNULL:!MD5;

Контекст http, server, location

Описывает разрешённые шифры для HTTPS-соединений при помощи Fetch API. Шифры задаются в формате, поддерживаемом библиотекой OpenSSL.

Полный список можно посмотреть с помощью команды “*openssl ciphers*”.

js_fetch_max_response_buffer_size

Синтаксис js_fetch_max_response_buffer_size *размер*;

Умолчание js_fetch_max_response_buffer_size 1m;

Контекст http, server, location

Задаёт максимальный размер ответа, полученного при помощи Fetch API.

js_fetch_protocols

Синтаксис js_fetch_protocols [TLSv1] [TLSv1.1] [TLSv1.2] [TLSv1.3];

Умолчание js_fetch_protocols TLSv1 TLSv1.1 TLSv1.2;

Контекст http, server, location

Разрешает указанные протоколы для HTTPS-соединений при помощи Fetch API.

js_fetch_timeout

Синтаксис js_fetch_timeout *время*;

Умолчание js_fetch_timeout 60s;

Контекст http, server, location

Задаёт таймаут при чтении и записи при помощи Fetch API. Таймаут устанавливается не на всю передачу ответа, а только между двумя операциями чтения. Если по истечении этого времени данные не передавались, соединение закрывается.

js_fetch_trusted_certificate

Синтаксис js_fetch_trusted_certificate *файл*;

Умолчание —

Контекст http, server, location

Задаёт файл с доверенными сертификатами CA в формате PEM, используемыми при проверке HTTPS-сертификата при помощи Fetch API.

js_fetch_verify

Синтаксис js_fetch_verify on | off;

Умолчание js_fetch_verify on;

Контекст http, server, location

Разрешает или запрещает проверку сертификата HTTPS-сервера при помощи Fetch API.

js_fetch_verify_depth

Синтаксис js_fetch_verify_depth *число*;

Умолчание js_fetch_verify_depth 100;

Контекст http, server, location

Устанавливает глубину проверки в цепочке HTTPS-сертификатов при помощи Fetch API.

js_header_filter

Синтаксис js_header_filter *функция* | *модуль.функция*;

Умолчание —

Контекст location, if in location, limit_except

Задаёт функцию njs в качестве фильтра заголовка ответа. Директива позволяет менять произвольные поля заголовка ответа.

Примечание

Так как обработчик *js_header_filter* должен сразу возвращать результат, то поддерживаются только синхронные операции. Таким образом, асинхронные операции, например r.subrequest() или setTimeout(), не поддерживаются.

js_import

Синтаксис js_import *модуль.js* | *имя_экспорта* from *модуль.js*;

Умолчание —

Контекст http, server, location

Импортирует модуль, позволяющий задавать обработчики location и переменных на njs. *Имя_экспорта* является пространством имён при доступе к функциям модуля. Если *имя_экспорта* не задано, то пространством имён будет являться имя модуля.

```
js_import http.js;
```

В примере при доступе к экспорту в качестве пространства имён используется имя модуля *http*. Если импортируемый модуль экспортирует *foo()*, то для доступа используется *http.foo*.

Директив *js_import* может быть несколько.

js_path

Синтаксис js_path *путь*;

Умолчание —

Контекст http, server, location

Задаёт дополнительный путь для модулей njs.

js_preload_object

Синтаксис js_preload_object *имя.json* | *имя from файл.json*;

Умолчание —

Контекст http, server, location

Предварительно загружает неизменяемый объект во время конфигурации. *Имя* используется в качестве имени глобальной переменной, через которую объект доступен в коде njs. Если *имя* не указано, то будет использоваться имя файла.

```
js_preload_object map.json;
```

В примере *map* используется в качестве имени во время доступа к предварительно загруженному объекту.

Директив *js_preload_object* может быть несколько.

js_set

Синтаксис js_set *$переменная функция* | *модуль.функция*;

Умолчание —

Контекст http, server, location

Задаёт функцию njs для указанной переменной. Можно ссылаться на функцию модуля.

Функция вызывается в момент первого обращения к переменной для данного запроса. Точный момент вызова функции зависит от фазы, в которой происходит обращение к переменной. Это можно использовать для реализации дополнительной логики, не относящейся к вычислению переменной. Например, если переменная указана в директиве log_format, то её обработчик не будет выполняться до фазы записи в лог. Этот обработчик также может использоваться для выполнения процедур непосредственно перед освобождением запроса.

Примечание

Так как обработчик *js_set* должен сразу возвращать результат, то поддерживаются только синхронные операции. Таким образом, асинхронные операции, например r.subrequest() или setTimeout(), не поддерживаются.

js_var

Синтаксис js_var *$переменная* [*значение*];

Умолчание —

Контекст http, server, location

Объявляет перезаписываемую переменную. В качестве значения можно использовать текст, переменные и их комбинации. Переменная не перезаписывается после перенаправления, в отличие от переменных, созданных при помощи директивы set.

Аргумент запроса

Каждый HTTP-обработчик njs получает один аргумент, объект запроса.

Модуль http_limit_conn

Позволяет ограничить число соединений по заданному ключу, в частности, число соединений с одного IP-адреса.

Учитываются не все соединения, а лишь те, в которых имеются запросы, обрабатываемые сервером, и заголовок запроса уже прочитан.

Пример конфигурации

```
http {
    limit_conn_zone $binary_remote_addr zone=addr:10m;

    ...

    server {

        ...

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

Директивы

limit_conn

Синтаксис limit_conn *зона число*;

Умолчание —

Контекст http, server, location

Задаёт зону разделяемой памяти и максимально допустимое число соединений для одного значения ключа. При превышении этого числа в ответ на запрос сервер вернёт ошибку. Например, директивы

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

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

разрешают одновременно обрабатывать не более одного соединения с одного IP-адреса.

Примечание

В HTTP/2 и SPDY каждый параллельный запрос считается отдельным соединением.

Директив limit_conn может быть несколько. Например, следующая конфигурация ограничивает число соединений с сервером с одного клиентского IP-адреса и в то же время ограничивает общее число соединений с виртуальным сервером:

```
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;
}
```

Директивы наследуются с предыдущего уровня конфигурации при условии, что на данном уровне не описаны свои директивы *limit_conn*.

limit_conn_dry_run

Синтаксис limit_conn_dry_run on | off;

Умолчание limit_conn_dry_run off;

Контекст http, server, location

Включает режим пробного запуска. В данном режиме число соединений не ограничивается, однако в зоне разделяемой памяти текущее число избыточных соединений учитывается как обычно.

limit_conn_log_level

Синтаксис limit_conn_log_level info | notice | warn | error;

Умолчание limit_conn_log_level error;

Контекст http, server, location

Задаёт желаемый уровень записи в лог случаев ограничения числа соединений.

limit_conn_status

Синтаксис limit_conn_status *код*;

Умолчание limit_conn_status 503;

Контекст http, server, location

Позволяет переопределить код ответа, используемый при отклонении запросов.

limit_conn_zone

Синтаксис limit_conn_zone *ключ* zone = *название:размер*;

Умолчание —

Контекст http

Задаёт параметры зоны разделяемой памяти, которая хранит состояние для разных значений ключа. Состояние в частности содержит текущее число соединений. В качестве ключа можно использовать текст, переменные и их комбинации. Запросы с пустым значением ключа не учитываются.

Пример использования:

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

Здесь в качестве ключа используется IP-адрес клиента. Обратите внимание, что вместо переменной $remote_addr использована переменная $binary_remote_addr.

Длина значения переменной $remote_addr может колебаться от 7 до 15 байт, при этом размер хранимого состояния составляет либо 32, либо 64 байта на 32-битных платформах и всегда 64 байта на 64-битных.

Длина значения переменной $binary_remote_addr всегда равна 4 байтам для IPv4-адресов или 16 байтам для IPv6-адресов. При этом размер состояния всегда равен 32 или 64 байтам на 32-битных платформах и 64 байтам на 64-битных.

В зоне размером 1 мегабайт может разместиться около 32 тысяч состояний размером 32 байта или 16 тысяч состояний размером 64 байта. При переполнении зоны в ответ на последующие запросы сервер будет возвращать ошибку.

Встроенные переменные

$limit_conn_status

хранит результат ограничения числа соединений: *PASSED*, *REJECTED* или *REJECTED_DRY_RUN*

Модуль http_limit_req

Позволяет ограничить скорость обработки запросов по заданному ключу или, как частный случай, скорость обработки запросов, поступающих с одного IP-адреса. Ограничение обеспечивается с помощью метода “leaky bucket”.

Пример конфигурации

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

    ...

    server {

        ...

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

Директивы

limit_req

Синтаксис limit_req zone = *название* [burst = *число*] [nodelay | delay = *число*];

Умолчание —

Контекст http, server, location

Задаёт зону разделяемой памяти (zone) и максимальный размер всплеска запросов (burst). Если скорость поступления запросов превышает описанную в зоне, то их обработка задерживается так, чтобы запросы обрабатывались с заданной скоростью. Избыточные запросы задерживаются до тех пор, пока их число не превысит максимальный размер всплеска. При превышении запрос завершается с ошибкой. По умолчанию максимальный размер всплеска равен нулю. Например, директивы

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

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

позволяют в среднем не более 1 запроса в секунду со всплесками не более 5 запросов.

Если же избыточные запросы в пределах лимита всплесков задерживать не требуется, то следует использовать параметр nodelay:

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

Параметр delay задаёт лимит, по достижении которого избыточные запросы задерживаются. Значение по умолчанию равно нулю и означает, что задерживаются все избыточные запросы.

Директив *limit_req* может быть несколько. Например, следующая конфигурация ограничивает скорость обработки запросов, поступающих с одного IP-адреса, и в то же время ограничивает скорость обработки запросов одним виртуальным сервером:

```
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;
}
```

Директивы наследуются с предыдущего уровня конфигурации при условии, что на данном уровне не описаны свои директивы *limit_req*.

limit_req_dry_run

Синтаксис limit_req_dry_run on | off;

Умолчание limit_req_dry_run off;

Контекст http, server, location

Включает режим пробного запуска. В данном режиме скорость обработки запросов не ограничивается, однако в зоне разделяемой памяти текущее число избыточных запросов учитывается как обычно.

limit_req_log_level

Синтаксис limit_req_log_level info | notice | warn | error;

Умолчание limit_req_log_level error;

Контекст http, server, location

Задаёт желаемый уровень записи в лог случаев отказа в обработке запросов при превышении скорости и случаев задержек при обработке запроса. Задержки записываются в лог с уровнем на единицу меньшим, чем отказы, например, если указано *limit_req_log_level notice;*, то задержки будут записываться в лог на уровне *info*.

limit_req_status

Синтаксис limit_req_status *код*;

Умолчание limit_req_status 503;

Контекст http, server, location

Позволяет переопределить код ответа, используемый при отклонении запросов.

limit_req_zone

Синтаксис limit_req_zone *ключ* zone = *название:размер* rate = *скорость*;

Умолчание —

Контекст http

Задаёт параметры зоны разделяемой памяти, которая хранит состояние для разных значений ключа. Состояние в частности хранит текущее число избыточных запросов. В качестве ключа можно использовать текст, переменные и их комбинации. Запросы с пустым значением ключа не учитываются.

Пример использования:

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

В данном случае состояния хранятся в зоне “*one*” размером 10 мегабайт, и средняя скорость обработки запросов для этой зоны не может превышать 1 запроса в секунду.

В качестве ключа используется IP-адрес клиента. Обратите внимание, что вместо переменной $remote_addr используется переменная $binary_remote_addr.

Длина значения переменной *$binary_remote_addr* всегда равна 4 байтам для IPv4-адресов или 16 байтам для IPv6-адресов. При этом размер состояния всегда равен 64 байтам на 32-битных платформах и 128 байтам на 64-битных платформах.

В зоне размером 1 мегабайт может разместиться около 16 тысяч состояний размером 64 байта или около 8 тысяч состояний размером 128 байт.

При переполнении зоны удаляется наименее востребованное состояние. Если и это не позволяет создать новое состояние, запрос завершается с ошибкой.

Скорость rate задаётся в запросах в секунду (r/s). Если же нужна скорость меньше одного запроса в секунду, то она задаётся в запросах в минуту (r/m), например, ползапроса в секунду — это 30r/m.

Встроенные переменные

$limit_req_status

хранит результат ограничения скорости поступления запросов: *PASSED*, *DELAYED*, *REJECTED*, *DELAYED_DRY_RUN* или *REJECTED_DRY_RUN*

Модуль http_log

Модуль записывает логи запросов в указанном формате.

Логи записываются в контексте location’а, где заканчивается обработка. Это может быть location, отличный от первоначального, если в процессе обработки запроса происходит внутреннее перенаправление.

Пример конфигурации

```
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;
```

Директивы

access_log

Синтаксис access_log *путь* [*формат* [buffer = *размер*] [gzip[= *степень*]] [flush = *время*] [if = *условие*]]; access_log off;

Умолчание access_log logs/access.log combined;

Контекст http, server, location, if в location, limit_except

Задаёт *путь*, *формат* и настройки буферизованной записи в лог. На одном уровне конфигурации может использоваться несколько логов. Запись в syslog настраивается указанием префикса *“syslog:”* в первом параметре. Специальное значение *off* отменяет все директивы *access_log* для текущего уровня. Если формат не указан, то используется предопределённый формат *“combined”*.

Если задан размер буфера с помощью параметра *buffer* или указан параметр *gzip*, то запись будет буферизованной.

Примечание

Размер буфера должен быть не больше размера атомарной записи в дисковый файл. Для FreeBSD этот размер неограничен.

При включённой буферизации данные записываются в файл:

- если очередная строка лога не помещается в буфер;
- если данные в буфере находятся дольше интервала времени, заданного параметром *flush*;
- при переоткрытии лог-файла или завершении рабочего процесса.

Если задан параметр *gzip*, то буфер будет сжиматься перед записью в файл. Степень сжатия может быть задана в диапазоне от *1* (быстрее, но хуже сжатие) до *9* (медленнее, но лучше сжатие). По умолчанию используются буфер размером *64K* байт и степень сжатия *1*. Данные сжимаются атомарными блоками, и в любой момент времени лог-файл может быть распакован или прочитан с помощью утилиты *“zcat”*.

Пример:

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

Примечание

Для поддержки gzip-сжатия логов Angie должен быть собран с библиотекой *zlib*.

В пути файла можно использовать переменные, но такие логи имеют некоторые ограничения:

- пользователь, с правами которого работают рабочие процессы, должен иметь права на создание файлов в каталоге с такими логами;
- не работает буферизация;
- файл открывается для каждой записи в лог и сразу же после записи закрывается. Следует однако иметь в виду, что поскольку дескрипторы часто используемых файлов могут храниться в кэше, то при ротации логов в течение времени, заданного параметром *valid* директивы open_log_file_cache, запись может продолжаться в старый файл.
- при каждой записи в лог проверяется существование корневого каталога для запроса — если этот каталог не существует, то лог не создаётся. Поэтому root и access_log нужно описывать на одном уровне конфигурации:

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

Параметр *if* включает условную запись в лог. Запрос не будет записываться в лог, если результатом вычисления условия является *“0”* или пустая строка. В следующем примере запросы с кодами ответа 2xx и 3xx не будут записываться в лог:

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

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

log_format

Синтаксис log_format *имя* [escape=default | json | none] *строка* …;

Умолчание log_format combined "...";

Контекст http

Задаёт формат лога.

Параметр *escape* позволяет задать экранирование символов *json* или *default* в переменных, по умолчанию используется *default*. Значение *none* отключает экранирование символов.

При использовании *default* символы “"”, “\”, а также символы со значениями меньше 32 или больше 126 экранируются как “\xXX”. Если значение переменной не найдено, то в качестве значения в лог будет записываться дефис “-”.

При использовании *json* экранируются все символы, недопустимые в JSON строках: символы “"” и “\” экранируются как “\"” и “\\”, символы со значениями меньше 32 экранируются как “\n”, “\r”, “\t”, “\b”, “\f” или “\u00XX”.

Строки заголовка, переданные клиенту, начинаются с префикса *sent_http_*, например, *$sent_http_content_range*.

В конфигурации всегда существует предопределённый формат *combined*:

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

open_log_file_cache

Синтаксис open_log_file_cache max = *N* [inactive = *время*] [min_uses = *N*] [valid = *время*]; open_log_file_cache off;

Умолчание open_log_file_cache off;

Контекст http, server, location

Задаёт кэш, в котором хранятся дескрипторы файлов часто используемых логов, имена которых заданы с использованием переменных. Параметры:

max	задаёт максимальное число дескрипторов в кэше; при переполнении кэша наименее востребованные (LRU) дескрипторы закрываются
inactive	задаёт время, после которого закэшированный дескриптор закрывается, если к нему не было обращений в течение этого времени; по умолчанию 10 секунд
min_uses	задаёт минимальное число использований файла в течение времени, заданного параметром *inactive*, после которого дескриптор файла будет оставаться открытым в кэше; по умолчанию 1
valid	задаёт, через какое время нужно проверять, что файл ещё существует под тем же именем; по умолчанию 60 секунд
off	запрещает кэш

Пример использования:

```
open_log_file_cache max=1000 inactive=20s valid=1m min_uses=2;
```

Модуль http_map

Создаёт переменные, значения которых зависят от значений других переменных.

Пример конфигурации

```
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;
}
```

Директивы

map

Синтаксис map *строка $переменная* { … };

Умолчание —

Контекст http

Создаёт новую переменную, значение которой зависит от значений одной или более исходных переменных, указанных в первом параметре.

Примечание

Поскольку переменные вычисляются только в момент использования, само по себе наличие даже большого числа объявлений переменных “map” не влечёт за собой никаких дополнительных расходов на обработку запросов.

Параметры внутри блока *map* задают соответствие между исходными и результирующими значениями.

Исходные значения задаются строками или регулярными выражениями.

Строки проверяются без учёта регистра.

Перед регулярным выражением ставится символ “~”, если при сравнении следует учитывать регистр символов, либо символы “~*”, если регистр символов учитывать не нужно. Регулярное выражение может содержать именованные и позиционные выделения, которые могут затем использоваться в других директивах совместно с результирующей переменной.

Если исходное значение совпадает с именем одного из специальных параметров, описанных ниже, перед ним следует поставить символ “”.

В качестве результирующего значения можно указать текст, переменную и их комбинации.

Также поддерживаются следующие специальные параметры:

default *значение*	задаёт результирующее значение, если исходное значение не совпадает ни с одним из перечисленных. Если параметр *default* не указан, результирующим значением по умолчанию будет пустая строка.
hostnames	указывает, что в качестве исходных значений можно использовать маску для первой или последней части имени хоста. Этот параметр следует указывать перед списком значений.

Например,

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

Вместо двух записей

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

можно использовать одну:

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

include *файл*	включает файл со значениями. Включений может быть несколько.
volatile	указывает, что переменная не кэшируется.

Если исходному значению соответствует несколько из указанных вариантов, например, одновременно подходят и маска, и регулярное выражение, будет выбран первый подходящий вариант в следующем порядке приоритета:

- строковое значение без маски
- самое длинное строковое значение с маской в начале, например “**.example.com*”
- самое длинное строковое значение с маской в конце, например “*mail.**”
- первое подходящее регулярное выражение (в порядке следования в конфигурационном файле)
- значение по умолчанию (*default*)

map_hash_bucket_size

Синтаксис map_hash_bucket_size *размер*;

Умолчание map_hash_bucket_size 32|64|128;

Контекст http

Задаёт размер корзины в хэш-таблицах для переменных map. Значение по умолчанию зависит от размера строки кэша процессора. Подробнее настройка хэш-таблиц обсуждается отдельно.

map_hash_max_size

Синтаксис map_hash_max_size *размер*;

Умолчание map_hash_max_size 2048;

Контекст http

Задаёт максимальный размер хэш-таблиц для переменных map. Подробнее настройка хэш-таблиц обсуждается отдельно.

Модуль http_memcached

Позволяет получать ответ из сервера memcached. Ключ задаётся в переменной $memcached_key. Ответ в memcached должен быть предварительно помещён внешним по отношению к Angie способом.

Пример конфигурации

```
server {
    location / {
        set             $memcached_key "$uri?$args";
        memcached_pass  host:11211;
        error_page      404 502 504 = @fallback;
    }

    location @fallback {
        proxy_pass      http://backend;
    }
}
```

Директивы

memcached_bind

Синтаксис memcached_bind *адрес* [transparent] | off;

Умолчание —

Контекст http, server, location

Задаёт локальный IP-адрес с необязательным портом, который будет использоваться в исходящих соединениях с сервером memcached. В значении параметра допустимо использование переменных. Специальное значение off отменяет действие унаследованной с предыдущего уровня конфигурации директивы *memcached_bind*, позволяя системе самостоятельно выбирать локальный IP-адрес и порт.

Параметр transparent позволяет задать нелокальный IP-адрес, который будет использоваться в исходящих соединениях с проксируемым сервером, например, реальный IP-адрес клиента:

```
memcached_bind $remote_addr transparent;
```

Для работы параметра обычно требуется запустить рабочие процессы Angie с привилегиями суперпользователя. В Linux это не требуется, так как если указан параметр *transparent*, то рабочие процессы наследуют *capability CAP_NET_RAW* из главного процесса.

Внимание

Необходимо настроить таблицу маршрутизации ядра для перехвата сетевого трафика с сервера memcached.

memcached_buffer_size

Синтаксис memcached_buffer_size *размер*;

Умолчание memcached_buffer_size 4k|8k;

Контекст http, server, location

Задаёт размер буфера, в который будет читаться первая часть ответа, получаемого от сервера memcached. Ответ синхронно передаётся клиенту сразу же по мере его поступления.

memcached_connect_timeout

Синтаксис memcached_connect_timeout *время*;

Умолчание memcached_connect_timeout 60s;

Контекст http, server, location

Задаёт таймаут для установления соединения с сервером memcached. Необходимо иметь в виду, что этот таймаут обычно не может превышать 75 секунд.

memcached_gzip_flag

Синтаксис memcached_gzip_flag *флаг*;

Умолчание —

Контекст http, server, location

Включает проверку указанного флага в ответе сервера memcached и установку поля “Content-Encoding” заголовка ответа в “gzip”, если этот флаг установлен.

memcached_next_upstream

Синтаксис memcached_next_upstream error | timeout | invalid_response | not_found | off …;

Умолчание memcached_next_upstream error timeout;

Контекст http, server, location

Определяет, в каких случаях запрос будет передан следующему в группе upstream серверу:

error	произошла ошибка соединения с сервером, передачи ему запроса

	или чтения заголовка ответа сервера;
timeout	произошёл таймаут во время соединения с сервером, передачи ему запроса или чтения заголовка ответа сервера;
invalid_response	сервер вернул пустой или неверный ответ;
not_found	сервер не нашёл ответ;
off	запрещает передачу запроса следующему серверу.

Примечание

Необходимо понимать, что передача запроса следующему серверу возможна только при условии, что клиенту ещё ничего не передавалось. То есть, если ошибка или таймаут возникли в середине передачи ответа клиенту, то действие директивы на такой запрос не распространяется.

Директива также определяет, что считается неудачной попыткой работы с сервером.

error timeout invalid_response	всегда считаются неудачными попытками, даже если они не указаны в директиве
not_found	никогда не считается неудачными попытками

Передача запроса следующему серверу может быть ограничена по количеству попыток и по времени.

memcached_next_upstream_timeout

Синтаксис memcached_next_upstream_timeout *время*;

Умолчание memcached_next_upstream_timeout 0;

Контекст http, server, location

Ограничивает время, в течение которого возможна передача запроса следующему серверу.

0	отключает это ограничение

memcached_next_upstream_tries

Синтаксис memcached_next_upstream_tries *число*;

Умолчание memcached_next_upstream_tries 0;

Контекст http, server, location

Ограничивает число допустимых попыток для передачи запроса следующему серверу.

0	отключает это ограничение

memcached_pass

Синтаксис memcached_pass *адрес*;

Умолчание —

Контекст location, if в location

Задаёт адрес сервера memcached. Адрес может быть указан в виде доменного имени или IP-адреса, и порта:

```
memcached_pass localhost:11211;
```

или в виде пути UNIX-сокета:

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

Если доменному имени соответствует несколько адресов, то все они будут использоваться по очереди (round-robin). Кроме того, в качестве адреса можно указать группу серверов.

memcached_read_timeout

Синтаксис memcached_read_timeout *время*;

Умолчание memcached_read_timeout 60s;

Контекст http, server, location

Задаёт таймаут при чтении ответа сервера memcached. Таймаут устанавливается не на всю передачу ответа, а только между двумя операциями чтения. Если по истечении этого времени сервер memcached ничего не передаст, соединение закрывается.

memcached_send_timeout

Синтаксис memcached_send_timeout *время*;

Умолчание memcached_send_timeout 60s;

Контекст http, server, location

Задаёт таймаут при передаче запроса серверу memcached. Таймаут устанавливается не на всю передачу запроса, а только между двумя операциями записи. Если по истечении этого времени сервер memcached не примет новых данных, соединение закрывается.

memcached_socket_keepalive

Синтаксис memcached_socket_keepalive on | off;

Умолчание memcached_socket_keepalive off;

Контекст http, server, location

Конфигурирует поведение “TCP keepalive” для исходящих соединений к проксируемому серверу.

""	По умолчанию для сокета действуют настройки операционной системы.
on	для сокета включается параметр *SO_KEEPALIVE*

Встроенные переменные

$memcached_key

Задаёт ключ для получения ответа из сервера memcached.

Модуль http_mirror

Позволяет зеркалировать исходный запрос при помощи создания фоновых зеркалирующих подзапросов. Ответы на зеркалирующие подзапросы игнорируются.

Пример конфигурации

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

location = /mirror {
    internal;
    proxy_pass http://test_backend$request_uri;
}
```

Директивы

mirror

Синтаксис mirror *uri* | off;

Умолчание mirror off;

Контекст http, server, location

Задаёт URI, на который будет зеркалироваться исходный запрос. На одном уровне конфигурации может быть задано несколько зеркал.

mirror_request_body

Синтаксис mirror_request_body on | off;

Умолчание mirror_request_body on;

Контекст http, server, location

Определяет, зеркалировать ли тело запроса клиента. Если включено, то тело запроса клиента будет прочитано перед созданием зеркалирующих подзапросов. В этом случае небуферизованное проксирование тела запроса клиента, задаваемое директивами proxy_request_buffering, fastcgi_request_buffering, scgi_request_buffering и uwsgi_request_buffering, будет отключено.

```
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;
}
```

Модуль http_mp4

Обеспечивает серверную поддержку псевдо-стриминга для файлов в формате MP4. Такие файлы обычно имеют расширения .mp4, .m4v и .m4a.

Псевдо-стриминг работает в паре с совместимым медиаплеером. Плеер посылает серверу HTTP-запрос с указанием точки времени старта в аргументе start строки запроса (время задаётся в секундах), а сервер в ответ посылает поток, у которого начальная позиция соответствует запрошенному времени, например:

```
http://example.com/elephants_dream.mp4?start=238.88
```

Это позволяет в любой момент времени выполнить произвольное позиционирование, а также начать воспроизведение с середины временной шкалы.

В форматах, основанных на H.264, метаданные, необходимые для поддержки позиционирования, хранятся в так называемом “moov-атоме”. Это часть файла, которая содержит индексную информацию для всего файла.

До начала воспроизведения плееру необходимо прочитать метаданные. Для этого он отсылает специальный запрос с аргументом *start=0*. Многие кодирующие программы добавляют метаданные в конец файла. Это неоптимально для псевдо-стриминга, поскольку плееру потребуется загрузить файл целиком прежде чем начать воспроизведение. Если метаданные находятся в начале файла, Angie достаточно начать отправлять в ответ содержимое файла. Если же метаданные находятся в конце файла, потребуется прочитать весь файл и подготовить новый поток, в котором метаданные предшествуют медийным данным. Это требует дополнительного процессорного времени, памяти и дискового ввода/вывода, поэтому лучше заранее подготовить исходный файл для псевдо-стриминга, нежели делать это для каждого запроса.

Модуль также поддерживает аргумент *end* HTTP-запроса, задающий время окончания воспроизведения потока. Аргумент *end* задаётся совместно с аргументом *start* или самостоятельно:

```
http://example.com/elephants_dream.mp4?start=238.88&end=555.55
```

Для запроса с ненулевыми аргументами *start* или *end* Angie считывает из файла метаданные, готовит поток с запрошенным диапазоном и отправляет его клиенту. Это тоже требует дополнительных ресурсов, как указано выше.

Если аргумент *start* указывает на видеокадр, не являющийся ключевым, то начало такого видео может воспроизводиться с ошибками. В этом случае к запрашиваемому видео

могут быть добавлены ближайший к точке *start* ключевой кадр и все промежуточные кадры между ними. При воспроизведении эти кадры будут скрыты при помощи edit-листа.

Если запрос, обрабатываемый этим модулем, не содержит аргументов *start* и *end*, дополнительные ресурсы не тратятся, а файл отсылается непосредственно как статический ресурс. Некоторые плееры также поддерживают запросы с указанием диапазона запрашиваемых байт (byte-range requests), для них этот модуль не требуется.

По умолчанию этот модуль не собирается, его сборку необходимо разрешить с помощью конфигурационного параметра *--with-http_mp4_module*.

Примечание

Если ранее использовался сторонний модуль mp4, следует его отключить.

Схожая поддержка псевдо-стриминга для FLV-файлов обеспечивается модулем http_flv.

Пример конфигурации

```
location /video/ {
    mp4;
    mp4_buffer_size     1m;
    mp4_max_buffer_size 5m;
}
```

Директивы

mp4

Синтаксис mp4;

Умолчание —

Контекст location

Включает в содержащем location обработку этим модулем.

mp4_buffer_size

Синтаксис mp4_buffer_size *размер*;

Умолчание mp4_buffer_size 512K;

Контекст http, server, location

Задаёт начальный размер буфера, используемого при обработке MP4-файлов.

mp4_max_buffer_size

Синтаксис mp4_max_buffer_size *размер*;

Умолчание mp4_max_buffer_size 10M;

Контекст http, server, location

В ходе обработки метаданных может понадобиться буфер большего размера. Его размер не может превышать указанного, иначе Angie вернёт серверную ошибку 500 (Internal Server Error) и запишет в лог следующее сообщение:

```
«/some/movie/file.mp4» mp4 moov atom is too large: 12583268, you may want to increase mp4_max_buffer_size
```

mp4_start_key_frame

Синтаксис	mp4_start_key_frame on | off;
Умолчание	mp4_start_key_frame off;
Контекст	http, server, location

Включает режим, в котором видео всегда начинается с ключевого видеокадра. Если аргумент start не указывает на ключевой кадр, то первоначальные кадры будут скрыты при помощи mp4 edit-листа. Edit-листы поддерживаются большинством плееров и браузеров включая Chrome, Safari, QuickTime и ffmpeg, частично поддерживаются в Firefox.

Модуль http_perl

Модуль позволяет писать обработчики location и переменных на Perl, а также вставлять вызовы Perl в SSI.

По умолчанию этот модуль не собирается, его сборку необходимо разрешить с помощью конфигурационного параметра *--with-http_perl_module*.

Внимание

Для сборки этого модуля необходим Perl версии 5.6.1 и выше. Компилятор C должен быть совместим с тем, которым был собран Perl.

Известные проблемы

Для того чтобы во время переконфигурации Perl перекомпилировал изменённые модули, его нужно собрать с параметрами *-Dusemultiplicity=yes* или *-Dusethreads=yes*. Кроме того, чтобы во время работы Perl терял меньше памяти, его нужно собрать с параметром *-Dusemymalloc=no*. Узнать значения этих параметров у уже собранного Perl можно так (в примере приведены желаемые значения параметров):

```
$ perl -V:usemultiplicity -V:usemymalloc |br|
usemultiplicity='define'; |br|
usemymalloc='n';
```

Необходимо учитывать, что после пересборки Perl с новыми параметрами *-Dusemultiplicity=yes* или *-Dusethreads=yes* придётся также пересобрать и все бинарные модули Perl, так как они просто перестанут работать с новым Perl.

Возможно, главный процесс, а вслед за ним и рабочие процессы, будут увеличиваться в размерах при каждой переконфигурации. Когда главный процесс вырастет до

неприемлемых размеров, можно воспользоваться процедурой обновления сервера на лету, не меняя при этом сам исполняемый файл.

Если модуль Perl выполняет длительную операцию, например, определяет адрес по имени, соединяется с другим сервером, делает запрос к базе данных, то на это время все остальные запросы, обслуживаемые данным рабочим процессом, не будут обрабатываться. Поэтому рекомендуется ограничиться операциями, время исполнения которых короткое и предсказуемое, например, обращение к локальной файловой системе.

Пример конфигурации

```
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;
        }
    }
```

Модуль perl/lib/hello.pm:

```
package hello;

use angie;

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__
```

Директивы

perl

Синтаксис perl *модуль* :: *функция* | 'sub { … }';

Умолчание —

Контекст location, limit_except

Устанавливает обработчик Perl для данного location.

perl_modules

Синтаксис perl_modules *путь*;

Умолчание —

Контекст http

Задаёт дополнительный путь для модулей Perl.

perl_require

Синтаксис perl_require *модуль*;

Умолчание —

Контекст http

Задаёт имя модуля, который будет подгружаться при каждой переконфигурации. Директив *perl_require* может быть несколько.

perl_set

Синтаксис perl_set *$переменная модуль* :: *функция* | 'sub { … }';

Умолчание —

Контекст http

Устанавливает обработчик Perl для указанной переменной.

Вызов Perl из SSI

Формат команды SSI с вызовом Perl следующий:

```
<!--# perl sub="модуль::функция" arg="параметр1" arg="параметр2" ...
-->
```

Методы объекта запроса $r

$r->args

возвращает аргументы запроса.

$r->filename

возвращает имя файла, соответствующее URI запроса.

$r->has_request_body (*обработчик*)

возвращает 0, если в запросе нет тела. Если же в запросе есть тело, то устанавливается указанный обработчик и возвращается 1. По окончании чтения тела запроса Angie вызовет установленный обработчик. Обратите внимание, что нужно передавать ссылку на функцию обработчика. Пример:

```
package hello;

use angie;

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

разрешает использовать диапазоны байт (byte ranges) при передаче ответа.

$r->discard_request_body

указывает Angie игнорировать тело запроса.

$r->header_in (*поле*)

возвращает значение заданного поля в заголовке запроса клиента.

$r->header_only

определяет, нужно ли передавать клиенту только заголовок ответа или весь ответ.

$r->header_out (*поле, значение*)

устанавливает значение для заданного поля в заголовке ответа.

$r->internal_redirect (*uri*)

делает внутреннее перенаправление на указанный uri. Перенаправление происходит уже после завершения обработчика Perl. Метод принимает экранированные URI и поддерживает перенаправления в именованные location.

$r->log_error (*код_ошибки, сообщение*)

записывает указанное сообщение в error_log. Если *код_ошибки* ненулевой, то к сообщению будет добавлен код ошибки и её описание.

$r->print (текст, …)

метод передаёт клиенту данные.

$r->request_body

возвращает тело запроса клиента при условии, что тело не записано во временный файл. Для того чтобы тело запроса клиента гарантированно находилось в памяти, нужно ограничить его размер с помощью client_max_body_size и задать достаточной размер для буфера client_body_buffer_size.

$r->request_body_file

возвращает имя файла, в котором хранится тело запроса клиента. По завершению обработки файл необходимо удалить. Для того чтобы тело запроса клиента всегда записывалось в файл, следует включить client_body_in_file_only.

$r->request_method

возвращает HTTP-метод запроса клиента.

$r->remote_addr

возвращает IP-адрес клиента.

$r->flush

немедленно передаёт данные клиенту.

$r->sendfile (*имя* [, *смещение* [, длина]])

передаёт клиенту содержимое указанного файла. Необязательные параметры задают начальное смещение и длину передаваемых данных. Непосредственно передача данных происходит уже после завершения обработчика Perl.

$r->send_http_header ([тип])

передаёт клиенту заголовок ответа. Необязательный параметр тип устанавливает значение поля “Content-Type” в заголовке ответа. Пустая строка в качестве типа запрещает передачу поля “Content-Type”.

$r->status (код)

устанавливает код ответа.

$r->sleep (*миллисекунды, обработчик*)

устанавливает указанный обработчик и останавливает обработку запроса на заданное время. Angie в это время продолжает обрабатывать другие запросы. По истечении указанного времени Angie вызовет установленный обработчик. Обратите внимание, что нужно передавать ссылку на функцию обработчика. Для передачи данных между обработчиками следует использовать $r->variable(). Пример:

```
package hello;

use agie;

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 (*текст*)

декодирует текст, заданный в виде “%XX”.

$r->uri

возвращает URI запроса.

$r->variable (*имя* [, *значение*])

возвращает или устанавливает значение указанной переменной. Переменные локальны для каждого запроса.

Модуль http_proxy

Позволяет передавать запросы другому (проксируемому) серверу.

Пример конфигурации

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

Директивы

proxy_bind

Синтаксис proxy_bind *адрес* [transparent] | off;

Умолчание —

Контекст http, server, location

Задаёт локальный IP-адрес с необязательным портом, который будет использоваться в исходящих соединениях с проксируемым сервером. В значении параметра допустимо использование переменных. Специальное значение off отменяет действие унаследованной с предыдущего уровня конфигурации директивы *proxy_bind*, позволяя системе самостоятельно выбирать локальный IP-адрес и порт.

Параметр transparent позволяет задать нелокальный IP-адрес, который будет использоваться в исходящих соединениях с проксируемым сервером, например, реальный IP-адрес клиента:

```
proxy_bind $remote_addr transparent;
```

Для работы параметра обычно требуется запустить рабочие процессы Angie с привилегиями суперпользователя. В Linux это не требуется, так как если указан параметр *transparent*, то рабочие процессы наследуют *capability CAP_NET_RAW* из главного процесса.

Внимание

Необходимо настроить таблицу маршрутизации ядра для перехвата сетевого трафика с проксируемого сервера.

proxy_buffer_size

Синтаксис proxy_buffer_size *размер*;

Умолчание proxy_buffer_size 4k|8k;

Контекст http, server, location

Задаёт размер буфера, в который будет читаться первая часть ответа, получаемого от проксируемого сервера. В этой части ответа находится, как правило, небольшой заголовок ответа. По умолчанию размер одного буфера равен размеру страницы памяти. В зависимости от платформы это или 4K, или 8K, однако его можно сделать меньше.

proxy_buffering

Синтаксис proxy_buffering *размер*;

Умолчание proxy_buffering on;

Контекст http, server, location

on	Angie принимает ответ проксируемого сервера как можно быстрее, сохраняя его в буферы, заданные директивами proxy_buffer_size и proxy_buffers. Если ответ не вмещается целиком в память, то его часть может быть записана на диск во временный файл. Запись во временные файлы контролируется директивами proxy_max_temp_file_size и proxy_temp_file_write_size.
off	Ответ синхронно передаётся клиенту сразу же по мере его поступления. Angie не пытается считать весь ответ проксируемого сервера. Максимальный размер данных, который Angie может принять от сервера за один раз, задаётся директивой proxy_buffer_size.

Буферизация может быть также включена или выключена путём передачи значения “yes” или “no” в поле “X-Accel-Buffering” заголовка ответа. Эту возможность можно запретить с помощью директивы proxy_ignore_headers.

proxy_buffers

Синтаксис proxy_buffers *число размер*;

Умолчание proxy_buffers 8 4k|8k;

Контекст http, server, location

Задаёт число и размер буферов для одного соединения, в которые будет читаться ответ, получаемый от проксируемого сервера.
По умолчанию размер одного буфера равен размеру страницы. В зависимости от платформы это или 4K, или 8K.

proxy_busy_buffers_size

Синтаксис proxy_busy_buffers_size *размер*;

Умолчание proxy_buffers 8k|16k;

Контекст http, server, location

При включённой буферизации ответов проксируемого сервера, ограничивает суммарный размер буферов, которые могут быть заняты для отправки ответа клиенту, пока ответ ещё не прочитан целиком. Оставшиеся буферы тем временем могут использоваться для чтения ответа и, при необходимости, буферизации части ответа во временный файл. По умолчанию размер ограничен величиной двух буферов, заданных директивами proxy_buffer_size и proxy_buffers.

proxy_cache

Синтаксис proxy_cache *зона* | off;

Умолчание proxy_cache off;

Контекст http, server, location

Задаёт зону разделяемой памяти, используемой для кэширования. Одна и та же зона может использоваться в нескольких местах. В значении параметра можно использовать переменные.

off	запрещает кэширование, унаследованное с предыдущего уровня конфигурации.

proxy_cache_background_update

Синтаксис proxy_cache_background_update on | off;

Умолчание proxy_cache_background_update off;

Контекст http, server, location

Позволяет запустить фоновый подзапрос для обновления просроченного элемента кэша, в то время как клиенту возвращается устаревший закэшированный ответ.

Внимание

Использование устаревшего закэшированного ответа в момент его обновления должно быть разрешено.

proxy_cache_bypass

Синтаксис proxy_cache_bypass …;

Умолчание —

Контекст http, server, location

Задаёт условия, при которых ответ не будет браться из кэша. Если значение хотя бы одного из строковых параметров непустое и не равно “0”, то ответ не берётся из кэша:

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

Можно использовать совместно с директивой proxy_no_cache.

proxy_cache_convert_head

Синтаксис proxy_cache_convert_head on | off;

Умолчание proxy_cache_convert_head on;

Контекст http, server, location

Разрешает или запрещает преобразование метода “HEAD” в “GET” для кэширования. Если преобразование выключено, то необходимо, чтобы ключ кэширования включал в себя $request_method.

proxy_cache_key

Синтаксис proxy_cache_key *строка*;

Умолчание proxy_cache_key $scheme$proxy_host$request_uri;

Контекст http, server, location

Задаёт ключ для кэширования, например,

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

По умолчанию значение директивы близко к строке

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

proxy_cache_lock

Синтаксис proxy_cache_lock on | off;

Умолчание proxy_cache_lock off;

Контекст http, server, location

Если включено, одновременно только одному запросу будет позволено заполнить новый элемент кэша, идентифицируемый согласно директиве proxy_cache_key, передав запрос на проксируемый сервер. Остальные запросы этого же элемента будут либо ожидать появления ответа в кэше, либо освобождения блокировки этого элемента, в течение времени, заданного директивой proxy_cache_lock_timeout.

proxy_cache_lock_age

Синтаксис proxy_cache_lock_age *время*;

Умолчание proxy_cache_lock_age 5s;

Контекст http, server, location

Если последний запрос, переданный на проксируемый сервер для заполнения нового элемента кэша, не завершился за указанное время, на проксируемый сервер может быть передан ещё один запрос.

proxy_cache_lock_timeout

Синтаксис proxy_cache_lock_timeout *время*;

Умолчание proxy_cache_lock_timeout 5s;

Контекст http, server, location

Задаёт таймаут для proxy_cache_lock. По истечении указанного времени запрос будет передан на проксируемый сервер, однако ответ не будет закэширован.

proxy_cache_max_range_offset

Синтаксис proxy_cache_max_range_offset *число*;

Умолчание —

Контекст http, server, location

Задаёт смещение в байтах для запросов с указанием диапазона запрашиваемых байт (byte-range requests). Если диапазон находится за указанным смещением, range-запрос будет передан на проксируемый сервер и ответ не будет закэширован.

proxy_cache_methods

Синтаксис proxy_cache_methods GET | HEAD | POST …;

Умолчание proxy_cache_methods GET HEAD;

Контекст http, server, location

Если метод запроса клиента указан в этой директиве, то ответ будет закэширован. Методы “GET” и “HEAD” всегда добавляются в список, но тем не менее рекомендуется перечислять их явно. См. также директиву proxy_no_cache.

proxy_cache_min_uses

Синтаксис proxy_cache_min_uses *число*;

Умолчание proxy_cache_min_uses 1;

Контекст http, server, location

Задаёт число запросов, после которого ответ будет закэширован.

proxy_cache_path

Синтаксис proxy_cache_path *путь* [levels = *уровни*] [use_temp_path=on|off] keys_zone = *имя:размер* [inactive = *время*] [max_size = *размер*] [min_free = *размер*] [manager_files = *число*] [manager_sleep = *время*] [manager_threshold = *время*] [loader_files = *число*] [loader_sleep = *время*] [loader_threshold = *время*];

Умолчание —

Контекст http

Задаёт путь и другие параметры кэша. Данные кэша хранятся в файлах. Именем файла в кэше является результат функции MD5 от ключа кэширования.

Параметр levels задаёт уровни иерархии кэша: можно задать от 1 до 3 уровней, на каждом уровне допускаются значения 1 или 2.

Например, при использовании

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

имена файлов в кэше будут такого вида:

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

Кэшируемый ответ сначала записывается во временный файл, а потом этот файл переименовывается. Временные файлы и кэш могут располагаться на разных файловых системах. Однако нужно учитывать, что в этом случае вместо дешёвой операции переименовывания в пределах одной файловой системы файл копируется с одной файловой системы на другую. Поэтому лучше, если кэш будет находиться на той же файловой системе, что и каталог с временными файлами.

Какой из каталогов будет использоваться для временных файлов, определяется параметром use_temp_path.

on	Если параметр не задан или установлен в значение “on”, будет использоваться каталог, задаваемый директивой proxy_temp_path для данного *location*

off	временные файлы будут располагаться непосредственно в каталоге кэша

Кроме того, все активные ключи и информация о данных хранятся в зоне разделяемой памяти, имя и размер которой задаются параметром keys_zone. Зоны размером в 1 мегабайт достаточно для хранения около 8 тысяч ключей.

Если к данным кэша не обращаются в течение времени, заданного параметром inactive, то данные удаляются, независимо от их свежести. По умолчанию *inactive* равен 10 минутам.

Специальный процесс **cache manager** следит за максимальным размером кэша, а также за минимальным объёмом свободного места на файловой системе с кэшем, и удаляет наименее востребованные данные при превышении максимального размера кэша или недостаточном объёме свободного места. Удаление данных происходит итерациями.

max_size	максимальное пороговое значение размера кэша
min_free	минимальное пороговое значение объёма свободного места на файловой системе с кэшем
manager_files	максимальное количество удаляемых элементов кэша за одну итерацию По умолчанию *100*
manager_threshold	ограничивает время работы одной итерации По умолчанию *200* миллисекунд
manager_sleep	время, в течение которого выдерживается пауза между итерациями По умолчанию *50* миллисекунд

Через минуту после старта Angie активируется специальный процесс **cache loader**, который загружает в зону кэша информацию о ранее закэшированных данных, хранящихся на файловой системе. Загрузка также происходит итерациями.

loader_files	максимальное количество элементов кэша к загрузке в одну итерацию По умолчанию *100*
loader_threshold	ограничивает время работы одной итерации По умолчанию *200* миллисекунд
loader_sleep	время, в течение которого выдерживается пауза между итерациями По умолчанию *50* миллисекунд

proxy_cache_revalidate

Синтаксис proxy_cache_revalidate on | off;

Умолчание proxy_cache_revalidate off;

Контекст http, server, location

Разрешает ревалидацию просроченных элементов кэша при помощи условных запросов с полями заголовка “If-Modified-Since” и “If-None-Match”.

proxy_cache_use_stale

Синтаксис proxy_cache_use_stale error | timeout | invalid_header | updating | http_500 | http_502 | http_503 | http_504 | http_403 | http_404 | http_429 | off …;

Умолчание proxy_cache_use_stale off;

Контекст http, server, location

Определяет, в каких случаях можно использовать устаревший закэшированный ответ. Параметры директивы совпадают с параметрами директивы proxy_next_upstream.

error	Позволяет использовать устаревший закэшированный ответ при невозможности выбора проксированного сервера для обработки запроса.
updating	Дополнительный параметр, разрешает использовать устаревший закэшированный ответ, если на данный момент он уже обновляется. Это позволяет минимизировать число обращений к проксированным серверам при обновлении закэшированных данных.

Использование устаревшего закэшированного ответа может также быть разрешено непосредственно в заголовке ответа на определённое количество секунд после того, как ответ устарел:

- Расширение stale-while-revalidate поля заголовка “Cache-Control” разрешает использовать устаревший закэшированный ответ, если на данный момент он уже обновляется.
- Расширение stale-if-error поля заголовка “Cache-Control” разрешает использовать устаревший закэшированный ответ в случае ошибки.

Примечание

Такой способ менее приоритетен, чем задание параметров директивы.

Чтобы минимизировать число обращений к проксированным серверам при заполнении нового элемента кэша, можно воспользоваться директивой proxy_cache_lock.

proxy_cache_valid

Синтаксис proxy_cache_valid [*код* …] *время*;

Умолчание —

Контекст http, server, location

Задаёт время кэширования для разных кодов ответа. Например, директивы

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

задают время кэширования 10 минут для ответов с кодами 200 и 302 и 1 минуту для ответов с кодом 404.

Если указано только время кэширования,

```
proxy_cache_valid 5m;
```

то кэшируются только ответы 200, 301 и 302.

Кроме того, можно кэшировать любые ответы с помощью параметра any:

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

Примечание

Параметры кэширования могут также быть заданы непосредственно в заголовке ответа. Такой способ приоритетнее, чем задание времени кэширования с помощью директивы.

- Поле заголовка “X-Accel-Expires” задаёт время кэширования ответа в секундах. Значение *0* запрещает кэшировать ответ. Если значение начинается с префикса @, оно задаёт абсолютное время в секундах с начала эпохи, до которого ответ может быть закэширован.
- Если в заголовке нет поля “X-Accel-Expires”, параметры кэширования определяются по полям заголовка “Expires” или “Cache-Control”.
- Ответ, в заголовке которого есть поле “Set-Cookie”, не будет кэшироваться.
- Ответ, в заголовке которого есть поле “Vary” со специальным значением “*”, не будет кэшироваться. Ответ, в заголовке которого есть поле “Vary” с другим значением, будет закэширован с учётом соответствующих полей заголовка запроса.

Обработка одного или более из этих полей заголовка может быть отключена при помощи директивы proxy_ignore_headers.

proxy_connect_timeout

Синтаксис proxy_connect_timeout *время*;

Умолчание proxy_connect_timeout 60s;

Контекст http, server, location

Задаёт таймаут для установления соединения с проксированным сервером. Необходимо иметь в виду, что этот таймаут обычно не может превышать 75 секунд.

proxy_cookie_domain

Синтаксис proxy_cookie_domain off; proxy_cookie_domain *домен замена*;

Умолчание proxy_cookie_domain off;

Контекст http, server, location

Задаёт текст, который нужно изменить в атрибуте domain полей “Set-Cookie” заголовка ответа проксируемого сервера. Предположим, проксируемый сервер вернул поле заголовка “Set-Cookie” с атрибутом “domain=localhost”. Директива

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

перепишет данный атрибут в виде “domain=example.org”.

Точка в начале строк *домен* и *замена*, а равно как и в атрибуте *domain* игнорируется. Регистр значения не имеет.

В строках *домен* и *замена* можно использовать переменные:

```
proxy_cookie_domain www.$host $host;
```

Директиву также можно задать при помощи регулярных выражений. При этом *домен* должен начинаться с символа “~”. Регулярное выражение может содержать именованные и позиционные выделения, а *замена* ссылаться на них:

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

На одном уровне может быть указано несколько директив *proxy_cookie_domain*:

```
proxy_cookie_domain localhost example.org;
proxy_cookie_domain ~\.([a-z]+\.[a-z]+)$ $1;
```

Если к куке могут быть применены несколько директив, будет выбрана первая из них.

Параметр off отменяет действие унаследованных с предыдущего уровня конфигурации директив *proxy_cookie_domain*.

proxy_cookie_flags

Синтаксис proxy_cookie_flags off | *кука* [*флаг* …];

Умолчание proxy_cookie_flags off;

Контекст http, server, location

Задаёт один или несколько флагов для куки. В качестве куки можно использовать текст, переменные и их комбинации. В качестве флага можно использовать текст, переменные и их комбинации.

Параметры *secure*, *httponly*, *samesite=strict*, *samesite=lax*, *samesite=none* добавляют соответствующие флаги.

Параметры *nosecure*, *nohttponly*, *nosamesite* удаляют соответствующие флаги.

Куки также можно задать при помощи регулярных выражений. При этом кука должна начинаться с символа “~”.

На одном уровне конфигурации может быть указано несколько директив *proxy_cookie_flags*:

```
proxy_cookie_flags one httponly;
proxy_cookie_flags ~ nosecure samesite=strict;
```

Если к куке могут быть применены несколько директив, будет выбрана первая из них. В данном примере флаг *httponly* добавляется к куке *one*, для остальных кук добавляется флаг *samesite=strict* и удаляется флаг *secure*.

Параметр off отменяет действие всех директив *proxy_cookie_flags* на данном уровне.

proxy_cookie_path

Синтаксис proxy_cookie_path off; proxy_cookie_path *путь замена*;

Умолчание proxy_cookie_path off;

Контекст http, server, location

Задаёт текст, который нужно изменить в атрибуте path полей “Set-Cookie” заголовка ответа проксируемого сервера. Предположим, проксируемый сервер вернул поле заголовка “Set-Cookie” с атрибутом “path=/two/some/uri/”. Директива

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

перепишет данный атрибут в виде “path=/some/uri/”.

В строках *путь* и *замена* можно использовать переменные:

```
proxy_cookie_path $uri /some$uri;
```

Директиву также можно задать при помощи регулярных выражений. При этом *путь* должен начинаться либо с символа “~”, если при сравнении следует учитывать регистр символов, либо с символов “~*”, если регистр символов учитывать не нужно. Регулярное выражение может содержать именованные и позиционные выделения, а *замена* ссылаться на них:

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

На одном уровне может быть указано несколько директив proxy_cookie_path:

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

Если к куке могут быть применены несколько директив, будет выбрана первая из них.

Параметр off отменяет действие унаследованных с предыдущего уровня конфигурации директив *proxy_cookie_path*.

proxy_force_ranges

Синтаксис proxy_force_ranges off;

Умолчание proxy_force_ranges off;

Контекст http, server, location

Включает поддержку диапазонов запрашиваемых байт (byte-range) для кэшированных и некэшированных ответов проксируемого сервера вне зависимости от наличия поля “Accept-Ranges” в заголовках этих ответов.

proxy_headers_hash_bucket_size

Синтаксис proxy_headers_hash_bucket_size *размер*;

Умолчание proxy_headers_hash_bucket_size 64;

Контекст http, server, location

Задаёт размер корзины для хэш-таблиц, используемых директивами proxy_hide_header и proxy_set_header. Подробнее настройка хэш-таблиц обсуждается отдельно.

proxy_headers_hash_max_size

Синтаксис proxy_headers_hash_max_size *размер*;

Умолчание proxy_headers_hash_max_size 512;

Контекст http, server, location

Задаёт максимальный размер хэш-таблиц, используемых директивами proxy_hide_header и proxy_set_header. Подробнее настройка хэш-таблиц обсуждается отдельно.

proxy_hide_header

Синтаксис proxy_hide_header *поле*;

Умолчание —

Контекст http, server, location

По умолчанию Angie не передаёт клиенту поля заголовка “Date”, “Server”, “X-Pad” и “X-Accel-…” из ответа проксированного сервера. Директива *proxy_hide_header* задаёт дополнительные поля, которые не будут передаваться. Если же передачу полей нужно разрешить, можно воспользоваться директивой proxy_pass_header.

proxy_http_version

Синтаксис proxy_http_version 1.0 | 1.1;

Умолчание proxy_http_version 1.0;

Контекст http, server, location

Задаёт версию протокола HTTP для проксирования. По умолчанию используется версия 1.0. Для работы постоянных соединений рекомендуется версия 1.1.

proxy_ignore_client_abort

Синтаксис proxy_ignore_client_abort on | off;

Умолчание proxy_ignore_client_abort off;

Контекст http, server, location

Определяет, закрывать ли соединение с проксируемым сервером в случае, если клиент закрыл соединение, не дождавшись ответа.

proxy_ignore_headers

Синтаксис proxy_ignore_headers *поле* …;

Умолчание —

Контекст http, server, location

Запрещает обработку некоторых полей заголовка из ответа проксированного сервера. В директиве можно указать поля “X-Accel-Redirect”, “X-Accel-Expires”, “X-Accel-Limit-Rate”, “X-Accel-Buffering”, “X-Accel-Charset”, “Expires”, “Cache-Control”, “Set-Cookie” и “Vary”.

Если не запрещено, обработка этих полей заголовка заключается в следующем:

- “X-Accel-Expires”, “Expires”, “Cache-Control”, “Set-Cookie” и “Vary” задают параметры кэширования ответа;
- “X-Accel-Redirect” производит внутреннее перенаправление на указанный URI;
- “X-Accel-Limit-Rate” задаёт ограничение скорости передачи ответа клиенту;
- “X-Accel-Buffering” включает или выключает буферизацию ответа;
- “X-Accel-Charset” задаёт желаемую кодировку ответа.

proxy_intercept_errors

Синтаксис proxy_intercept_errors on | off;

Умолчание proxy_intercept_errors off;

Контекст http, server, location

Определяет, передавать ли клиенту проксированные ответы с кодом больше либо равным 300, или же перехватывать их и перенаправлять на обработку Angie с помощью директивы error_page.

proxy_limit_rate

Синтаксис proxy_limit_rate *скорость*;

Умолчание proxy_limit_rate 0;

Контекст http, server, location

Ограничивает скорость чтения ответа от проксируемого сервера. *Скорость* задаётся в байтах в секунду.

0	отключает ограничение скорости

Примечание

Ограничение устанавливается на запрос, поэтому, если Angie одновременно откроет два соединения к проксируемому серверу, суммарная скорость будет вдвое выше заданного ограничения. Ограничение работает только в случае, если включена буферизация ответов проксируемого сервера.

proxy_max_temp_file_size

Синтаксис proxy_max_temp_file_size *размер*;

Умолчание proxy_max_temp_file_size 1024m;

Контекст http, server, location

Если включена буферизация ответов проксируемого сервера, и ответ не вмещается целиком в буферы, заданные директивами proxy_buffer_size и proxy_buffers, часть ответа может быть записана во временный файл. Эта директива задаёт максимальный размер временного файла. Размер данных, сбрасываемых во временный файл за один раз, задаётся директивой proxy_temp_file_write_size.

0	отключает возможность буферизации ответов во временные файлы

Примечание

Данное ограничение не распространяется на ответы, которые будут закэшированы или сохранены на диске.

proxy_method

Синтаксис proxy_method *метод*;

Умолчание —

Контекст http, server, location

Задаёт HTTP-метод, который будет использоваться в передаваемых на проксируемый сервер запросах вместо метода из клиентского запроса. В значении параметра допустимо использование переменных.

proxy_next_upstream

Синтаксис proxy_next_upstream error | timeout | invalid_header | http_500 | http_502 | http_503 | http_504 | http_403 | http_404 | http_429 | non_idempotent | off …;

Умолчание proxy_next_upstream error timeout;

Контекст http, server, location

Определяет, в каких случаях запрос будет передан следующему в группе upstream серверу:

error	произошла ошибка соединения с сервером, передачи ему запроса или чтения заголовка ответа сервера;
timeout	произошёл таймаут во время соединения с сервером, передачи ему запроса или чтения заголовка ответа сервера;
invalid_header	сервер вернул пустой или неверный ответ;
http_500	сервер вернул ответ с кодом 500;
http_502	сервер вернул ответ с кодом 502;

http_503	сервер вернул ответ с кодом 503;
http_504	сервер вернул ответ с кодом 504;
http_403	сервер вернул ответ с кодом 403;
http_404	сервер вернул ответ с кодом 404;
http_429	сервер вернул ответ с кодом 429;
non_idempotent	обычно запросы с неидемпотентным методом (*POST*, *LOCK*, *PATCH*) не передаются на другой сервер, если запрос серверу группы уже был отправлен; включение параметра явно разрешает повторять подобные запросы;
off	запрещает передачу запроса следующему серверу.

Примечание

Необходимо понимать, что передача запроса следующему серверу возможна только при условии, что клиенту ещё ничего не передавалось. То есть, если ошибка или таймаут возникли в середине передачи ответа клиенту, то действие директивы на такой запрос не распространяется.

Директива также определяет, что считается неудачной попыткой работы с сервером.

error timeout invalid_header	всегда считаются неудачными попытками, даже если они не указаны в директиве
http_500 http_502 http_503 http_504 http_429	считаются неудачными попытками, только если они указаны в директиве
http_403 http_404	никогда не считаются неудачными попытками

Передача запроса следующему серверу может быть ограничена по количеству попыток и по времени.

proxy_next_upstream_timeout

Синтаксис proxy_next_upstream_timeout *время*;

Умолчание proxy_next_upstream_timeout 0;

Контекст http, server, location

Ограничивает время, в течение которого возможна передача запроса следующему серверу.

0	отключает это ограничение

proxy_next_upstream_tries

Синтаксис proxy_next_upstream_tries *число*;

Умолчание proxy_next_upstream_tries 0;

Контекст http, server, location

Ограничивает число допустимых попыток для передачи запроса следующему серверу.

0	отключает это ограничение

proxy_no_cache

Синтаксис proxy_no_cache *строка* …;

Умолчание —

Контекст http, server, location

Задаёт условия, при которых ответ не будет сохраняться в кэш. Если значение хотя бы одного из строковых параметров непустое и не равно “0”, то ответ не будет сохранён:

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

Можно использовать совместно с директивой proxy_cache_bypass.

proxy_pass

Синтаксис proxy_pass URL;

Умолчание —

Контекст location, if в location, limit_except

Задаёт протокол и адрес проксируемого сервера, а также необязательный URI, на который должен отображаться location. В качестве протокола можно указать “*http*” или “*https*”. Адрес может быть указан в виде доменного имени или IP-адреса, и необязательного порта:

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

или в виде пути UNIX-сокета, который указывается после слова “*unix*” и заключается в двоеточия:

```
proxy_pass http://unix:/tmp/backend.socket:/uri/;
```

Если доменному имени соответствует несколько адресов, то все они будут использоваться по очереди (round-robin). Кроме того, в качестве адреса можно указать группу серверов.

В значении параметра можно использовать переменные. В этом случае, если адрес указан в виде доменного имени, имя ищется среди описанных групп серверов и если не найдено, то определяется с помощью resolver’а.

URI запроса передаётся на сервер так:

- Если директива *proxy_pass* указана **с URI**, то при передаче запроса серверу часть нормализованного URI запроса, соответствующая location, заменяется на URI, указанный в директиве:

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

- Если директива proxy_pass указана **без URI**, то при обработке первоначального запроса на сервер передаётся URI запроса в том же виде, в каком его прислал клиент, а при обработке изменённого URI - нормализованный URI запроса целиком:

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

В ряде случаев часть URI запроса, подлежащую замене, выделить невозможно:

- Если *location* задан регулярным выражением, а также в именованных *location*'ах.

В этих случаях *proxy_pass* следует указывать без URI.

- Если внутри проксируемого *location* с помощью директивы rewrite изменяется URI, и именно с этой конфигурацией будет обрабатываться запрос (*break*):

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

В этом случае URI, указанный в директиве, игнорируется, и на сервер передаётся изменённый URI запроса целиком.

- При использовании переменных в *proxy_pass*:

```
location /name/ {
    proxy_pass http://127.0.0.1$request_uri;
}
```

В этом случае если в директиве указан URI, он передаётся на сервер как есть, заменяя URI первоначального запроса.

Проксирование WebSocket требует особой настройки.

proxy_pass_header

Синтаксис proxy_pass_header *поле* …;

Умолчание —

Контекст http, server, location

Разрешает передавать от проксируемого сервера клиенту запрещённые для передачи поля заголовка.

proxy_pass_request_body

Синтаксис proxy_pass_request_body on | off;

Умолчание proxy_pass_request_body on;

Контекст http, server, location

Позволяет запретить передачу исходного тела запроса на проксируемый сервер.

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

    proxy_pass ...;
}
```

См. также директивы proxy_set_header и proxy_pass_request_headers.

proxy_pass_request_headers

Синтаксис proxy_pass_request_headers on | off;

Умолчание proxy_pass_request_headers on;

Контекст http, server, location

Позволяет запретить передачу полей заголовка исходного запроса на проксируемый сервер.

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

    proxy_pass ...;
}
```

См. также директивы proxy_set_header и proxy_pass_request_body.

proxy_read_timeout

Синтаксис proxy_read_timeout *время*;

Умолчание proxy_read_timeout 60s;

Контекст http, server, location

Задаёт таймаут при чтении ответа проксированного сервера. Таймаут устанавливается не на всю передачу ответа, а только между двумя операциями чтения. Если по истечении этого времени проксируемый сервер ничего не передаст, соединение закрывается.

proxy_redirect

Синтаксис	proxy_redirect	default;
	proxy_redirect	off;
	proxy_redirect *перенаправление замена*;	
Умолчание	proxy_redirect default;	
Контекст	http, server, location	

Задаёт текст, который нужно изменить в полях заголовка “Location” и “Refresh” в ответе проксируемого сервера.

Предположим, проксируемый сервер вернул поле заголовка:

```
Location: http://localhost:8000/two/some/uri/
```

Директива

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

перепишет эту строку в виде:

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

В заменяемой строке можно не указывать имя сервера:

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

тогда будут подставлены основное имя сервера и порт, если он отличен от 80.

Стандартная замена, задаваемая параметром default, использует параметры директив location и proxy_pass. Поэтому две нижеприведённые конфигурации одинаковы:

```
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/;
```

Внимание

Параметр default недопустим, если в proxy_pass используются переменные.

В строке *замена* можно использовать переменные:

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

В строке *перенаправление* тоже можно использовать переменные:

```
proxy_redirect http://$proxy_host:8000/ /;
```

Директиву также можно задать при помощи регулярных выражений. При этом *перенаправление* должно начинаться либо с символа “~”, если при сравнении следует

учитывать регистр символов, либо с символов “~*”, если регистр символов учитывать не нужно. Регулярное выражение может содержать именованные и позиционные выделения, а *замена* ссылаться на них:

```
proxy_redirect ~^(http://[^:]+):\d+(/.+)$ $1$2;
proxy_redirect ~*/user/([^/]+)/(.+)$      http://$1.example.com/$2;
```

На одном уровне может быть указано несколько директив *proxy_redirect*:

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

Если к полям заголовка в ответе проксируемого сервера могут быть применены несколько директив, будет выбрана первая из них.

Параметр off отменяет действие унаследованных с предыдущего уровня конфигурации директив *proxy_redirect*.

С помощью этой директивы можно также добавлять имя хоста к относительным перенаправлениям, выдаваемым проксируемым сервером:

```
proxy_redirect / /;
```

proxy_request_buffering

Синтаксис proxy_request_buffering on | off;

Умолчание proxy_request_buffering on;

Контекст http, server, location

Разрешает или запрещает использовать буферизацию тела запроса клиента.

on	тело запроса полностью читается от клиента перед отправкой запроса на проксируемый сервер.
off	тело запроса отправляется на проксируемый сервер сразу же по мере его поступления. В этом случае запрос не может быть передан следующему серверу, если Angie уже начал отправку тела запроса.

Если для отправки тела исходного запроса используется HTTP/1.1 и передача данных частями (chunked transfer encoding), то тело запроса буферизуется независимо от значения директивы, если для проксирования также не включён HTTP/1.1.

proxy_send_lowat

Синтаксис proxy_send_lowat *размер*;

Умолчание proxy_send_lowat 0;

Контекст http, server, location

При установке директивы в ненулевое значение Angie будет пытаться минимизировать число операций отправки на исходящих соединениях с проксируемым сервером либо при

помощи флага *NOTE_LOWAT* метода kqueue, либо при помощи параметра сокета *SO_SNDLOWAT*, с указанным размером.

Примечание

Эта директива игнорируется на Linux, Solaris и Windows.

proxy_send_timeout

Синтаксис proxy_send_timeout *время*;

Умолчание proxy_send_timeout 60s;

Контекст http, server, location

Задаёт таймаут при передаче запроса проксируемому серверу. Таймаут устанавливается не на всю передачу запроса, а только между двумя операциями записи. Если по истечении этого времени проксируемый сервер не примет новых данных, соединение закрывается.

proxy_set_body

Синтаксис proxy_set_body *значение*;

Умолчание —

Контекст http, server, location

Позволяет переопределить тело запроса, передаваемое на проксируемый сервер. В качестве значения можно использовать текст, переменные и их комбинации.

proxy_set_header

Синтаксис proxy_set_header *поле значение*;

Умолчание proxy_set_header Host $proxy_host;

Контекст http, server, location

Позволяет переопределять или добавлять поля заголовка запроса, передаваемые проксируемому серверу. В качестве *значения* можно использовать текст, переменные и их комбинации. Директивы наследуются с предыдущего уровня конфигурации при условии, что на данном уровне не описаны свои директивы *proxy_set_header*. По умолчанию переопределяются только два поля:

```
proxy_set_header Host       $proxy_host;
proxy_set_header Connection close;
```

Если включено кэширование, поля заголовка “If-Modified-Since”, “If-Unmodified-Since”, “If-None-Match”, “If-Match”, “Range” и “If-Range” исходного запроса не передаются на проксируемый сервер.

Неизменённое поле заголовка запроса “Host” можно передать так:

```
proxy_set_header Host       $http_host;
```

Однако, если это поле отсутствует в заголовке запроса клиента, то ничего передаваться не будет. В этом случае лучше воспользоваться переменной $host - её значение равно имени сервера в поле “Host” заголовка запроса, или же основному имени сервера, если поля нет:

```
proxy_set_header Host       $host;
```

Кроме того, можно передать имя сервера вместе с портом проксируемого сервера:

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

Если значение поля заголовка — пустая строка, то поле вообще не будет передаваться проксируемому серверу:

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

proxy_socket_keepalive

Синтаксис proxy_socket_keepalive on | off;

Умолчание proxy_socket_keepalive off;

Контекст http, server, location

Конфигурирует поведение “TCP keepalive” для исходящих соединений к проксируемому серверу.

""	По умолчанию для сокета действуют настройки операционной системы.
on	для сокета включается параметр *SO_KEEPALIVE*

proxy_ssl_certificate

Синтаксис proxy_ssl_certificate *файл*;

Умолчание —

Контекст http, server, location

Задаёт файл с сертификатом в формате PEM для аутентификации на проксируемом HTTPS-сервере. В имени файла можно использовать переменные.

proxy_ssl_certificate_key

Синтаксис proxy_ssl_certificate_key *файл*;

Умолчание —

Контекст http, server, location

Задаёт файл с секретным ключом в формате PEM для аутентификации на проксируемом HTTPS-сервере.

Вместо файла можно указать значение “engine:*имя*:id“, которое загружает ключ с указанным *id* из OpenSSL engine с заданным именем.

В имени файла можно использовать переменные.

proxy_ssl_ciphers

Синтаксис proxy_ssl_ciphers *шифры*;

Умолчание proxy_ssl_ciphers DEFAULT;

Контекст http, server, location

Описывает разрешённые шифры для запросов к проксируемому HTTPS-серверу. Шифры задаются в формате, поддерживаемом библиотекой OpenSSL.

Полный список можно посмотреть с помощью команды “*openssl ciphers*”.

proxy_ssl_conf_command

Синтаксис proxy_ssl_conf_command *имя значение*;

Умолчание —

Контекст http, server, location

Задаёт произвольные конфигурационные команды OpenSSL при установлении соединения с проксируемым HTTPS-сервером.

Примечание

Директива поддерживается при использовании OpenSSL 1.0.2 и выше.

На одном уровне может быть указано несколько директив *proxy_ssl_conf_command*. Директивы наследуются с предыдущего уровня конфигурации при условии, что на данном уровне не описаны свои директивы *proxy_ssl_conf_command*.

Внимание

Следует учитывать, что изменение настроек OpenSSL напрямую может привести к неожиданному поведению.

proxy_ssl_crl

Синтаксис proxy_ssl_crl *файл*;

Умолчание —

Контекст http, server, location

Указывает файл с отозванными сертификатами (CRL) в формате PEM, используемыми при проверке сертификата проксируемого HTTPS-сервера.

proxy_ssl_name

Синтаксис proxy_ssl_name *имя*;

Умолчание proxy_ssl_name $proxy_host;

Контекст http, server, location

Позволяет переопределить имя сервера, используемое при проверке сертификата проксируемого HTTPS-сервера, а также для передачи его через SNI при установлении соединения с проксируемым HTTPS-сервером.

По умолчанию используется имя хоста из URL'а, заданного директивой proxy_pass.

proxy_ssl_password_file

Синтаксис proxy_ssl_password_file *файл*;

Умолчание —

Контекст http, server, location

Задаёт файл с паролями от секретных ключей, где каждый пароль указан на отдельной строке. Пароли применяются по очереди в момент загрузки ключа.

proxy_ssl_protocols

Синтаксис proxy_ssl_protocols [SSLv2] [SSLv3] [TLSv1] [TLSv1.1] [TLSv1.2] [TLSv1.3];

Умолчание proxy_ssl_protocols TLSv1 TLSv1.1 TLSv1.2;

Контекст http, server, location

Разрешает указанные протоколы для запросов к проксируемому HTTPS-серверу.

proxy_ssl_server_name

Синтаксис proxy_ssl_server_name on | off;

Умолчание proxy_ssl_server_name off;

Контекст http, server, location

Разрешает или запрещает передачу имени сервера через расширение Server Name Indication протокола TLS (SNI, RFC 6066) при установлении соединения с проксируемым HTTPS-сервером.

proxy_ssl_session_reuse

Синтаксис proxy_ssl_session_reuse on | off;

Умолчание proxy_ssl_session_reuse on;

Контекст http, server, location

Определяет, использовать ли повторно SSL-сессии при работе с проксируемым сервером. Если в логах появляются ошибки "*SSL3_GET_FINISHED:digest check failed*", то можно попробовать выключить повторное использование сессий.

proxy_ssl_trusted_certificate

Синтаксис proxy_ssl_trusted_certificate *файл*;

Умолчание —

Контекст http, server, location

Задаёт файл с доверенными сертификатами CA в формате PEM, используемыми при проверке сертификата проксируемого HTTPS-сервера.

proxy_ssl_verify

Синтаксис proxy_ssl_verify on | off;

Умолчание proxy_ssl_verify off;

Контекст http, server, location

Разрешает или запрещает проверку сертификата проксируемого HTTPS-сервера.

proxy_ssl_verify_depth

Синтаксис proxy_ssl_verify_depth *число*;

Умолчание proxy_ssl_verify_depth 1;

Контекст http, server, location

Устанавливает глубину проверки в цепочке сертификатов проксируемого HTTPS-сервера.

proxy_store

Синтаксис proxy_store on | off | *строка*;

Умолчание proxy_store off;

Контекст http, server, location

Разрешает сохранение на диск файлов.

on	сохраняет файлы в соответствии с путями, указанными в директивах alias или root
off	запрещает сохранение файлов

Имя файла можно задать явно с помощью строки с переменными:

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

Время изменения файлов выставляется согласно полученному полю “Last-Modified” в заголовке ответа. Ответ сначала записывается во временный файл, а потом этот файл переименовывается. Временный файл и постоянное место хранения ответа могут располагаться на разных файловых системах. Однако нужно учитывать, что в этом случае вместо дешёвой операции переименовывания в пределах одной файловой системы файл копируется с одной файловой системы на другую. Поэтому лучше, если сохраняемые файлы будут находиться на той же файловой системе, что и каталог с временными файлами, задаваемый директивой proxy_temp_path для данного *location*.

Директиву можно использовать для создания локальных копий статических неизменяемых файлов, например, так:

```
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/;
}
```

или так:

```
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

Синтаксис proxy_store_access *пользователи*:*права* …;

Умолчание proxy_store_access user:rw;

Контекст http, server, location

Задаёт права доступа для создаваемых файлов и каталогов, например,

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

Если заданы какие-либо права для *group* или *all*, то права для user указывать необязательно:

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

proxy_temp_file_write_size

Синтаксис proxy_temp_file_write_size *размер*;

Умолчание proxy_temp_file_write_size 8k|16k;

Контекст http, server, location

Ограничивает размер данных, сбрасываемых во временный файл за один раз, при включённой буферизации ответов проксируемого сервера во временные файлы. По

умолчанию размер ограничен двумя буферами, заданными директивами proxy_buffer_size и proxy_buffers. Максимальный размер временного файла задаётся директивой proxy_max_temp_file_size.

proxy_temp_path

Синтаксис proxy_temp_path *путь* [*уровень1* [*уровень2* [*уровень3*]]]`;

Умолчание proxy_temp_path proxy_temp;

Контекст http, server, location

Задаёт имя каталога для хранения временных файлов с данными, полученными от проксируемых серверов. В каталоге может использоваться иерархия подкаталогов до трёх уровней. Например, при такой конфигурации

```
proxy_temp_path /spool/angie/proxy_temp 1 2;
```

временный файл будет следующего вида:

```
/spool/angie/proxy_temp/7/45/00000123457
```

См. также параметр *use_temp_path* директивы proxy_cache_path.

Встроенные переменные

В модуле *http_proxy* есть встроенные переменные, которые можно использовать для формирования заголовков с помощью директивы proxy_set_header:

$proxy_host

имя и порт проксируемого сервера, как указано в директиве proxy_pass;

$proxy_port

порт проксируемого сервера, как указано в директиве proxy_pass, или стандартный порт протокола;

$proxy_add_x_forwarded_for

поле заголовка запроса клиента “X-Forwarded-For” и добавленная к нему через запятую переменная $remote_addr. Если же поля “X-Forwarded-For” в заголовке запроса клиента нет, то переменная $proxy_add_x_forwarded_for равна переменной $remote_addr.

Модуль http_random_index

Обслуживает запросы, оканчивающиеся слэшом (‘/’), и выдаёт случайный файл в качестве индексного файла каталога. Модуль выполняется до модуля http_index.

По умолчанию этот модуль не собирается, его сборку необходимо разрешить с помощью конфигурационного параметра --with-http_random_index_module.

Пример конфигурации

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

```
}
```

Директивы

random_index

Синтаксис random_index on | off.;

Умолчание random_index off;

Контекст location

Разрешает или запрещает в содержащем *location* обработку этим модулем.

Модуль http_realip

Позволяет менять адрес и необязательный порт клиента на переданные в указанном поле заголовка.

По умолчанию этот модуль не собирается, его сборку необходимо разрешить с помощью конфигурационного параметра --with-http_realip_module.

Пример конфигурации

```
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;
```

Директивы

set_real_ip_from

Синтаксис set_real_ip_from *адрес* | *CIDR* | unix:;

Умолчание —

Контекст http, server, location

Задаёт доверенные адреса, которые передают верный адрес для замены. Если указано специальное значение “*unix:*“, доверенными будут считаться все UNIX-сокеты. Доверенные адреса могут быть также заданы при помощи имени хоста.

real_ip_header

Синтаксис real_ip_header *поле* | X-Real-IP | X-Forwarded-For | proxy_protocol;

Умолчание real_ip_header X-Real-IP;

Контекст http, server, location

Задаёт поле заголовка запроса, значение которого будет использоваться для замены адреса клиента.

Значение поля заголовка запроса, содержащее необязательный порт, также используется для замены порта клиента. Адрес и порт должны быть указаны согласно RFC 3986.

Параметр proxy_protocol меняет адрес клиента на указанный в заголовке PROXY-протокола. Протокол PROXY должен быть предварительно включён при помощи установки параметра *proxy_protocol* в директиве listen.

real_ip_recursive

Синтаксис real_ip_recursive on | off;

Умолчание real_ip_recursive off;

Контекст http, server, location

При выключенном рекурсивном поиске исходный адрес клиента, совпадающий с одним из доверенных адресов, заменяется на последний адрес, переданный в поле заголовка запроса, заданного директивой real_ip_header. При включённом рекурсивном поиске исходный адрес клиента, совпадающий с одним из доверенных адресов, заменяется на последний не доверенный адрес, переданный в заданном поле заголовка запроса.

Встроенные переменные

$realip_remote_addr

хранит исходный адрес клиента

$realip_remote_port

хранит исходный порт клиента

Модуль http_referer

Позволяет блокировать доступ к сайту для запросов с неверными значениями поля “Referer” в заголовке. Следует иметь в виду, что подделать запрос с нужным значением поля “Referer” не составляет большого труда, поэтому цель использования данного модуля заключается не в стопроцентном блокировании подобных запросов, а в блокировании массового потока запросов, сделанных обычными браузерами. Нужно также учитывать, что обычные браузеры могут не передавать поле “Referer” даже для верных запросов.

Пример конфигурации

```
valid_referers none blocked server_names
               *.example.com example.* www.example.org/galleries/
               ~\.google\.;

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

Директивы

referer_hash_bucket_size

Синтаксис referer_hash_bucket_size *размер*;

Умолчание referer_hash_bucket_size 64;

Контекст server, location

Задаёт размер корзины хэш-таблиц со значениями “Referer”. Подробнее настройка хэш-таблиц обсуждается отдельно.

referer_hash_max_size

Синтаксис referer_hash_max_size *размер*;

Умолчание referer_hash_max_size 2048;

Контекст server, location

Задаёт максимальный размер хэш-таблиц со значениями “Referer”. Подробнее настройка хэш-таблиц обсуждается отдельно.

valid_referers

Синтаксис valid_referers none | blocked | server_names | *строка* …;

Умолчание —

Контекст server, location

Задаёт значения поля “Referer” заголовка запроса, при которых встроенная переменная $invalid_referer будет иметь пустую строку в качестве значения. В противном случае значение переменной равно “1”. Поиск совпадения производится без учёта регистра символов.

Параметры могут быть следующие:

none	поле “Referer” в заголовке запроса отсутствует;
blocked	поле “Referer” в заголовке запроса присутствует, но его значение удалено межсетевым экраном (firewall) или прокси-сервером; к таким значениям относятся строки, не начинающиеся на “*http://*” или “*https://*”;
server_names	в поле “Referer” заголовка запроса указано одно из имён сервера;
произвольная строка	задаёт имя сервера и необязательное начало URI. В начале или конце имени сервера может быть “*”. При проверке порт сервера в поле “Referer” игнорируется;
регулярное выражение	в начале должен быть символ “~”. Необходимо учитывать, что на совпадение с выражением будет проверяться текст, начинающийся после “*http://*” или “*https://*”.

Пример:

```
valid_referers none blocked server_names
```

```
        *.example.com example.* www.example.org/galleries/
        ~\.google\.;
```

Встроенные переменные

$invalid_referer

Пустая строка, если значение поля “Referer” заголовка запроса считается правильным, иначе “1”.

Модуль http_rewrite

Позволяет изменять URI запроса с помощью регулярных выражений PCRE, делать перенаправления и выбирать конфигурацию по условию.

Директивы break, if, return, rewrite и set обрабатываются в следующем порядке:

- последовательно выполняются директивы этого модуля, описанные на уровне *server*;
- в цикле:
  - ищется *location* по URI запроса;
  - последовательно выполняются директивы этого модуля, описанные в найденном *location*;
  - цикл повторяется, если URI запроса изменялся, но не более 10 раз.

Директивы

break

Синтаксис break;

Умолчание —

Контекст server, location, if

Завершает обработку текущего набора директив модуля *http_rewrite*.

Если директива указана внутри *location*, дальнейшая обработка запроса продолжается в этом *location*.

Пример:

```
if ($slow) {
    limit_rate 10k;
    break;
}
```

if

Синтаксис if (*условие*) { … }

Умолчание —

Контекст server, location

Проверяется указанное условие. Если оно истинно, то выполняются указанные в фигурных скобках директивы этого модуля и запросу назначается конфигурация, указанная внутри директивы *if*. Конфигурации внутри директив *if* наследуются с предыдущего уровня конфигурации.

В качестве условия могут быть заданы:

- имя переменной; ложными значениями переменной являются пустая строка или “0”;
- сравнение переменной со строкой с помощью операторов “=” и “!=”;
- соответствие переменной регулярному выражению с учётом регистра символов — “~” и без него — “~*”. В регулярных выражениях можно использовать выделения, которые затем доступны в виде переменных $1..$9. Также можно использовать отрицательные операторы “!~” и “!~*”. Если в регулярном выражении встречаются символы “}” или “;”, то всё выражение следует заключить в одинарные или двойные кавычки.
- проверка существования файла с помощью операторов “-f” и “!-f”;
- проверка существования каталога с помощью операторов “-d” и “!-d”;
- проверка существования файла, каталога или символической ссылки с помощью операторов “-e” и “!-e”;
- проверка исполняемости файла с помощью операторов “-x” и “!-x”.

Примеры:

```
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;
}
```

Примечание

Значение встроенной переменной $invalid_referer задаётся директивой valid_referers.

return

Синтаксис	return *код* [*текст*]; return *код* *URL*; return *URL*;
Умолчание	—
Контекст	server, location, if

Завершает обработку и возвращает клиенту указанный код. Нестандартный код 444 закрывает соединение без передачи заголовка ответа.

Можно задать либо URL перенаправления (для кодов 301, 302, 303, 307 и 308) либо текст тела ответа (для остальных кодов). В тексте тела ответа и URL перенаправления можно использовать переменные. Как частный случай, URL перенаправления может быть задан как URI, локальный для данного сервера, при этом полный URL перенаправления формируется согласно схеме запроса (*$scheme*) и директивам server_name_in_redirect и port_in_redirect.

Кроме того, в качестве единственного параметра можно указать URL для временного перенаправления с кодом 302. Такой параметр должен начинаться со строк “*http://*”, “*https://*” или “*$scheme*”. В URL можно использовать переменные.

См. также директиву error_page.

rewrite

Синтаксис	rewrite *regex замена* [*флаг*];
Умолчание	—
Контекст	server, location, if

Если указанное регулярное выражение соответствует URI запроса, URI изменяется в соответствии со строкой замены. Директивы *rewrite* выполняются последовательно, в порядке их следования в конфигурационном файле. С помощью флагов можно прекратить дальнейшую обработку директив. Если строка замены начинается с “*http://*”, “*https://*” или “$scheme”, то обработка завершается и клиенту возвращается перенаправление.

Необязательный параметр флаг может быть одним из:

last	завершает обработку текущего набора директив модуля *http_rewrite*, после чего ищется новый *location*, соответствующий изменённому URI;
break	завершает обработку текущего набора директив модуля *http_rewrite* аналогично директиве *break*;
redirect	возвращает временное перенаправление с кодом 302; используется, если строка замены не начинается с “*http://*”, “*https://*” или “*$scheme*”;
permanent	возвращает постоянное перенаправление с кодом 301.

Полный URL перенаправлений формируется согласно схеме запроса (*$scheme*) и директив server_name_in_redirect и port_in_redirect.

Пример:

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

Если же эти директивы поместить в location “/download/”, то нужно заменить флаг last на break, иначе Angie сделает 10 циклов и вернёт ошибку 500:

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

Если в строке замены указаны новые аргументы запроса, то предыдущие аргументы запроса добавляются после них. Если такое поведение нежелательно, можно отказаться от этого добавления, указав в конце строки замены знак вопроса, например:

```
rewrite ^/users/(.*)$ /show?user=$1? last;
```

Если в регулярном выражении встречаются символы “}” или “;”, то всё выражение следует заключить в одинарные или двойные кавычки.

rewrite_log

Синтаксис rewrite_log on | off;

Умолчание rewrite_log off;

Контекст http, server, location, if

Разрешает или запрещает записывать в error_log на уровне *notice* результаты обработки директив модуля *http_rewrite*.

set

Синтаксис set *$переменная значение*;

Умолчание —

Контекст server, location, if

Устанавливает значение указанной переменной. В качестве значения можно использовать текст, переменные и их комбинации.

uninitialized_variable_warn

Синтаксис uninitialized_variable_warn on | off;

Умолчание rewrite_log on;

Контекст http, server, location, if

Определяет, нужно ли писать в лог предупреждения о неинициализированных переменных.

Внутреннее устройство

Директивы модуля *http_rewrite* компилируются на стадии конфигурации во внутренние инструкции, интерпретируемые во время обработки запроса. Интерпретатор представляет из себя простую стековую виртуальную машину.

Например, директивы

```
location /download/ {
    if ($forbidden) {
        return 403;
    }

    if ($slow) {
        limit_rate 10k;
    }

    rewrite ^/(download/.*)/media/(.*)\..*$ /$1/mp3/$2.mp3 break;
}
```

будут транслированы в такие инструкции:

```
переменная $forbidden
проверка на ноль
    возврат 403
    завершение всего кода
переменная $slow
проверка на ноль
проверка регулярного выражения
копирование "/"
копирование $1
копирование "/mp3/"
копирование $2
копирование ".mp3"
завершение регулярного выражения
завершение всего кода
```

Обратите внимание, что инструкций для директивы limit_rate нет, поскольку она не имеет отношения к модулю *http_rewrite*. Для блока *if* создаётся отдельная конфигурация. Если условие истинно, запрос получает эту конфигурацию, и в ней *limit_rate* равен 10k.

Директиву

```
rewrite ^/(download/.*)/media/(.*)\..*$ /$1/mp3/$2.mp3 break;
```

можно сделать на одну инструкцию меньше, если в регулярном выражении перенести первый слэш внутрь скобок:

```
rewrite ^(/download/.*)/media/(.*)\..*$ $1/mp3/$2.mp3 break;
```

Тогда соответствующие инструкции будут выглядеть так:

```
проверка регулярного выражения
копирование $1
копирование "/mp3/"
копирование $2
копирование ".mp3"
завершение регулярного выражения
завершение всего кода
```

Модуль http_scgi

Позволяет передавать запросы SCGI-серверу.

Пример конфигурации

```
location / {
    include   scgi_params;
    scgi_pass localhost:9000;
}
```

Директивы

scgi_bind

Синтаксис scgi_bind *адрес* [transparent] | off;

Умолчание —

Контекст http, server, location

Задаёт локальный IP-адрес с необязательным портом, который будет использоваться в исходящих соединениях с SCGI-сервером. В значении параметра допустимо использование переменных. Специальное значение off отменяет действие унаследованной с предыдущего уровня конфигурации директивы *scgi_bind*, позволяя системе самостоятельно выбирать локальный IP-адрес и порт.

Параметр transparent позволяет задать нелокальный IP-адрес, который будет использоваться в исходящих соединениях с SCGI-сервером, например, реальный IP-адрес клиента:

```
scgi_bind $remote_addr transparent;
```

Для работы параметра обычно требуется запустить рабочие процессы Angie с привилегиями суперпользователя. В Linux это не требуется, так как если указан параметр *transparent*, то рабочие процессы наследуют *capability CAP_NET_RAW* из главного процесса.

Внимание

Необходимо настроить таблицу маршрутизации ядра для перехвата сетевого трафика с SCGI-сервера.

scgi_buffer_size

Синтаксис scgi_buffer_size *размер*;

Умолчание scgi_buffer_size 4k|8k;

Контекст http, server, location

Задаёт размер буфера, в который будет читаться первая часть ответа, получаемого от SCGI-сервера. В этой части ответа находится, как правило, небольшой заголовок ответа. По умолчанию размер одного буфера равен размеру страницы памяти. В зависимости от платформы это или 4K, или 8K, однако его можно сделать меньше.

scgi_buffering

Синтаксис scgi_buffering *размер*;

Умолчание scgi_buffering on;

Контекст http, server, location

on	Angie принимает ответ SCGI-сервера как можно быстрее, сохраняя его в буферы, заданные директивами scgi_buffer_size и scgi_buffers. Если ответ не вмещается целиком в память, то его часть может быть записана на диск во временный файл. Запись во временные файлы контролируется директивами scgi_max_temp_file_size и scgi_temp_file_write_size.
off	Ответ синхронно передаётся клиенту сразу же по мере его поступления. Angie не пытается считать весь ответ SCGI-сервера. Максимальный размер данных, который Angie может принять от сервера за один раз, задаётся директивой scgi_buffer_size.

Буферизация может быть также включена или выключена путём передачи значения “yes” или “no” в поле “X-Accel-Buffering” заголовка ответа. Эту возможность можно запретить с помощью директивы scgi_ignore_headers.

scgi_buffers

Синтаксис scgi_buffers *число размер*;

Умолчание scgi_buffers 8 4k|8k;

Контекст http, server, location

Задаёт число и размер буферов для одного соединения, в которые будет читаться ответ, получаемый от SCGI-сервера.
По умолчанию размер одного буфера равен размеру страницы. В зависимости от платформы это или 4K, или 8K.

scgi_busy_buffers_size

Синтаксис scgi_busy_buffers_size *размер*;

Умолчание scgi_buffers 8k|16k;

Контекст http, server, location

При включённой буферизации ответов SCGI-сервера, ограничивает суммарный размер буферов, которые могут быть заняты для отправки ответа клиенту, пока ответ ещё не прочитан целиком. Оставшиеся буферы тем временем могут использоваться для чтения ответа и, при необходимости, буферизации части ответа во временный файл.

По умолчанию размер ограничен величиной двух буферов, заданных директивами scgi_buffer_size и scgi_buffers.

scgi_cache

Синтаксис scgi_cache *зона* | off;

Умолчание scgi_cache off;

Контекст http, server, location

Задаёт зону разделяемой памяти, используемой для кэширования. Одна и та же зона может использоваться в нескольких местах. В значении параметра можно использовать переменные.

off	запрещает кэширование, унаследованное с предыдущего уровня конфигурации.

scgi_cache_background_update

Синтаксис scgi_cache_background_update on | off;

Умолчание scgi_cache_background_update off;

Контекст http, server, location

Позволяет запустить фоновый подзапрос для обновления просроченного элемента кэша, в то время как клиенту возвращается устаревший закэшированный ответ.

Внимание

Использование устаревшего закэшированного ответа в момент его обновления должно быть разрешено.

scgi_cache_bypass

Синтаксис scgi_cache_bypass …;

Умолчание —

Контекст http, server, location

Задаёт условия, при которых ответ не будет браться из кэша. Если значение хотя бы одного из строковых параметров непустое и не равно “0”, то ответ не берётся из кэша:

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

Можно использовать совместно с директивой scgi_no_cache.

scgi_cache_key

Синтаксис scgi_cache_key *строка*;

Умолчание —

Контекст http, server, location

Задаёт ключ для кэширования, например,

```
scgi_cache_key localhost:9000$request_uri;
```

scgi_cache_lock

Синтаксис scgi_cache_lock on | off;

Умолчание scgi_cache_lock off;

Контекст http, server, location

Если включено, одновременно только одному запросу будет позволено заполнить новый элемент кэша, идентифицируемый согласно директиве scgi_cache_key, передав запрос на SCGI-сервер. Остальные запросы этого же элемента будут либо ожидать появления ответа в кэше, либо освобождения блокировки этого элемента, в течение времени, заданного директивой scgi_cache_lock_timeout.

scgi_cache_lock_age

Синтаксис scgi_cache_lock_age *время*;

Умолчание scgi_cache_lock_age 5s;

Контекст http, server, location

Если последний запрос, переданный на SCGI-сервер для заполнения нового элемента кэша, не завершился за указанное время, на SCGI-сервер может быть передан ещё один запрос.

scgi_cache_lock_timeout

Синтаксис scgi_cache_lock_timeout *время*;

Умолчание scgi_cache_lock_timeout 5s;

Контекст http, server, location

Задаёт таймаут для scgi_cache_lock. По истечении указанного времени запрос будет передан на SCGI-сервер, однако ответ не будет закэширован.

scgi_cache_max_range_offset

Синтаксис scgi_cache_max_range_offset *число*;

Умолчание —

Контекст http, server, location

Задаёт смещение в байтах для запросов с указанием диапазона запрашиваемых байт (byte-range requests). Если диапазон находится за указанным смещением, range-запрос будет передан на SCGI-сервер и ответ не будет закэширован.

scgi_cache_methods

Синтаксис scgi_cache_methods GET | HEAD | POST …;

Умолчание scgi_cache_methods GET HEAD;

Контекст http, server, location

Если метод запроса клиента указан в этой директиве, то ответ будет закэширован. Методы “GET” и “HEAD” всегда добавляются в список, но тем не менее рекомендуется перечислять их явно. См. также директиву scgi_no_cache.

scgi_cache_min_uses

Синтаксис scgi_cache_min_uses *число*;

Умолчание scgi_cache_min_uses 1;

Контекст http, server, location

Задаёт число запросов, после которого ответ будет закэширован.

scgi_cache_path

Синтаксис scgi_cache_path *путь* [levels = *уровни*] [use_temp_path=on|off] keys_zone = *имя:размер* [inactive = *время*] [max_size = *размер*] [min_free = *размер*] [manager_files = *число*] [manager_sleep = *время*] [manager_threshold = *время*] [loader_files = *число*] [loader_sleep = *время*] [loader_threshold = *время*];

Умолчание —

Контекст http

Задаёт путь и другие параметры кэша. Данные кэша хранятся в файлах. Именем файла в кэше является результат функции MD5 от ключа кэширования.

Параметр levels задаёт уровни иерархии кэша: можно задать от 1 до 3 уровней, на каждом уровне допускаются значения 1 или 2.

Например, при использовании

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

имена файлов в кэше будут такого вида:

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

Кэшируемый ответ сначала записывается во временный файл, а потом этот файл переименовывается. Временные файлы и кэш могут располагаться на разных файловых системах. Однако нужно учитывать, что в этом случае вместо дешёвой операции переименовывания в пределах одной файловой системы файл копируется с одной файловой системы на другую. Поэтому лучше, если кэш будет находиться на той же файловой системе, что и каталог с временными файлами.

Какой из каталогов будет использоваться для временных файлов, определяется параметром use_temp_path.

on	Если параметр не задан или установлен в значение “on”, будет использоваться каталог, задаваемый директивой scgi_temp_path для данного *location*
off	временные файлы будут располагаться непосредственно в каталоге кэша

Кроме того, все активные ключи и информация о данных хранятся в зоне разделяемой памяти, имя и размер которой задаются параметром keys_zone. Зоны размером в 1 мегабайт достаточно для хранения около 8 тысяч ключей.

Если к данным кэша не обращаются в течение времени, заданного параметром inactive, то данные удаляются, независимо от их свежести. По умолчанию *inactive* равен 10 минутам.

Специальный процесс **cache manager** следит за максимальным размером кэша, а также за минимальным объёмом свободного места на файловой системе с кэшем, и удаляет наименее востребованные данные при превышении максимального размера кэша или недостаточном объёме свободного места. Удаление данных происходит итерациями.

max_size	максимальное пороговое значение размера кэша
min_free	минимальное пороговое значение объёма свободного места на файловой системе с кэшем
manager_files	максимальное количество удаляемых элементов кэша за одну итерацию По умолчанию *100*
manager_threshold	ограничивает время работы одной итерации По умолчанию *200* миллисекунд
manager_sleep	время, в течение которого выдерживается пауза между итерациями По умолчанию *50* миллисекунд

Через минуту после старта Angie активируется специальный процесс cache **loader**, который загружает в зону кэша информацию о ранее закэшированных данных, хранящихся на файловой системе. Загрузка также происходит итерациями.

loader_files	максимальное количество элементов кэша к загрузке в одну итерацию По умолчанию *100*
loader_threshold	ограничивает время работы одной итерации По умолчанию *200* миллисекунд
loader_sleep	время, в течение которого выдерживается пауза между итерациями По умолчанию *50* миллисекунд

scgi_cache_revalidate

Синтаксис scgi_cache_revalidate on | off;

Умолчание scgi_cache_revalidate off;

Контекст http, server, location

Разрешает ревалидацию просроченных элементов кэша при помощи условных запросов с полями заголовка “If-Modified-Since” и “If-None-Match”.

scgi_cache_use_stale

Синтаксис scgi_cache_use_stale error | timeout | invalid_header | updating | http_500 | http_503 | http_403 | http_404 | http_429 | off …;

Умолчание scgi_cache_use_stale off;

Контекст http, server, location

Определяет, в каких случаях можно использовать устаревший закэшированный ответ. Параметры директивы совпадают с параметрами директивы scgi_next_upstream.

error	Позволяет использовать устаревший закэшированный ответ при невозможности выбора SCGI-сервера для обработки запроса.
updating	Дополнительный параметр, разрешает использовать устаревший закэшированный ответ, если на данный момент он уже обновляется. Это позволяет минимизировать число обращений к SCGI-серверам при обновлении закэшированных данных.

Использование устаревшего закэшированного ответа может также быть разрешено непосредственно в заголовке ответа на определённое количество секунд после того, как ответ устарел:

- Расширение stale-while-revalidate поля заголовка “Cache-Control” разрешает использовать устаревший закэшированный ответ, если на данный момент он уже обновляется.
- Расширение stale-if-error поля заголовка “Cache-Control” разрешает использовать устаревший закэшированный ответ в случае ошибки.

Примечание

Такой способ менее приоритетен, чем задание параметров директивы.

Чтобы минимизировать число обращений к SCGI-серверам при заполнении нового элемента кэша, можно воспользоваться директивой scgi_cache_lock.

scgi_cache_valid

Синтаксис scgi_cache_valid [*код* …] *время*;

Умолчание —

Контекст http, server, location

Задаёт время кэширования для разных кодов ответа. Например, директивы

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

задают время кэширования 10 минут для ответов с кодами 200 и 302 и 1 минуту для ответов с кодом 404.

Если указано только время кэширования,

```
scgi_cache_valid 5m;
```

то кэшируются только ответы 200, 301 и 302.

Кроме того, можно кэшировать любые ответы с помощью параметра any:

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

Примечание

Параметры кэширования могут также быть заданы непосредственно в заголовке ответа. Такой способ приоритетнее, чем задание времени кэширования с помощью директивы.

- Поле заголовка “X-Accel-Expires” задаёт время кэширования ответа в секундах. Значение *0* запрещает кэшировать ответ. Если значение начинается с префикса @, оно задаёт абсолютное время в секундах с начала эпохи, до которого ответ может быть закэширован.
- Если в заголовке нет поля “X-Accel-Expires”, параметры кэширования определяются по полям заголовка “Expires” или “Cache-Control”.
- Ответ, в заголовке которого есть поле “Set-Cookie”, не будет кэшироваться.
- Ответ, в заголовке которого есть поле “Vary” со специальным значением “*”, не будет кэшироваться. Ответ, в заголовке которого есть поле “Vary” с другим значением, будет закэширован с учётом соответствующих полей заголовка запроса.

Обработка одного или более из этих полей заголовка может быть отключена при помощи директивы scgi_ignore_headers.

scgi_connect_timeout

Синтаксис scgi_connect_timeout *время*;

Умолчание scgi_connect_timeout 60s;

Контекст http, server, location

Задаёт таймаут для установления соединения с SCGI-сервером. Необходимо иметь в виду, что этот таймаут обычно не может превышать 75 секунд.

scgi_force_ranges

Синтаксис scgi_force_ranges off;

Умолчание scgi_force_ranges off;

Контекст http, server, location

Включает поддержку диапазонов запрашиваемых байт (byte-range) для кэшированных и некэшированных ответов SCGI-сервера вне зависимости от наличия поля “Accept-Ranges” в заголовках этих ответов.

scgi_hide_header

Синтаксис scgi_hide_header *поле*;

Умолчание —

Контекст http, server, location

По умолчанию Angie не передаёт клиенту поля заголовка “Status” и “X-Accel-…” из ответа SCGI-сервера. Директива *scgi_hide_header* задаёт дополнительные поля, которые не будут передаваться. Если же передачу полей нужно разрешить, можно воспользоваться директивой scgi_pass_header.

scgi_ignore_client_abort

Синтаксис scgi_ignore_client_abort on | off;

Умолчание scgi_ignore_client_abort off;

Контекст http, server, location

Определяет, закрывать ли соединение с SCGI-сервером в случае, если клиент закрыл соединение, не дождавшись ответа.

scgi_ignore_headers

Синтаксис scgi_ignore_headers *поле* …;

Умолчание —

Контекст http, server, location

Запрещает обработку некоторых полей заголовка из ответа SCGI-сервера. В директиве можно указать поля “X-Accel-Redirect”, “X-Accel-Expires”, “X-Accel-Limit-Rate”, “X-Accel-Buffering”, “X-Accel-Charset”, “Expires”, “Cache-Control”, “Set-Cookie” и “Vary”.

Если не запрещено, обработка этих полей заголовка заключается в следующем:

- “X-Accel-Expires”, “Expires”, “Cache-Control”, “Set-Cookie” и “Vary” задают параметры кэширования ответа;
- “X-Accel-Redirect” производит внутреннее перенаправление на указанный URI;
- “X-Accel-Limit-Rate” задаёт ограничение скорости передачи ответа клиенту;
- “X-Accel-Buffering” включает или выключает буферизацию ответа;
- “X-Accel-Charset” задаёт желаемую кодировку ответа.

scgi_intercept_errors

Синтаксис scgi_intercept_errors on | off;

Умолчание scgi_intercept_errors off;

Контекст http, server, location

Определяет, передавать ли клиенту ответы SCGI-сервера с кодом больше либо равным 300, или же перехватывать их и перенаправлять на обработку Angie с помощью директивы error_page.

scgi_limit_rate

Синтаксис scgi_limit_rate *скорость*;

Умолчание scgi_limit_rate 0;

Контекст http, server, location

Ограничивает скорость чтения ответа от SCGI-сервера. *Скорость* задаётся в байтах в секунду.

0	отключает ограничение скорости

Примечание

Ограничение устанавливается на запрос, поэтому, если Angie одновременно откроет два соединения к SCGI-серверу, суммарная скорость будет вдвое выше заданного ограничения. Ограничение работает только в случае, если включена буферизация ответов SCGI-сервера.

scgi_max_temp_file_size

Синтаксис scgi_max_temp_file_size *размер*;

Умолчание scgi_max_temp_file_size 1024m;

Контекст http, server, location

Если включена буферизация ответов SCGI-сервера, и ответ не вмещается целиком в буферы, заданные директивами scgi_buffer_size и scgi_buffers, часть ответа может быть записана во временный файл. Эта директива задаёт максимальный размер временного файла. Размер данных, сбрасываемых во временный файл за один раз, задаётся директивой scgi_temp_file_write_size.

0	отключает возможность буферизации ответов во временные файлы

Примечание

Данное ограничение не распространяется на ответы, которые будут закэшированы или сохранены на диске.

scgi_next_upstream

Синтаксис scgi_next_upstream error | timeout | invalid_header | http_500 | http_503 | http_403 | http_404 | http_429 | non_idempotent | off …;

Умолчание scgi_next_upstream error timeout;

Контекст http, server, location

Определяет, в каких случаях запрос будет передан следующему серверу:

error	произошла ошибка соединения с сервером, передачи ему запроса или чтения заголовка ответа сервера;
timeout	произошёл таймаут во время соединения с сервером, передачи ему запроса или чтения заголовка ответа сервера;
invalid_header	сервер вернул пустой или неверный ответ;
http_500	сервер вернул ответ с кодом 500;
http_503	сервер вернул ответ с кодом 503;
http_403	сервер вернул ответ с кодом 403;
http_404	сервер вернул ответ с кодом 404;

http_429	сервер вернул ответ с кодом 429;
non_idempotent	обычно запросы с неидемпотентным методом (*POST*, *LOCK*, *PATCH*) не передаются на другой сервер, если запрос серверу группы уже был отправлен; включение параметра явно разрешает повторять подобные запросы;
off	запрещает передачу запроса следующему серверу.

Примечание

Необходимо понимать, что передача запроса следующему серверу возможна только при условии, что клиенту ещё ничего не передавалось. То есть, если ошибка или таймаут возникли в середине передачи ответа клиенту, то действие директивы на такой запрос не распространяется.

Директива также определяет, что считается неудачной попыткой работы с сервером.

error timeout invalid_header	всегда считаются неудачными попытками, даже если они не указаны в директиве
http_500 http_503 http_429	считаются неудачными попытками, только если они указаны в директиве
http_403 http_404	никогда не считаются неудачными попытками

Передача запроса следующему серверу может быть ограничена по количеству попыток и по времени.

scgi_next_upstream_timeout

Синтаксис scgi_next_upstream_timeout *время*;

Умолчание scgi_next_upstream_timeout 0;

Контекст http, server, location

Ограничивает время, в течение которого возможна передача запроса следующему серверу.

0	отключает это ограничение

scgi_next_upstream_tries

Синтаксис scgi_next_upstream_tries *число*;

Умолчание scgi_next_upstream_tries 0;

Контекст http, server, location

Ограничивает число допустимых попыток для передачи запроса следующему серверу.

0	отключает это ограничение

scgi_no_cache

Синтаксис scgi_no_cache *строка* …;

Умолчание —

Контекст http, server, location

Задаёт условия, при которых ответ не будет сохраняться в кэш. Если значение хотя бы одного из строковых параметров непустое и не равно “0”, то ответ не будет сохранён:

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

Можно использовать совместно с директивой scgi_cache_bypass.

scgi_param

Синтаксис scgi_param *параметр значение* [if_not_empty];

Умолчание —

Контекст http, server, location

Задаёт параметр, который будет передаваться SCGI-серверу. В качестве значения можно использовать текст, переменные и их комбинации. Директивы наследуются с предыдущего уровня конфигурации при условии, что на данном уровне не описаны свои директивы *SCGI-param*.

Стандартные переменные окружения CGI должны передаваться как заголовки SCGI, см. файл *scgi_params* из дистрибутива:

```
location / {
    include scgi_params;
#    ...
}
```

Если директива указана с if_not_empty, то такой параметр с пустым значением передаваться на сервер не будет:

```
scgi_param HTTPS $https if_not_empty;
```

scgi_pass

Синтаксис scgi_pass URL;

Умолчание —

Контекст location, if в location

Задаёт адрес SCGI-сервера. Адрес может быть указан в виде доменного имени или IP-адреса, и порта:

```
scgi_pass localhost:9000;
```

или в виде пути UNIX-сокета:

```
scgi_pass unix:/tmp/scgi.socket;
```

Если доменному имени соответствует несколько адресов, то все они будут использоваться по очереди (round-robin). Кроме того, в качестве адреса можно указать группу серверов.

В значении параметра можно использовать переменные. В этом случае, если адрес указан в виде доменного имени, имя ищется среди описанных групп серверов и если не найдено, то определяется с помощью resolver'а.

scgi_pass_header

Синтаксис scgi_pass_header *поле* …;

Умолчание —

Контекст http, server, location

Разрешает передавать от SCGI-сервера клиенту запрещённые для передачи поля заголовка.

scgi_pass_request_body

Синтаксис scgi_pass_request_body on | off;

Умолчание scgi_pass_request_body on;

Контекст http, server, location

Позволяет запретить передачу исходного тела запроса на SCGI-сервер. См. также директиву scgi_pass_request_headers.

scgi_pass_request_headers

Синтаксис scgi_pass_request_headers on | off;

Умолчание scgi_pass_request_headers on;

Контекст http, server, location

Позволяет запретить передачу полей заголовка исходного запроса на SCGI-сервер. См. также директиву scgi_pass_request_body.

scgi_read_timeout

Синтаксис scgi_read_timeout *время*;

Умолчание scgi_read_timeout 60s;

Контекст http, server, location

Задаёт таймаут при чтении ответа SCGI-сервера. Таймаут устанавливается не на всю передачу ответа, а только между двумя операциями чтения. Если по истечении этого времени SCGI-сервер ничего не передаст, соединение закрывается.

scgi_request_buffering

Синтаксис scgi_request_buffering on | off;

Умолчание scgi_request_buffering on;

Контекст http, server, location

Разрешает или запрещает использовать буферизацию тела запроса клиента.

on	тело запроса полностью читается от клиента перед отправкой запроса на SCGI-сервер.
off	тело запроса отправляется на SCGI-сервер сразу же по мере его поступления. В этом случае запрос не может быть передан следующему серверу, если Angie уже начал отправку тела запроса.

Если для отправки тела исходного запроса используется HTTP/1.1 и передача данных частями (chunked transfer encoding), то тело запроса буферизуется независимо от значения директивы.

scgi_send_timeout

Синтаксис scgi_send_timeout *время*;

Умолчание scgi_send_timeout 60s;

Контекст http, server, location

Задаёт таймаут при передаче запроса SCGI-серверу. Таймаут устанавливается не на всю передачу запроса, а только между двумя операциями записи. Если по истечении этого времени SCGI-сервер не примет новых данных, соединение закрывается.

scgi_socket_keepalive

Синтаксис scgi_socket_keepalive on | off;

Умолчание scgi_socket_keepalive off;

Контекст http, server, location

Конфигурирует поведение “TCP keepalive” для исходящих соединений к SCGI-серверу.

""	По умолчанию для сокета действуют настройки операционной системы.
on	для сокета включается параметр *SO_KEEPALIVE*

scgi_store

Синтаксис scgi_store on | off | *строка*;

Умолчание scgi_store off;

Контекст http, server, location

Разрешает сохранение на диск файлов.

on	сохраняет файлы в соответствии с путями, указанными в директивах alias или root
off	запрещает сохранение файлов

Имя файла можно задать явно с помощью строки с переменными:

```
scgi_store /data/www$original_uri;
```

Время изменения файлов выставляется согласно полученному полю “Last-Modified” в заголовке ответа. Ответ сначала записывается во временный файл, а потом этот файл переименовывается. Временный файл и постоянное место хранения ответа могут располагаться на разных файловых системах. Однако нужно учитывать, что в этом случае вместо дешёвой операции переименовывания в пределах одной файловой системы файл

копируется с одной файловой системы на другую. Поэтому лучше, если сохраняемые файлы будут находиться на той же файловой системе, что и каталог с временными файлами, задаваемый директивой scgi_temp_path для данного *location*.

Директиву можно использовать для создания локальных копий статических неизменяемых файлов:

```
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

Синтаксис scgi_store_access *пользователи*:*права* …;

Умолчание scgi_store_access user:rw;

Контекст http, server, location

Задаёт права доступа для создаваемых файлов и каталогов, например,

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

Если заданы какие-либо права для *group* или *all*, то права для user указывать необязательно:

```
scgi_store_access group:rw all:r;
```

scgi_temp_file_write_size

Синтаксис scgi_temp_file_write_size *размер*;

Умолчание scgi_temp_file_write_size 8k|16k;

Контекст http, server, location

Ограничивает размер данных, сбрасываемых во временный файл за один раз, при включённой буферизации ответов SCGI-сервера во временные файлы. По умолчанию размер ограничен двумя буферами, заданными директивами scgi_buffer_size и scgi_buffers. Максимальный размер временного файла задаётся директивой scgi_max_temp_file_size.

scgi_temp_path

Синтаксис scgi_temp_path *путь* [*уровень1* [*уровень2* [*уровень3*]]]`;

Умолчание scgi_temp_path scgi_temp;

Контекст http, server, location

Задаёт имя каталога для хранения временных файлов с данными, полученными от SCGI-серверов. В каталоге может использоваться иерархия подкаталогов до трёх уровней. Например, при такой конфигурации

```
scgi_temp_path /spool/angie/scgi_temp 1 2;
```

временный файл будет следующего вида:

```
/spool/angie/scgi_temp/7/45/00000123457
```

См. также параметр *use_temp_path* директивы scgi_cache_path.

Модуль http_secure_link

Позволяет проверять аутентичность запрашиваемых ссылок, защищать ресурсы от несанкционированного доступа, а также ограничивать срок действия ссылок.

Правильность запрашиваемой ссылки проверяется сравнением переданного в запросе значения контрольной суммы со значением, вычисляемым для запроса. Если ссылка имеет ограниченный срок действия и он истёк, ссылка считается устаревшей. Результат этих проверок делается доступным в переменной $secure_link.

Модуль реализует два альтернативных режима работы. В первом режиме, который включается директивой secure_link_secret, можно проверить аутентичность запрашиваемых ссылок и защитить их от несанкционированного доступа. Второй режим включается директивами secure_link и secure_link_md5, и позволяет также ограничить срок действия ссылок.

По умолчанию этот модуль не собирается, его сборку необходимо разрешить с помощью конфигурационного параметра *--with-http_secure_link_module*.

Директивы

secure_link

Синтаксис secure_link *выражение*;

Умолчание —

Контекст http, server, location

Задаёт строку с переменными, из которой будет выделено значение контрольной суммы и время действия ссылки.

Используемые в выражении переменные обычно связаны с запросом; см. пример ниже.

Выделенное из строки значение контрольной суммы сравнивается со значением MD5-хэша, вычисляемым для выражения, заданного директивой secure_link_md5.

Если контрольные суммы не совпадают, значением переменной $secure_link становится пустая строка. Если контрольные суммы совпадают, проверяется время действия ссылки.

Если срок действия ссылки задан и истёк, переменная $secure_link получает значение 0. В противном случае она получает значение 1. Значение MD5-хэш передаётся в запросе закодированным в base64url.

Если ссылка имеет ограниченный срок действия, время её действия задаётся в секундах с начала эпохи (1 января 1970 года 00:00:00 GMT). Значение указывается в выражении после MD5-хэша и отделяется от него запятой. Время действия ссылки, переданное в запросе, делается доступным в переменной $secure_link_expires для использования в директиве secure_link_md5. Если время действия ссылки не задано, ссылка имеет неограниченный срок действия.

secure_link_md5

Синтаксис secure_link_md5 *выражение*;

Умолчание —

Контекст http, server, location

Задаёт выражение, для которого считается значение MD5-хэш, сравниваемое с переданным в запросе.

Выражение должно содержать защищаемую часть ссылки (ресурс) и секретную составляющую. Если ссылка имеет ограниченный срок действия, выражение также должно содержать $secure_link_expires.

Для предотвращения несанкционированного доступа выражение может содержать информацию о клиенте, например, его адрес и версию браузера.

Пример:

```
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;
    }

#    ...
}
```

Ссылка “/s/link?md5=_e4Nc3iduzkWRm01TBBNYw&expires=2147483647” ограничивает доступ к “/s/link” для клиента с IP-адресом 127.0.0.1. Ссылка также имеет ограниченный срок действия до 19 января 2038 года (GMT).

Значение аргумента запроса md5 на UNIX можно получить так:

```
echo -n '2147483647/s/link127.0.0.1 secret' | \
    openssl md5 -binary | openssl base64 | tr +/ -_ | tr -d =
```

secure_link_secret

Синтаксис secure_link_secret *слово*;

Умолчание —

Контекст location

Задаёт секретное слово для проверки аутентичности запрашиваемых ссылок.

Полный URI запрашиваемой ссылки выглядит так:

```
/префикс/хэш/ссылка
```

где хэш — MD5-хэш в шестнадцатеричном виде, вычисленный для конкатенации ссылки и секретного слова, а префикс — произвольная строка без слэшей.

Если запрашиваемая ссылка проходит проверку на аутентичность, значением переменной $secure_link становится ссылка, выделенная из URI запроса. В противном случае значением переменной $secure_link становится пустая строка.

Пример:

```
location /p/ {
    secure_link_secret secret;

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

    rewrite ^ /secure/$secure_link;
}

location /secure/ {
    internal;
}
```

По запросу “/p/5e814704a28d9bc1914ff19fa0c4a00a/link” будет выполнено внутреннее перенаправление на “/secure/link”.

Значение хэша для данного примера на UNIX можно получить так:

```
echo -n 'linksecret' | openssl md5 -hex
```

Встроенные переменные

$secure_link

Результат проверки ссылки. Конкретное значение зависит от выбранного режима работы.

$secure_link_expires

Время действия ссылки, переданное в запросе. Предназначено исключительно для использования в директиве secure_link_md5.

Модуль http_slice

Фильтр, который разбивает запрос на подзапросы, каждый из которых возвращает определённый диапазон ответа. Фильтр обеспечивает более эффективное кэширование больших ответов.

По умолчанию этот модуль не собирается, его сборку необходимо разрешить с помощью конфигурационного параметра *--with-http_slice_module*.

Пример конфигурации

```
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;
}
```

Директивы

slice

Синтаксис slice *размер*;

Умолчание slice 0;

Контекст http, server, location

Задаёт размер фрагмента. Нулевое значение запрещает разбиение ответов на фрагменты.

Предупреждение

Обратите внимание, что слишком низкое значение может привести к излишнему потреблению памяти и открытию большого количества файлов.

Для того, чтобы подзапрос вернул необходимый диапазон, переменная $slice_range должна быть передана на проксируемый сервер в качестве поля “Range“ заголовка запроса. Если включено кэширование, то необходимо добавить $slice_range в ключ кэширования и включить кэширование ответов с кодом 206.

Встроенные переменные

$slice_range

текущий диапазон фрагмента в формате HTTP byte range, например *bytes=0-1048575*.

Модуль http_split_clients

Создаёт переменные для A/B тестирования (также известного как “split-тестирование”).

Пример конфигурации

```
http {
    split_clients "${remote_addr}AAA" $variant {
                  0.5%               .one;
                  2.0%               .two;
                  *                  "";
    }

    server {
        location / {
            index index${variant}.html;
```

Директивы

split_clients

Синтаксис split_clients *строка $переменная* { … }

Умолчание —

Контекст http

Создаёт переменную для A/B тестирования, например:

```
split_clients "${remote_addr}AAA" $variant {
               0.5%               .one;
               2.0%               .two;
               *                  "";
}
```

Значение исходной строки хэшируется с помощью MurmurHash2. В приведённом примере при значениях хэша от 0 до 21474835 (0.5%) переменная *$variant* получит значение «*.one*». При значениях хэша от 21474836 до 107374180 (2%) — «*.two*». И при значениях хэша от 107374181 до 4294967295 — «» (пустая строка).

Модуль http_ssi

Фильтр, обрабатывающий команды SSI (Server Side Includes) в проходящих через него ответах.

Пример конфигурации

```
location / {
    ssi on;
#    ...
```

```
}
```

Директивы

ssi

Синтаксис ssi on | off;

Умолчание ssi off;

Контекст http, server, location, if в location

Разрешает или запрещает обработку команд SSI в ответах.

ssi_last_modified

Синтаксис ssi_last_modified on | off;

Умолчание ssi_last_modified off;

Контекст http, server, location

Позволяет сохранить поле заголовка “Last-Modified” исходного ответа во время обработки SSI для лучшего кэширования ответов.

По умолчанию поле заголовка удаляется, так как содержимое ответа изменяется во время обработки и может содержать динамически созданные элементы или части, которые изменились независимо от исходного ответа.

ssi_min_file_chunk

Синтаксис ssi_min_file_chunk *размер*;

Умолчание ssi_min_file_chunk 1k;

Контекст http, server, location

Задаёт минимальный размер частей ответа, хранящихся на диске, начиная с которого имеет смысл посылать их с помощью sendfile.

ssi_silent_errors

Синтаксис ssi_silent_errors on | off;

Умолчание ssi_silent_errors off;

Контекст http, server, location

Разрешает не выводить строку “*[an error occurred while processing the directive]*”, если во время обработки SSI произошла ошибка.

ssi_types

Синтаксис ssi_types mime-*тип* …;

Умолчание ssi_types text/html;

Контекст http, server, location

Разрешает обработку команд SSI в ответах с указанными MIME-типами в дополнение к “*text/html*”. Специальное значение “*” соответствует любому MIME-типу.

ssi_value_length

Синтаксис ssi_value_length *длина*;

Умолчание ssi_value_length 256;

Контекст http, server, location

Задаёт максимальную длину значений параметров в SSI-командах.

Команды SSI

Общий формат команд SSI такой:

<!--# команда параметр1=значение1 параметр2=значение2 ... -->

Поддерживаются следующие команды:

block

Описывает блок, который можно использовать как заглушку в команде *include*. Внутри блока могут быть другие команды SSI. Параметр команды:

name

имя блока.

Пример:

```
<!--# block name="one" -->
```

заглушка

```
<!--# endblock -->
```

config

Задаёт некоторые параметры, используемые при обработке SSI, а именно:

errmsg

строка, выводящаяся при ошибке во время обработки SSI. По умолчанию выводится такая строка:

```
[an error occurred while processing the directive]
```

timefmt

строка формата, передаваемая функции strftime() для вывода даты и времени. По умолчанию используется такой формат:

```
«%A, %d-%b-%Y %H:%M:%S %Z»
```

Для вывода времени в секундах подходит формат “%s”.

echo

Выводит значение переменной. Параметры команды:

var

имя переменной.

encoding

способ кодирования. Возможны три значения — *none*, *url* и *entity*. По умолчанию используется entity.

default

нестандартный параметр, задающий строку, которая выводится, если переменная не определена.
По умолчанию выводится строка “*(none)*”.

Команда

```
<!--# echo var="name" default="нет" -->
```

заменяет такую последовательность команд:

```
<!--# if expr="$name" --><!--# echo var="name" --><!--#
     else -->нет<!--# endif -->
```

if

Выполняет условное включение. Поддерживаются следующие команды:

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

На данный момент поддерживается только один уровень вложенности. Параметр команды:

expr

выражение. В выражении может быть:

- проверка существования переменной:

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

- сравнение переменной с текстом:

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

- сравнение переменной с регулярным выражением:

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

Если в *text* встречаются переменные, то производится подстановка их значений. В регулярном выражении можно задать позиционные и именованные выделения, а затем использовать их через переменные, например:

```
<!--# if expr="$name = /(.+)@(?P<domain>.+)/" -->
  <!--# echo var="1" -->
  <!--# echo var="domain" -->
<!--# endif -->
```

include

Включает в ответ результат другого запроса. Параметры команды:

file

задаёт включаемый файл, например:

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

virtual

задаёт включаемый запрос, например:

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

Несколько запросов, указанных на одной странице и обрабатываемых проксируемыми или FastCGI/uwsgi/SCGI/gRPC-серверами, работают параллельно. Если нужна последовательная обработка, следует воспользоваться параметром *wait*.

stub

нестандартный параметр, задающий имя блока, содержимое которого будет выведено, если тело ответа на включаемый запрос пустое или если при исполнении запроса произошла ошибка, например:

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

Содержимое замещающего блока обрабатывается в контексте включаемого запроса.

wait

нестандартный параметр, указывающий, нужно ли ждать полного исполнения данного запроса, прежде чем продолжать выполнение SSI, например:

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

set

нестандартный параметр, указывающий, что удачный результат выполнения запроса нужно записать в заданную переменную, например:

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

Максимальный размер ответа задаётся директивой subrequest_output_buffer_size:

```
location /remote/ {
    subrequest_output_buffer_size 64k;
#    ...
}
```

set

Присваивает значение переменной. Параметры команды:

var

имя переменной.

value

значение переменной. Если в присваиваемом значении есть переменные, то производится подстановка их значений.

Встроенные переменные

$date_local

текущее время в локальной временной зоне. Формат задаётся командой *config* с параметром *timefmt*.

$date_gmt

текущее время в GMT. Формат задаётся командой *config* с параметром *timefmt*.

Модуль http_ssl

Обеспечивает работу по протоколу HTTPS.

По умолчанию этот модуль не собирается, его сборку необходимо разрешить с помощью конфигурационного параметра *--with-http_ssl_module*.

Примечание

Для сборки и работы этого модуля нужна библиотека OpenSSL.

Пример конфигурации

Для уменьшения загрузки процессора рекомендуется

- установить число рабочих процессов равным числу процессоров,
- разрешить keep-alive соединения,
- включить разделяемый кэш сессий,
- выключить встроенный кэш сессий
- и, возможно, увеличить время жизни сессии (по умолчанию 5 минут):

```
worker_processes auto;

http {
```

```
    # ...

    server {
        listen               443 ssl;
        keepalive_timeout    70;

        ssl_protocols        TLSv1 TLSv1.1 TLSv1.2;
        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;

        # ...
    }
```

Директивы

ssl_buffer_size

Синтаксис ssl_buffer_size *размер*;

Умолчание ssl_buffer_size 16k;

Контекст http, server

Задаёт размер буфера, используемого при отправке данных.

По умолчанию размер буфера равен 16k, что соответствует минимальным накладным расходам при передаче больших ответов. С целью минимизации времени получения начала ответа (Time To First Byte) может быть полезно использовать меньшие значения, например:

```
ssl_buffer_size 4k;
```

ssl_certificate

Синтаксис ssl_certificate *файл*;

Умолчание —

Контекст http, server

Указывает файл с сертификатом в формате PEM для данного виртуального сервера. Если вместе с основным сертификатом нужно указать промежуточные, то они должны находиться в этом же файле в следующем порядке: сначала основной сертификат, а затем промежуточные. В этом же файле может находиться секретный ключ в формате PEM.

Эта директива может быть указана несколько раз для загрузки сертификатов разных типов, например RSA и 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;

    # ...
}
```

Возможность задавать отдельные цепочки сертификатов для разных сертификатов есть только в OpenSSL 1.0.2 и выше. Для более старых версий следует указывать только одну цепочку сертификатов.

Примечание

В имени файла можно использовать переменные при использовании OpenSSL 1.0.2 и выше:

```
ssl_certificate     $ssl_server_name.crt;
ssl_certificate_key $ssl_server_name.key;
```

При использовании переменных сертификат загружается при каждой операции SSL handshake, что может отрицательно влиять на производительность.

Вместо файла можно указать значение “data:*$переменная*“, при котором сертификат загружается из переменной без использования промежуточных файлов.

Ненадлежащее использование подобного синтаксиса может быть небезопасно, например данные секретного ключа могут попасть в лог ошибок.

Внимание

Нужно иметь в виду, что из-за ограничения протокола HTTPS для максимальной совместимости виртуальные серверы должны слушать на разных IP-адресах.

ssl_certificate_key

Синтаксис ssl_certificate_key *файл*;

Умолчание —

Контекст http, server

Указывает файл с секретным ключом в формате PEM для данного виртуального сервера.

Примечание

В имени файла можно использовать переменные при использовании OpenSSL 1.0.2 и выше.

Вместо файла можно указать значение “engine:*имя*:id“, которое загружает ключ с указанным *id* из OpenSSL engine с заданным именем.

Вместо файла можно указать значение “data:*$переменная*“, при котором секретный ключ загружается из переменной без использования промежуточных файлов. При этом следует учитывать, что ненадлежащее использование подобного синтаксиса может быть небезопасно, например данные секретного ключа могут попасть в лог ошибок.

ssl_ciphers

Синтаксис ssl_ciphers *шифры*;

Умолчание ssl_ciphers HIGH:!aNULL:!MD5;

Контекст http, server

Описывает разрешённые шифры. Шифры задаются в формате, поддерживаемом библиотекой OpenSSL, например:

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

Полный список можно посмотреть с помощью команды “*openssl ciphers*”.

ssl_client_certificate

Синтаксис ssl_client_certificate *файл*;

Умолчание —

Контекст http, server

Указывает файл с доверенными сертификатами CA в формате PEM, которые используются для проверки клиентских сертификатов и ответов OCSP, если включён ssl_stapling.

Список сертификатов будет отправляться клиентам. Если это нежелательно, можно воспользоваться директивой ssl_trusted_certificate.

ssl_conf_command

Синтаксис ssl_conf_command *имя значение*;

Умолчание —

Контекст http, server

Задаёт произвольные конфигурационные команды OpenSSL.

Примечание

Директива поддерживается при использовании OpenSSL 1.0.2 и выше.

На одном уровне может быть указано несколько директив *ssl_conf_command*:

```
ssl_conf_command Options PrioritizeChaCha;
ssl_conf_command Ciphersuites TLS_CHACHA20_POLY1305_SHA256;
```

Директивы наследуются с предыдущего уровня конфигурации при условии, что на данном уровне не описаны свои директивы *ssl_conf_command*.

Внимание

Изменение настроек OpenSSL напрямую может привести к неожиданному поведению.

ssl_crl

Синтаксис ssl_crl *файл*;

Умолчание —

Контекст http, server

Указывает файл с отозванными сертификатами (CRL) в формате PEM, используемыми для проверки клиентских сертификатов.

ssl_dhparam

Синтаксис ssl_dhparam *файл*;

Умолчание —

Контекст http, server

Указывает файл с параметрами для DHE-шифров.

Внимание

По умолчанию параметры не заданы, и соответственно DHE-шифры не будут использоваться.

ssl_early_data

Синтаксис ssl_early_data on | off;

Умолчание ssl_early_data off;

Контекст http, server

Разрешает или запрещает TLS 1.3 early data.

Запросы, отправленные внутри early data, могут быть подвержены атакам повторного воспроизведения (replay). Для защиты от подобных атак на уровне приложения необходимо использовать переменную $ssl_early_data.

```
proxy_set_header Early-Data $ssl_early_data;
```

Примечание

Директива поддерживается при использовании OpenSSL 1.1.1 и выше или BoringSSL.

ssl_ecdh_curve

Синтаксис ssl_ecdh_curve *кривая*;

Умолчание ssl_ecdh_curve auto;

Контекст http, server

Задаёт кривую для ECDHE-шифров.

Примечание

При использовании OpenSSL 1.0.2 и выше можно указывать несколько кривых, например:

```
ssl_ecdh_curve prime256v1:secp384r1;
```

Специальное значение auto соответствует встроенному в библиотеку OpenSSL списку кривых для OpenSSL 1.0.2 и выше, или *prime256v1* для более старых версий.

Примечание

При использовании OpenSSL 1.0.2 и выше директива задаёт список кривых, поддерживаемых сервером. Поэтому для работы ECDSA-сертификатов важно, чтобы список включал кривые, используемые в сертификатах.

ssl_ocsp

Синтаксис ssl_ocsp on | off | leaf;

Умолчание ssl_ocsp off;

Контекст http, server

Включает проверку OCSP для цепочки клиентских сертификатов. Параметр leaf включает проверку только клиентского сертификата.

Для работы проверки OCSP необходимо дополнительно установить значение директивы ssl_verify_client в on или optional.

Для преобразования имени хоста OCSP responder'а в адрес необходимо дополнительно задать директиву resolver.

Пример:

```
ssl_verify_client on;
ssl_ocsp          on;
resolver          192.0.2.1;
```

ssl_ocsp_cache

Синтаксис ssl_ocsp_cache off | [shared:*имя*:*размер*];

Умолчание ssl_ocsp_cache off;

Контекст http, server

Задаёт имя и размер кэша, который хранит статус клиентских сертификатов для проверки OCSP-ответов. Кэш разделяется между всеми рабочими процессами. Кэш с одинаковым названием может использоваться в нескольких виртуальных серверах.

Параметр off запрещает использование кэша.

ssl_ocsp_responder

Синтаксис ssl_ocsp_responder url;

Умолчание —

Контекст http, server

Переопределяет URL OCSP responder'а, указанный в расширении сертификата "Authority Information Access" для проверки клиентских сертификатов.

Поддерживаются только "*http://*" OCSP responder'ы:

```
ssl_ocsp_responder http://ocsp.example.com/;
```

ssl_password_file

Синтаксис ssl_password_file *файл*;

Умолчание —

Контекст http, server

Задаёт файл с паролями от секретных ключей, где каждый пароль указан на отдельной строке. Пароли применяются по очереди в момент загрузки ключа.

Пример:

```
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;

        # вместо файла можно указать именованный канал
        ssl_password_file /etc/keys/fifo;
        ssl_certificate_key /etc/keys/second.key;
    }
}
```

ssl_prefer_server_ciphers

Синтаксис ssl_prefer_server_ciphers on | off;

Умолчание ssl_prefer_server_ciphers off;

Контекст http, server

При использовании протоколов SSLv3 и TLS устанавливает приоритет серверных шифров над клиентскими.

ssl_protocols

Синтаксис ssl_protocols [SSLv2] [SSLv3] [TLSv1] [TLSv1.1] [TLSv1.2] [TLSv1.3];

Умолчание ssl_protocols TLSv1 TLSv1.1 TLSv1.2;

Контекст http, server

Разрешает указанные протоколы.

Примечание

Параметры TLSv1.1 и TLSv1.2 работают только при использовании OpenSSL 1.0.1 и выше.

Параметр TLSv1.3 работает только при использовании OpenSSL 1.1.1 и выше.

ssl_reject_handshake

Синтаксис ssl_reject_handshake on | off;

Умолчание ssl_reject_handshake off;

Контекст http, server

Если включено, то операции SSL handshake в блоке server будут отклонены.

Например, в этой конфигурации отклоняются все операции SSL handshake с именем сервера, отличным от *example.com*:

```
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

Синтаксис ssl_session_cache off | none | [builtin[:*размер*]] [shared:*название*:*размер*];

Умолчание ssl_session_cache none;

Контекст http, server

Задаёт тип и размеры кэшей для хранения параметров сессий. Тип кэша может быть следующим:

off	жёсткое запрещение использования кэша сессий: Angie явно сообщает клиенту, что сессии не могут использоваться повторно.
none	мягкое запрещение использования кэша сессий: Angie сообщает клиенту, что сессии могут использоваться повторно, но на самом деле не хранит параметры сессии в кэше.
builtin	встроенный в OpenSSL кэш, используется в рамках только одного рабочего процесса. Размер кэша задаётся в сессиях. Если размер не

	задан, то он равен 20480 сессиям. Использование встроенного кэша может вести к фрагментации памяти.
shared	кэш, разделяемый между всеми рабочими процессами. Размер кэша задаётся в байтах, в 1 мегабайт может поместиться около 4000 сессий. У каждого разделяемого кэша должно быть произвольное название. Кэш с одинаковым названием может использоваться в нескольких виртуальных серверах. Также он используется для автоматического создания, хранения и периодического обновления ключей TLS session tickets, если они не указаны явно с помощью директивы ssl_session_ticket_key.

Можно использовать одновременно оба типа кэша, например:

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

однако использование только разделяемого кэша без встроенного должно быть более эффективным.

ssl_session_ticket_key

Синтаксис ssl_session_ticket_key *файл*;

Умолчание —

Контекст http, server

Задаёт файл с секретным ключом, применяемым при шифровании и расшифровании TLS session tickets. Директива необходима, если один и тот же ключ нужно использовать на нескольких серверах. По умолчанию используется случайно сгенерированный ключ.

Если указано несколько ключей, то только первый ключ используется для шифрования TLS session tickets. Это позволяет настроить ротацию ключей, например:

```
ssl_session_ticket_key current.key;
ssl_session_ticket_key previous.key;
```

Файл должен содержать 80 или 48 байт случайных данных и может быть создан следующей командой:

```
openssl rand 80 > ticket.key
```

В зависимости от размера файла для шифрования будет использоваться либо AES256 (для 80-байтных ключей), либо AES128 (для 48-байтных ключей).

ssl_session_tickets

Синтаксис ssl_session_tickets on | off;

Умолчание ssl_session_tickets on;

Контекст http, server

Разрешает или запрещает возобновление сессий при помощи TLS session tickets.

ssl_session_timeout

Синтаксис ssl_session_timeout *время*;

Умолчание ssl_session_timeout 5m;

Контекст http, server

Задаёт время, в течение которого клиент может повторно использовать параметры сессии.

ssl_stapling

Синтаксис ssl_stapling on | off;

Умолчание ssl_stapling off;

Контекст http, server

Разрешает или запрещает прикрепление OCSP-ответов сервером. Пример:

```
ssl_stapling on;
resolver 192.0.2.1;
```

Для работы OCSP stapling должен быть известен сертификат издателя сертификата сервера. Если в заданном директивой ssl_certificate файле не содержится промежуточных сертификатов, то сертификат издателя сертификата сервера следует поместить в файл, заданный директивой ssl_trusted_certificate.

Внимание

Для преобразования имени хоста OCSP responder'а в адрес необходимо дополнительно задать директиву resolver.

ssl_stapling_file

Синтаксис ssl_stapling_file *файл*;

Умолчание —

Контекст http, server

Если задано, то вместо опроса OCSP responder'а, указанного в сертификате сервера, ответ берётся из указанного файла.

Ответ должен быть в формате DER и может быть сгенерирован командой "*openssl ocsp*".

ssl_stapling_responder

Синтаксис ssl_stapling_responder url;

Умолчание —

Контекст http, server

Переопределяет URL OCSP responder'а, указанный в расширении сертификата "Authority Information Access".

Поддерживаются только “*http://*” OCSP responder’ы:

```
ssl_stapling_responder http://ocsp.example.com/;
```

ssl_stapling_verify

Синтаксис ssl_stapling_verify on | off;

Умолчание ssl_stapling_verify off;

Контекст http, server

Разрешает или запрещает проверку сервером ответов OCSP.

Для работоспособности проверки сертификат издателя сертификата сервера, корневой сертификат и все промежуточные сертификаты должны быть указаны как доверенные с помощью директивы ssl_trusted_certificate.

ssl_trusted_certificate

Синтаксис ssl_trusted_certificate *файл*;

Умолчание —

Контекст http, server

Задаёт файл с доверенными сертификатами CA в формате PEM, которые используются для проверки клиентских сертификатов и ответов OCSP, если включён ssl_stapling.

В отличие от ssl_client_certificate, список этих сертификатов не будет отправляться клиентам.

ssl_verify_client

Синтаксис ssl_verify_client on | off | optional | optional_no_ca;

Умолчание ssl_verify_client off;

Контекст http, server

Разрешает проверку клиентских сертификатов. Результат проверки доступен через переменную $ssl_client_verify.

optional	запрашивает клиентский сертификат, и если сертификат был предоставлен, проверяет его
optional_no_ca	запрашивает сертификат клиента, но не требует, чтобы он был подписан доверенным сертификатом CA. Это предназначено для случаев, когда фактическая проверка сертификата осуществляется внешним по отношению к Angie сервисом.

ssl_verify_depth

Синтаксис ssl_verify_depth *число*;

Умолчание ssl_verify_depth 1;

Контекст http, server

Устанавливает глубину проверки в цепочке клиентских сертификатов.

Обработка ошибок

Модуль http_ssl поддерживает несколько нестандартных кодов ошибок, которые можно использовать для перенаправления с помощью директивы error_page:

495	при проверке клиентского сертификата произошла ошибка;
496	клиент не предоставил требуемый сертификат;
497	обычный запрос был послан на порт HTTPS.

Перенаправление делается после того, как запрос полностью разобран и доступны такие переменные, как $request_uri, $uri, $args и другие переменные.

Встроенные переменные

Модуль *http_ssl* поддерживает встроенные переменные:

$ssl_alpn_protocol

возвращает протокол, выбранный при помощи ALPN во время операции SSL handshake, либо пустую строку.

$ssl_cipher

возвращает название используемого шифра для установленного SSL-соединения.

$ssl_ciphers

возвращает список шифров, поддерживаемых клиентом. Известные шифры указаны по имени, неизвестные указаны в шестнадцатеричном виде, например:

```
AES128-SHA:AES256-SHA:0x00ff
```

Примечание

Переменная полностью поддерживается при использовании OpenSSL версии 1.0.2 и выше. При использовании более старых версий переменная доступна только для новых сессий и может содержать только известные шифры.

$ssl_client_escaped_cert

возвращает клиентский сертификат в формате PEM (закодирован в формате urlencode) для установленного SSL-соединения.

$ssl_client_fingerprint

возвращает SHA1-отпечаток клиентского сертификата для установленного SSL-соединения.

$ssl_client_i_dn

возвращает строку “issuer DN” клиентского сертификата для установленного SSL-соединения согласно RFC 2253.

$ssl_client_i_dn_legacy

возвращает строку “issuer DN” клиентского сертификата для установленного SSL-соединения.

$ssl_client_raw_cert

возвращает клиентский сертификат для установленного SSL-соединения в формате PEM.

$ssl_client_s_dn

возвращает строку “subject DN” клиентского сертификата для установленного SSL-соединения согласно RFC 2253.

$ssl_client_s_dn_legacy

возвращает строку “subject DN” клиентского сертификата для установленного SSL-соединения.

$ssl_client_serial

возвращает серийный номер клиентского сертификата для установленного SSL-соединения.

$ssl_client_v_end

возвращает дату окончания срока действия клиентского сертификата.

$ssl_client_v_remain

возвращает число дней, оставшихся до истечения срока действия клиентского сертификата.

$ssl_client_v_start

возвращает дату начала срока действия клиентского сертификата.

$ssl_client_verify

возвращает результат проверки клиентского сертификата: “*SUCCESS*”, “*FAILED:reason*” и, если сертификат не был предоставлен, “*NONE*”.

$ssl_curve

возвращает согласованную кривую, использованную для обмена ключами во время операции SSL handshake. Известные кривые указаны по имени, неизвестные указаны в шестнадцатеричном виде, например:

```
prime256v1
```

Примечание

Переменная поддерживается при использовании OpenSSL версии 3.0 и выше. При использовании более старых версий значением переменной будет пустая строка.

$ssl_curves

возвращает список кривых, поддерживаемых клиентом. Известные кривые указаны по имени, неизвестные указаны в шестнадцатеричном виде, например:

```
0x001d:prime256v1:secp521r1:secp384r1
```

Примечание

Переменная поддерживается при использовании OpenSSL версии 1.0.2 и выше. При использовании более старых версий значением переменной будет пустая строка.

Переменная доступна только для новых сессий.

$ssl_early_data

возвращает “*1*”, если используется TLS 1.3 early data и операция handshake не завершена, иначе “”.

$ssl_protocol

возвращает протокол установленного SSL-соединения.

$ssl_server_name

возвращает имя сервера, запрошенное через SNI.

$ssl_session_id

возвращает идентификатор сессии установленного SSL-соединения.

$ssl_session_reused#

возвращает “*r*”, если сессия была использована повторно, иначе “.”.

Модуль http_stub_status

Предоставляет доступ к базовой информации о состоянии сервера.

По умолчанию этот модуль не собирается, его сборку необходимо разрешить с помощью конфигурационного параметра *--with-http_stub_status_module*.

Пример конфигурации

```
location = /basic_status {
    stub_status;
}
```

В данной конфигурации создаётся простая веб-страница с основной информацией о состоянии, которая может выглядеть следующим образом:

```
Active connections: 291
server accepts handled requests
 16630948 16630948 31070465
Reading: 6 Writing: 179 Waiting: 106
```

Директивы

stub_status

Синтаксис stub_status;

Умолчание —

Контекст server, location

Информация о состоянии будет доступна из данного location.

Данные

Доступна следующая информация:

Active connections

Текущее число активных клиентских соединений, включая Waiting-соединения.

Accepts

Суммарное число принятых клиентских соединений.

Handled

Суммарное число обработанных соединений. Как правило, значение этого параметра такое же, как accepts, если не достигнуто какое-нибудь системное ограничение (например, лимит worker_connections).

Requests

Суммарное число клиентских запросов.

Reading

Текущее число соединений, в которых Angie в настоящий момент читает заголовок запроса.

Writing

Текущее число соединений, в которых Angie в настоящий момент отвечает клиенту.

Waiting

Текущее число бездействующих клиентских соединений в ожидании запроса.

Встроенные переменные

$connections_active

то же, что и значение *Active connections*;

$connections_reading

то же, что и значение *Reading*;

$connections_writing

то же, что и значение *Writing*;

$connections_waiting

то же, что и значение *Waiting*.

Модуль http_sub

Фильтр, изменяющий в ответе одну заданную строку на другую.

По умолчанию этот модуль не собирается, его сборку необходимо разрешить с помощью конфигурационного параметра *--with-http_sub_module*.

Пример конфигурации

```
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;
}
```

Директивы

sub_filter

Синтаксис sub_filter *строка замена*;

Умолчание —

Контекст http, server, location

Задаёт строку, которую нужно заменить, и строку замены. Заменяемая строка проверяется без учёта регистра. В заменяемой строке и в строке замены можно использовать переменные. На одном уровне конфигурации может быть указано несколько директив *sub_filter*. Директивы наследуются с предыдущего уровня конфигурации при условии, что на данном уровне не описаны свои директивы *sub_filter*.

sub_filter_last_modified

Синтаксис sub_filter_last_modified on | off;

Умолчание sub_filter_last_modified off;

Контекст http, server, location

Позволяет сохранить поле заголовка “Last-Modified” исходного ответа во время замены для лучшего кэширования ответов.

По умолчанию поле заголовка удаляется, так как содержимое ответа изменяется во время обработки.

sub_filter_once

Синтаксис sub_filter_once on | off;

Умолчание sub_filter_once on;

Контекст http, server, location

Определяет, сколько раз нужно искать каждую из заменяемых строк: один раз или многократно.

sub_filter_types

Синтаксис sub_filter_types mime-*тип* …;

Умолчание sub_filter_types text/html;

Контекст http, server, location

Разрешает замену строк в ответах с указанными MIME-типами в дополнение к “*text/html*”. Специальное значение “*” соответствует любому MIME-типу.

Модуль http_upstream

Предоставляет контекст для описания группы серверов, которые могут использоваться в директивах proxy_pass, fastcgi_pass, uwsgi_pass, scgi_pass, memcached_pass и grpc_pass.

Пример конфигурации

```
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;
}

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

Директивы

upstream

Синтаксис upstream *имя* { … }

Умолчание —

Контекст http

Описывает группу серверов. Серверы могут слушать на разных портах. Кроме того, можно одновременно использовать серверы, слушающие на TCP- и UNIX-сокетах.

Пример:

```
upstream backend {
    server backend1.example.com weight=5;
    server 127.0.0.1:8080       max_fails=3 fail_timeout=30s;
    server unix:/tmp/backend3;

    server backup1.example.com  backup;
}
```

По умолчанию запросы распределяются по серверам циклически (в режиме round-robin) с учётом весов серверов. В вышеприведённом примере каждые 7 запросов будут распределены так: 5 запросов на backend1.example.com и по одному запросу на второй и третий серверы.

Если при попытке работы с сервером происходит ошибка, то запрос передаётся следующему серверу, и так далее до тех пор, пока не будут опробованы все работающие серверы. Если не удастся получить успешный ответ ни от одного из серверов, то клиенту будет возвращён результат работы с последним сервером.

server

Синтаксис server *адрес* [*параметры*];

Умолчание —

Контекст upstream

Задаёт адрес и другие параметры сервера. Адрес может быть указан в виде доменного имени или IP-адреса, и необязательного порта, или в виде пути UNIX-сокета, который указывается после префикса “*unix:*”. Если порт не указан, используется порт 80. Доменное имя, которому соответствует несколько IP-адресов, задаёт сразу несколько серверов.

Могут быть заданы следующие параметры:

weight= число	задаёт вес сервера по умолчанию 1.
max_conns= число	ограничивает максимальное число одновременных активных соединений к проксируемому серверу. Значение по умолчанию равно 0 и означает, что ограничения нет. Если группа не находится в зоне разделяемой памяти, то ограничение работает отдельно для каждого рабочего процесса.

Примечание

При включённых неактивных постоянных соединениях, нескольких рабочих процессах и зоне разделяемой памяти, суммарное число активных и неактивных соединений с проксируемым сервером может превышать значение *max_conns*.

max_fails= *число* — задаёт число неудачных попыток работы с сервером, которые должны произойти в течение времени, заданного параметром fail_timeout, чтобы сервер считался недоступным на период времени, также заданный параметром fail_timeout. Что считается неудачной попыткой, определяется директивами proxy_next_upstream, fastcgi_next_upstream, uwsgi_next_upstream, scgi_next_upstream, memcached_next_upstream и grpc_next_upstream.

max_fails=1	число попыток по умолчанию
max_fails=0	отключает учёт попыток

fail_timeout= *время* — задаёт:

- время, в течение которого должно произойти заданное число неудачных попыток работы с сервером для того, чтобы сервер считался недоступным;
- и время, в течение которого сервер будет считаться недоступным. По умолчанию параметр равен 10 секундам.

backup	помечает сервер как запасной. На него будут передаваться запросы в случае, если не работают основные серверы.
down	помечает сервер как постоянно недоступный.

Внимание

Параметр backup нельзя использовать совместно с методами балансировки нагрузки hash, ip_hash и random.

resolve	позволяет отслеживать изменения списка IP-адресов, соответствующего доменному имени, и обновлять его без перезагрузки конфигурации. Для работы этого параметра директивы *resolver* и *resolver_timeout* должны быть заданы в блоке upstream или унаследованы из блока http.
service= *имя*	включает преобразование SRV-записей DNS и задаёт имя сервиса. Для работы параметра необходимо указать параметр *resolve* для сервера и не указывать порт сервера.

zone

Синтаксис zone *имя* [*размер*];

Умолчание —

Контекст upstream

Задаёт имя и размер зоны разделяемой памяти, в которой хранятся конфигурация группы и её рабочее состояние, разделяемые между рабочими процессами. В одной и той же зоне могут быть сразу несколько групп. В этом случае достаточно указать размер только один раз.

hash

Синтаксис hash *ключ* [consistent];

Умолчание —

Контекст upstream

Задаёт метод балансировки нагрузки для группы, при котором соответствие клиента серверу определяется при помощи хэшированного значения ключа. В качестве ключа может использоваться текст, переменные и их комбинации. Следует отметить, что любое добавление или удаление серверов в группе может привести к перераспределению большинства ключей на другие серверы. Метод совместим с библиотекой Perl Cache::Memcached.

Если задан параметр consistent, то вместо вышеописанного метода будет использоваться метод консистентного хэширования ketama. Метод гарантирует, что при добавлении сервера в группу или его удалении на другие серверы будет перераспределено минимальное число ключей. Применение метода для кэширующих серверов обеспечивает

больший процент попаданий в кэш. Метод совместим с библиотекой Perl Cache::Memcached::Fast при значении параметра ketama_points равным 160.

ip_hash

Синтаксис ip_hash;

Умолчание —

Контекст upstream

Задаёт для группы метод балансировки нагрузки, при котором запросы распределяются по серверам на основе IP-адресов клиентов. В качестве ключа для хэширования используются первые три октета IPv4-адреса клиента или IPv6-адрес клиента целиком. Метод гарантирует, что запросы одного и того же клиента будут всегда передаваться на один и тот же сервер. Если же этот сервер будет считаться недоступным, то запросы этого клиента будут передаваться на другой сервер. С большой долей вероятности это также будет один и тот же сервер.

Если один из серверов нужно убрать на некоторое время, то для сохранения текущего хэширования IP-адресов клиентов этот сервер нужно пометить параметром down:

```
upstream backend {
    ip_hash;

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

keepalive

Синтаксис keepalive *соединения*;

Умолчание —

Контекст upstream

Задействует кэш соединений для группы серверов.

Параметр соединения устанавливает максимальное число неактивных постоянных соединений с серверами группы, которые будут сохраняться в кэше каждого рабочего процесса. При превышении этого числа наиболее давно не используемые соединения закрываются.

Примечание

Следует особо отметить, что директива *keepalive* не ограничивает общее число соединений с серверами группы, которые рабочие процессы Angie могут открыть. Параметр соединения следует устанавливать достаточно консервативно, чтобы серверы группы по-прежнему могли обрабатывать новые входящие соединения.

Внимание

При использовании методов балансировки нагрузки, отличных от стандартного round-robin, следует указать их до директивы *keepalive*.

Пример конфигурации группы серверов memcached с постоянными соединениями:

```
upstream memcached_backend {
    server 127.0.0.1:11211;
    server 10.0.0.2:11211;

    keepalive 32;
}

server {
    #...

    location /memcached/ {
        set $memcached_key $uri;
        memcached_pass memcached_backend;
    }

}
```

Для HTTP директиву proxy_http_version следует установить в “1.1”, а поле заголовка “Connection” — очистить:

```
upstream http_backend {
    server 127.0.0.1:8080;

    keepalive 16;
}

server {
    #...

    location /http/ {
        proxy_pass http://http_backend;
        proxy_http_version 1.1;
        proxy_set_header Connection "";
    #     ...
    }
}
```

Примечание

Хоть это и не рекомендуется, но также возможно использование постоянных соединений с HTTP/1.0, путём передачи поля заголовка “Connection: Keep-Alive” серверу группы.

Для работы постоянных соединений с FastCGI-серверами потребуется включить директиву fastcgi_keep_conn:

```
upstream fastcgi_backend {
    server 127.0.0.1:9000;
```

```
    keepalive 8;
}

server {
    #...

    location /fastcgi/ {
        fastcgi_pass fastcgi_backend;
        fastcgi_keep_conn on;
    #     ...
    }
}
```

Примечание

Протоколы SCGI и uwsgi не определяют семантику постоянных соединений.

keepalive_requests

Синтаксис keepalive_requests *число*;

Умолчание keepalive_requests 1000;

Контекст upstream

Задаёт максимальное число запросов, которые можно сделать по одному постоянному соединению. После того как сделано максимальное число запросов, соединение закрывается.

Периодическое закрытие соединений необходимо для освобождения памяти, выделенной под конкретные соединения. Поэтому использование слишком большого максимального числа запросов может приводить к чрезмерному потреблению памяти и не рекомендуется.

keepalive_time

Синтаксис keepalive_time *время*;

Умолчание keepalive_time 1h;

Контекст upstream

Ограничивает максимальное время, в течение которого могут обрабатываться запросы в рамках постоянного соединения. По достижении заданного времени соединение закрывается после обработки очередного запроса.

keepalive_timeout

Синтаксис keepalive_timeout *время*;

Умолчание keepalive_timeout 60s;

Контекст upstream

Задаёт таймаут, в течение которого неактивное постоянное соединение с сервером группы не будет закрыто.

least_conn

Синтаксис least_conn;

Умолчание —

Контекст upstream

Задаёт для группы метод балансировки нагрузки, при котором запрос передаётся серверу с наименьшим числом активных соединений, с учётом весов серверов. Если подходит сразу несколько серверов, они выбираются циклически (в режиме round-robin) с учётом их весов.

random

Синтаксис random [two];

Умолчание —

Контекст upstream

Задаёт для группы метод балансировки нагрузки, при котором запрос передаётся случайно выбранному серверу, с учётом весов серверов.

Если указан необязательный параметр two, Angie случайным образом выбирает два сервера, из которых выбирает сервер, используя метод least_conn, при котором запрос передаётся на сервер с наименьшим количеством активных соединений.

resolver

Синтаксис resolver *адрес* … [valid= *время*] [ipv4=on|off] [ipv6=on|off] [status_zone= *зона*];

Умолчание —

Контекст upstream

Задаёт серверы DNS, используемые для преобразования имён вышестоящих серверов в адреса, например:

```
resolver 127.0.0.1 [::1]:5353;
```

Адрес может быть указан в виде доменного имени или IP-адреса, и необязательного порта. Если порт не указан, используется порт 53. Серверы DNS опрашиваются циклически.

По умолчанию Angie кэширует ответы, используя значение TTL из ответа.

valid	*необязательный* параметр, позволяет переопределить срок кэширования ответа

```
resolver 127.0.0.1 [::1]:5353 valid=30s;
```

По умолчанию Angie будет искать как IPv4-, так и IPv6-адреса при преобразовании имён в адреса.

ipv4=off	запрещает поиск IPv4-адресов
ipv6=off	запрещает поиск IPv6-адресов
status_zone	*необязательный* параметр, включает сбор информации о запросах и ответах сервера DNS в указанной зоне

Примечание

Для предотвращения DNS-спуфинга рекомендуется использовать DNS-серверы в защищённой доверенной локальной сети.

resolver_timeout

Синтаксис resolver_timeout *время;*

Умолчание resolver_timeout 30s;

Контекст upstream

Задаёт таймаут для преобразования имени в адрес, например:

```
resolver_timeout 5s;
```

Встроенные переменные

Модуль http_upstream поддерживает следующие встроенные переменные:

$upstream_addr

хранит IP-адрес и порт или путь к UNIX-сокету сервера группы. Если при обработке запроса были сделаны обращения к нескольким серверам, то их адреса разделяются запятой, например:

```
192.168.1.1:80, 192.168.1.2:80, unix:/tmp/sock
```

Если произошло внутреннее перенаправление от одной группы серверов на другую с помощью “X-Accel-Redirect” или error_page, то адреса, соответствующие разным группам серверов, разделяются двоеточием, например:

```
192.168.1.1:80, 192.168.1.2:80, unix:/tmp/sock : 192.168.10.1:80, 192.168.10.2:80
```

Если сервер не может быть выбран, то переменная хранит *имя* группы серверов.

$upstream_bytes_received

число байт, полученных от сервера группы. Значения нескольких соединений разделяются запятыми и двоеточиями подобно адресам в переменной $upstream_addr.

$upstream_bytes_sent

число байт, переданных на сервер группы. Значения нескольких соединений разделяются запятыми и двоеточиями подобно адресам в переменной $upstream_addr.

$upstream_cache_status

хранит статус доступа к кэшу ответов. Статус может быть одним из “*MISS*”, “*BYPASS*”, “*EXPIRED*”, “*STALE*”, “*UPDATING*”, “*REVALIDATED*” или “*HIT*”.

$upstream_connect_time

хранит время, затраченное на установление соединения с сервером группы; время хранится в секундах с точностью до миллисекунд. В случае SSL, включает в себя время, потраченное на handshake. Времена нескольких соединений разделяются запятыми и двоеточиями подобно адресам в переменной $upstream_addr.

$upstream_cookie_ *имя*

кука с указанным именем, переданная сервером группы в поле “Set-Cookie” заголовка ответа. Необходимо иметь в виду, что куки запоминаются только из ответа последнего сервера.

$upstream_header_time

хранит время, затраченное на получение заголовка ответа от сервера группы; время хранится в секундах с точностью до миллисекунд. Времена нескольких ответов разделяются запятыми и двоеточиями подобно адресам в переменной $upstream_addr.

$upstream_http_ *имя*

хранят поля заголовка ответа сервера. Например, поле заголовка ответа “Server” доступно в переменной *$upstream_http_server*. Правила преобразования имён полей заголовка ответа в имена переменных такие же, как для переменных с префиксом “$http_”. Необходимо иметь в виду, что поля заголовка запоминаются только из ответа последнего сервера.

$upstream_response_length

хранит длину ответа, полученного от сервера группы; длина хранится в байтах. Длины нескольких ответов разделяются запятыми и двоеточиями подобно адресам в переменной $upstream_addr.

$upstream_response_time

хранит время, затраченное на получение ответа от сервера группы; время хранится в секундах с точностью до миллисекунд. Времена нескольких ответов разделяются запятыми и двоеточиями подобно адресам в переменной $upstream_addr.

$upstream_status

хранит статус ответа, полученного от сервера группы. Статусы нескольких ответов разделяются запятыми и двоеточиями подобно адресам в переменной $upstream_addr. Если сервер не может быть выбран, то переменная хранит статус 502 (Bad Gateway).

$upstream_trailer_ *имя*

хранит поля из конца ответа, полученного от сервера группы.

Модуль http_userid

Выдаёт куки для идентификации клиентов. Для записи в лог полученных и выданных кук можно использовать встроенные переменные $uid_got и $uid_set. Модуль совместим с модулем mod_uid для Apache.

Пример конфигурации

```
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"';
```

Директивы

userid

Синтаксис userid on | v1 | log | off;

Умолчание userid off;

Контекст http, server, location

Разрешает или запрещает выдачу кук и запись приходящих кук в лог:

on	разрешает выдачу кук версии 2 и запись приходящих кук в лог;
v1	разрешает выдачу кук версии 1 и запись приходящих кук в лог;
log	запрещает выдачу кук, но разрешает запись приходящих кук в лог;
off	запрещает выдачу кук и запись приходящих кук в лог.

userid_domain

Синтаксис userid_domain *имя* | none;

Умолчание userid_domain none;

Контекст http, server, location

Задаёт домен, для которого устанавливается кука. Параметр none запрещает выдавать домен для куки.

userid_expires

Синтаксис userid_expires *время* | max | off;

Умолчание userid_expires off;

Контекст http, server, location

Задаёт время, в течение которого браузер должен хранить куку. Параметр max устанавливает срок хранения куки до 31 декабря 2037 года 23:55:55 GMT. Указание параметра off позволяет ограничить время действия куки сессией браузера.

userid_flags

Синтаксис userid_flags off | *флаг* …;

Умолчание userid_flags off;

Контекст http, server, location

Если параметр не off, задаёт один или несколько дополнительных флагов для куки: *secure*, *httponly*, *samesite=strict*, *samesite=lax*, *samesite=none*.

userid_mark

Синтаксис userid_mark *буква* | *цифра* | = | off;

Умолчание userid_mark off;

Контекст http, server, location

Если параметр не off, включает механизм маркировки кук и задаёт символ, используемый в качестве метки. Этот механизм позволяет добавить или изменить userid_p3p и/или время хранения куки, но при этом оставить неизменным идентификатор клиента. Меткой может быть любая буква английского алфавита (с учётом регистра), цифра или знак “=”.

Если метка задана, то она сравнивается с первым дополняющим символом в base64 представлении идентификатора клиента, передаваемом в куке. Если они не совпадают, то кука перепосылается с заданной меткой, временем хранения и заголовком “P3P”.

userid_name

Синтаксис userid_name *имя*;

Умолчание userid_name uid;

Контекст http, server, location

Задаёт имя куки.

userid_p3p

Синтаксис userid_p3p *строка* | none;

Умолчание userid_p3p none;

Контекст http, server, location

Задаёт значение для поля заголовка “P3P”, которое будет выдаваться вместе с кукой. Если задано специальное значение none, то в ответе не будет заголовка “P3P”.

userid_path

Синтаксис userid_path *путь*;

Умолчание userid_path /;

Контекст http, server, location

Задаёт путь, для которого устанавливается кука.

userid_service

Синтаксис userid_service *номер*;

Умолчание userid_service IP-адрес сервера;

Контекст http, server, location

Если идентификаторы выдаются несколькими серверами (сервисами), то каждому сервису следует назначить свой собственный номер, для обеспечения уникальности выдаваемых идентификаторов клиентов. По умолчанию для кук первой версии используется ноль. Для кук второй версии по умолчанию используется число, составленное из последних четырёх октетов IP-адреса сервера.

Встроенные переменные

$uid_got

Имя куки и полученный идентификатор клиента.

$uid_reset

Если значением является непустая строка не равная 0, то клиентские идентификаторы перевыдаются. Специальное значение log дополнительно приводит к выдаче сообщений о перевыданных идентификаторах в error_log.

$uid_set

Имя куки и выданный идентификатор клиента.

Модуль http_uwsgi

Позволяет передавать запросы uwsgi-серверу.

Пример конфигурации

```
location / {
    include    uwsgi_params;
    uwsgi_pass localhost:9000;
}
```

Директивы

uwsgi_bind

Синтаксис uwsgi_bind *адрес* [transparent] | off;

Умолчание —

Контекст http, server, location

Задаёт локальный IP-адрес с необязательным портом, который будет использоваться в исходящих соединениях с uwsgi-сервером. В значении параметра допустимо использование переменных. Специальное значение off отменяет действие унаследованной с предыдущего уровня конфигурации директивы *uwsgi_bind*, позволяя системе самостоятельно выбирать локальный IP-адрес и порт.

Параметр transparent позволяет задать нелокальный IP-адрес, который будет использоваться в исходящих соединениях с uwsgi-сервером, например, реальный IP-адрес клиента:

```
uwsgi_bind $remote_addr transparent;
```

Для работы параметра обычно требуется запустить рабочие процессы Angie с привилегиями суперпользователя. В Linux это не требуется, так как если указан параметр *transparent*, то рабочие процессы наследуют *capability CAP_NET_RAW* из главного процесса.

Внимание

Необходимо настроить таблицу маршрутизации ядра для перехвата сетевого трафика с uwsgi-сервера.

uwsgi_buffer_size

Синтаксис uwsgi_buffer_size *размер*;

Умолчание uwsgi_buffer_size 4k|8k;

Контекст http, server, location

Задаёт размер буфера, в который будет читаться первая часть ответа, получаемого от uwsgi-сервера. В этой части ответа находится, как правило, небольшой заголовок ответа. По умолчанию размер одного буфера равен размеру страницы памяти. В зависимости от платформы это или 4K, или 8K, однако его можно сделать меньше.

uwsgi_buffering

Синтаксис uwsgi_buffering *размер*;

Умолчание uwsgi_buffering on;

Контекст http, server, location

on	Angie принимает ответ uwsgi-сервера как можно быстрее, сохраняя его в буферы, заданные директивами uwsgi_buffer_size и uwsgi_buffers. Если ответ не вмещается целиком в память, то его часть может быть записана на диск во временный файл. Запись во временные файлы контролируется директивами uwsgi_max_temp_file_size и uwsgi_temp_file_write_size.
off	Ответ синхронно передаётся клиенту сразу же по мере его поступления. Angie не пытается считать весь ответ uwsgi-сервера. Максимальный размер данных, который Angie может принять от сервера за один раз, задаётся директивой uwsgi_buffer_size.

Буферизация может быть также включена или выключена путём передачи значения “yes” или “no” в поле “X-Accel-Buffering” заголовка ответа. Эту возможность можно запретить с помощью директивы uwsgi_ignore_headers.

uwsgi_buffers

Синтаксис uwsgi_buffers *число размер*;

Умолчание uwsgi_buffers 8 4k|8k;

Контекст http, server, location

Задаёт число и размер буферов для одного соединения, в которые будет читаться ответ, получаемый от uwsgi-сервера. По умолчанию размер одного буфера равен размеру страницы. В зависимости от платформы это или 4K, или 8K.

uwsgi_busy_buffers_size

Синтаксис uwsgi_busy_buffers_size *размер*;

Умолчание uwsgi_buffers 8k|16k;

Контекст http, server, location

При включённой буферизации ответов uwsgi-сервера, ограничивает суммарный размер буферов, которые могут быть заняты для отправки ответа клиенту, пока ответ ещё не прочитан целиком. Оставшиеся буферы тем временем могут использоваться для чтения ответа и, при необходимости, буферизации части ответа во временный файл. По умолчанию размер ограничен величиной двух буферов, заданных директивами uwsgi_buffer_size и uwsgi_buffers.

uwsgi_cache

Синтаксис uwsgi_cache *зона* | off;

Умолчание uwsgi_cache off;

Контекст http, server, location

Задаёт зону разделяемой памяти, используемой для кэширования. Одна и та же зона может использоваться в нескольких местах. В значении параметра можно использовать переменные.

off	запрещает кэширование, унаследованное с предыдущего уровня конфигурации.

uwsgi_cache_background_update

Синтаксис uwsgi_cache_background_update on | off;

Умолчание uwsgi_cache_background_update off;

Контекст http, server, location

Позволяет запустить фоновый подзапрос для обновления просроченного элемента кэша, в то время как клиенту возвращается устаревший закэшированный ответ.

Внимание

Использование устаревшего закэшированного ответа в момент его обновления должно быть разрешено.

uwsgi_cache_bypass

Синтаксис uwsgi_cache_bypass …;

Умолчание —

Контекст http, server, location

Задаёт условия, при которых ответ не будет браться из кэша. Если значение хотя бы одного из строковых параметров непустое и не равно “0”, то ответ не берётся из кэша:

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

Можно использовать совместно с директивой uwsgi_no_cache.

uwsgi_cache_key

Синтаксис uwsgi_cache_key *строка*;

Умолчание —

Контекст http, server, location

Задаёт ключ для кэширования, например,

```
uwsgi_cache_key localhost:9000$request_uri;
```

uwsgi_cache_lock

Синтаксис uwsgi_cache_lock on | off;

Умолчание uwsgi_cache_lock off;

Контекст http, server, location

Если включено, одновременно только одному запросу будет позволено заполнить новый элемент кэша, идентифицируемый согласно директиве uwsgi_cache_key, передав запрос на uwsgi-сервер. Остальные запросы этого же элемента будут либо ожидать появления ответа в кэше, либо освобождения блокировки этого элемента, в течение времени, заданного директивой uwsgi_cache_lock_timeout.

uwsgi_cache_lock_age

Синтаксис uwsgi_cache_lock_age *время*;

Умолчание uwsgi_cache_lock_age 5s;

Контекст http, server, location

Если последний запрос, переданный на uwsgi-сервер для заполнения нового элемента кэша, не завершился за указанное время, на uwsgi-сервер может быть передан ещё один запрос.

uwsgi_cache_lock_timeout

Синтаксис uwsgi_cache_lock_timeout *время*;

Умолчание uwsgi_cache_lock_timeout 5s;

Контекст http, server, location

Задаёт таймаут для uwsgi_cache_lock. По истечении указанного времени запрос будет передан на uwsgi-сервер, однако ответ не будет закэширован.

uwsgi_cache_max_range_offset

Синтаксис uwsgi_cache_max_range_offset *число*;

Умолчание —

Контекст http, server, location

Задаёт смещение в байтах для запросов с указанием диапазона запрашиваемых байт (byte-range requests). Если диапазон находится за указанным смещением, range-запрос будет передан на uwsgi-сервер и ответ не будет закэширован.

uwsgi_cache_methods

Синтаксис uwsgi_cache_methods GET | HEAD | POST …;

Умолчание uwsgi_cache_methods GET HEAD;

Контекст http, server, location

Если метод запроса клиента указан в этой директиве, то ответ будет закэширован. Методы “GET” и “HEAD” всегда добавляются в список, но тем не менее рекомендуется перечислять их явно. См. также директиву uwsgi_no_cache.

uwsgi_cache_min_uses

Синтаксис uwsgi_cache_min_uses *число*;

Умолчание uwsgi_cache_min_uses 1;

Контекст http, server, location

Задаёт число запросов, после которого ответ будет закэширован.

uwsgi_cache_path

Синтаксис uwsgi_cache_path *путь* [levels = *уровни*] [use_temp_path=on|off] keys_zone = *имя:размер* [inactive = *время*] [max_size = *размер*] [min_free = *размер*] [manager_files = *число*] [manager_sleep = *время*] [manager_threshold = *время*] [loader_files = *число*] [loader_sleep = *время*] [loader_threshold = *время*];

Умолчание —

Контекст http

Задаёт путь и другие параметры кэша. Данные кэша хранятся в файлах. Именем файла в кэше является результат функции MD5 от ключа кэширования.

Параметр levels задаёт уровни иерархии кэша: можно задать от 1 до 3 уровней, на каждом уровне допускаются значения 1 или 2.

Например, при использовании

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

имена файлов в кэше будут такого вида:

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

Кэшируемый ответ сначала записывается во временный файл, а потом этот файл переименовывается. Временные файлы и кэш могут располагаться на разных файловых системах. Однако нужно учитывать, что в этом случае вместо дешёвой операции переименовывания в пределах одной файловой системы файл копируется с одной файловой системы на другую. Поэтому лучше, если кэш будет находиться на той же файловой системе, что и каталог с временными файлами.

Какой из каталогов будет использоваться для временных файлов, определяется параметром use_temp_path.

on	Если параметр не задан или установлен в значение “on”, будет использоваться каталог, задаваемый директивой uwsgi_temp_path для данного *location*
off	временные файлы будут располагаться непосредственно в каталоге кэша

Кроме того, все активные ключи и информация о данных хранятся в зоне разделяемой памяти, имя и размер которой задаются параметром keys_zone. Зоны размером в 1 мегабайт достаточно для хранения около 8 тысяч ключей.

Если к данным кэша не обращаются в течение времени, заданного параметром inactive, то данные удаляются, независимо от их свежести.
По умолчанию *inactive* равен 10 минутам.

Специальный процесс **cache manager** следит за максимальным размером кэша, а также за минимальным объёмом свободного места на файловой системе с кэшем, и удаляет наименее востребованные данные при превышении максимального размера кэша или недостаточном объёме свободного места. Удаление данных происходит итерациями.

max_size	максимальное пороговое значение размера кэша
min_free	минимальное пороговое значение объёма свободного места на файловой системе с кэшем
manager_files	максимальное количество удаляемых элементов кэша за одну итерацию По умолчанию 100
manager_threshold	ограничивает время работы одной итерации По умолчанию 200 миллисекунд
manager_sleep	время, в течение которого выдерживается пауза между итерациями По умолчанию 50 миллисекунд

Через минуту после старта Angie активируется специальный процесс **cache loader**, который загружает в зону кэша информацию о ранее закэшированных данных, хранящихся на файловой системе. Загрузка также происходит итерациями.

loader_files	максимальное количество элементов кэша к загрузке в одну итерацию По умолчанию 100
loader_threshold	ограничивает время работы одной итерации

	По умолчанию 200 миллисекунд
loader_sleep	время, в течение которого выдерживается пауза между итерациями По умолчанию 50 миллисекунд

uwsgi_cache_revalidate

Синтаксис uwsgi_cache_revalidate on | off;

Умолчание uwsgi_cache_revalidate off;

Контекст http, server, location

Разрешает ревалидацию просроченных элементов кэша при помощи условных запросов с полями заголовка “If-Modified-Since” и “If-None-Match”.

uwsgi_cache_use_stale

Синтаксис uwsgi_cache_use_stale error | timeout | invalid_header | updating | http_500 | http_503 | http_403 | http_404 | http_429 | off …;

Умолчание uwsgi_cache_use_stale off;

Контекст http, server, location

Определяет, в каких случаях можно использовать устаревший закэшированный ответ. Параметры директивы совпадают с параметрами директивы uwsgi_next_upstream.

error	Позволяет использовать устаревший закэшированный ответ при невозможности выбора uwsgi-сервера для обработки запроса.
updating	Дополнительный параметр, разрешает использовать устаревший закэшированный ответ, если на данный момент он уже обновляется. Это позволяет минимизировать число обращений к uwsgi-серверам при обновлении закэшированных данных.

Использование устаревшего закэшированного ответа может также быть разрешено непосредственно в заголовке ответа на определённое количество секунд после того, как ответ устарел:

- Расширение stale-while-revalidate поля заголовка “Cache-Control” разрешает использовать устаревший закэшированный ответ, если на данный момент он уже обновляется.
- Расширение stale-if-error поля заголовка “Cache-Control” разрешает использовать устаревший закэшированный ответ в случае ошибки.

Примечание

Такой способ менее приоритетен, чем задание параметров директивы.

Чтобы минимизировать число обращений к uwsgi-серверам при заполнении нового элемента кэша, можно воспользоваться директивой uwsgi_cache_lock.

uwsgi_cache_valid

Синтаксис uwsgi_cache_valid [*код* …] *время*;

Умолчание —

Контекст http, server, location

Задаёт время кэширования для разных кодов ответа. Например, директивы

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

задают время кэширования 10 минут для ответов с кодами 200 и 302 и 1 минуту для ответов с кодом 404.

Если указано только время кэширования,

```
uwsgi_cache_valid 5m;
```

то кэшируются только ответы 200, 301 и 302.

Кроме того, можно кэшировать любые ответы с помощью параметра any:

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

Примечание

Параметры кэширования могут также быть заданы непосредственно в заголовке ответа. Такой способ приоритетнее, чем задание времени кэширования с помощью директивы.

- Поле заголовка “X-Accel-Expires” задаёт время кэширования ответа в секундах. Значение *0* запрещает кэшировать ответ. Если значение начинается с префикса @, оно задаёт абсолютное время в секундах с начала эпохи, до которого ответ может быть закэширован.
- Если в заголовке нет поля “X-Accel-Expires”, параметры кэширования определяются по полям заголовка “Expires” или “Cache-Control”.
- Ответ, в заголовке которого есть поле “Set-Cookie”, не будет кэшироваться.
- Ответ, в заголовке которого есть поле “Vary” со специальным значением “*”, не будет кэшироваться. Ответ, в заголовке которого есть поле “Vary” с другим значением, будет закэширован с учётом соответствующих полей заголовка запроса.

Обработка одного или более из этих полей заголовка может быть отключена при помощи директивы uwsgi_ignore_headers.

uwsgi_connect_timeout

Синтаксис uwsgi_connect_timeout *время*;

Умолчание uwsgi_connect_timeout 60s;

Контекст http, server, location

Задаёт таймаут для установления соединения с uwsgi-сервером. Необходимо иметь в виду, что этот таймаут обычно не может превышать 75 секунд.

uwsgi_force_ranges

Синтаксис uwsgi_force_ranges off;

Умолчание uwsgi_force_ranges off;

Контекст http, server, location

Включает поддержку диапазонов запрашиваемых байт (byte-range) для кэшированных и некэшированных ответов uwsgi-сервера вне зависимости от наличия поля “Accept-Ranges” в заголовках этих ответов.

uwsgi_hide_header

Синтаксис uwsgi_hide_header *поле*;

Умолчание —

Контекст http, server, location

По умолчанию Angie не передаёт клиенту поля заголовка “Date”, “Server”, “X-Pad” и “X-Accel-…” из ответа uwsgi-сервера. Директива *uwsgi_hide_header* задаёт дополнительные поля, которые не будут передаваться. Если же передачу полей нужно разрешить, можно воспользоваться директивой uwsgi_pass_header.

uwsgi_ignore_client_abort

Синтаксис uwsgi_ignore_client_abort on | off;

Умолчание uwsgi_ignore_client_abort off;

Контекст http, server, location

Определяет, закрывать ли соединение с uwsgi-сервером в случае, если клиент закрыл соединение, не дождавшись ответа.

uwsgi_ignore_headers

Синтаксис uwsgi_ignore_headers *поле* …;

Умолчание —

Контекст http, server, location

Запрещает обработку некоторых полей заголовка из ответа uwsgi-сервера. В директиве можно указать поля “X-Accel-Redirect”, “X-Accel-Expires”, “X-Accel-Limit-Rate”, “X-Accel-Buffering”, “X-Accel-Charset”, “Expires”, “Cache-Control”, “Set-Cookie” и “Vary”.

Если не запрещено, обработка этих полей заголовка заключается в следующем:

- “X-Accel-Expires”, “Expires”, “Cache-Control”, “Set-Cookie” и “Vary” задают параметры кэширования ответа;
- “X-Accel-Redirect” производит внутреннее перенаправление на указанный URI;
- “X-Accel-Limit-Rate” задаёт ограничение скорости передачи ответа клиенту;

- “X-Accel-Buffering” включает или выключает буферизацию ответа;
- “X-Accel-Charset” задаёт желаемую кодировку ответа.

uwsgi_intercept_errors

Синтаксис uwsgi_intercept_errors on | off;

Умолчание uwsgi_intercept_errors off;

Контекст http, server, location

Определяет, передавать ли клиенту ответы uwsgi-сервера с кодом больше либо равным 300, или же перехватывать их и перенаправлять на обработку Angie с помощью директивы error_page.

uwsgi_limit_rate

Синтаксис uwsgi_limit_rate *скорость*;

Умолчание uwsgi_limit_rate 0;

Контекст http, server, location

Ограничивает скорость чтения ответа от uwsgi-сервера. *Скорость* задаётся в байтах в секунду.

0	отключает ограничение скорости

Примечание

Ограничение устанавливается на запрос, поэтому, если Angie одновременно откроет два соединения к uwsgi-серверу, суммарная скорость будет вдвое выше заданного ограничения. Ограничение работает только в случае, если включена буферизация ответов uwsgi-сервера.

uwsgi_max_temp_file_size

Синтаксис uwsgi_max_temp_file_size *размер*;

Умолчание uwsgi_max_temp_file_size 1024m;

Контекст http, server, location

Если включена буферизация ответов uwsgi-сервера, и ответ не вмещается целиком в буферы, заданные директивами uwsgi_buffer_size и uwsgi_buffers, часть ответа может быть записана во временный файл. Эта директива задаёт максимальный размер временного файла. Размер данных, сбрасываемых во временный файл за один раз, задаётся директивой uwsgi_temp_file_write_size.

0	отключает возможность буферизации ответов во временные файлы

Примечание

Данное ограничение не распространяется на ответы, которые будут закэшированы или сохранены на диске.

uwsgi_modifier1

Синтаксис uwsgi_modifier1 *число*;

Умолчание uwsgi_modifier1 0;

Контекст http, server, location

Задаёт значение поля modifier1 в заголовке пакета uwsgi.

uwsgi_modifier2

Синтаксис uwsgi_modifier2 *число*;

Умолчание uwsgi_modifier1 0;

Контекст http, server, location

Задаёт значение поля modifier2 в заголовке пакета uwsgi.

uwsgi_next_upstream

Синтаксис uwsgi_next_upstream error | timeout | invalid_header | http_500 | http_503 | http_403 | http_404 | http_429 | non_idempotent | off …;

Умолчание uwsgi_next_upstream error timeout;

Контекст http, server, location

Определяет, в каких случаях запрос будет передан следующему в группе upstream серверу:

error	произошла ошибка соединения с сервером, передачи ему запроса или чтения заголовка ответа сервера;
timeout	произошёл таймаут во время соединения с сервером, передачи ему запроса или чтения заголовка ответа сервера;
invalid_header	сервер вернул пустой или неверный ответ;
http_500	сервер вернул ответ с кодом 500;
http_503	сервер вернул ответ с кодом 503;
http_403	сервер вернул ответ с кодом 403;
http_404	сервер вернул ответ с кодом 404;
http_429	сервер вернул ответ с кодом 429;
non_idempotent	обычно запросы с неидемпотентным методом (POST, LOCK, PATCH) не передаются на другой сервер, если запрос серверу группы уже был отправлен; включение параметра явно разрешает повторять подобные запросы;
off	запрещает передачу запроса следующему серверу.

Примечание

Необходимо понимать, что передача запроса следующему серверу возможна только при условии, что клиенту ещё ничего не передавалось. То есть, если ошибка или таймаут возникли в середине передачи ответа клиенту, то действие директивы на такой запрос не распространяется.

Директива также определяет, что считается неудачной попыткой работы с сервером.

error timeout invalid_header	всегда считаются неудачными попытками, даже если они не указаны в директиве
http_500 http_503 http_429	считаются неудачными попытками, только если они указаны в директиве
http_403 http_404	никогда не считаются неудачными попытками

Передача запроса следующему серверу может быть ограничена по количеству попыток и по времени.

uwsgi_next_upstream_timeout

Синтаксис uwsgi_next_upstream_timeout *время*;

Умолчание uwsgi_next_upstream_timeout 0;

Контекст http, server, location

Ограничивает время, в течение которого возможна передача запроса следующему серверу.

0	отключает это ограничение

uwsgi_next_upstream_tries

Синтаксис uwsgi_next_upstream_tries *число*;

Умолчание uwsgi_next_upstream_tries 0;

Контекст http, server, location

Ограничивает число допустимых попыток для передачи запроса следующему серверу.

0	отключает это ограничение

uwsgi_no_cache

Синтаксис uwsgi_no_cache *строка* …;

Умолчание —

Контекст http, server, location

Задаёт условия, при которых ответ не будет сохраняться в кэш. Если значение хотя бы одного из строковых параметров непустое и не равно “0”, то ответ не будет сохранён:

```
uwsgi_no_cache $cookie_nocache $arg_nocache$arg_comment;
```

```
uwsgi_no_cache $http_pragma    $http_authorization;
```

Можно использовать совместно с директивой uwsgi_cache_bypass.

uwsgi_pass

Синтаксис uwsgi_pass [*протокол*://] *адрес*;

Умолчание —

Контекст location, if в location

Задаёт протокол и адрес uwsgi-сервера. В качестве протокола можно указать “*uwsgi*” или “*suwsgi*” (secured uwsgi, uwsgi через SSL). Адрес может быть указан в виде доменного имени или IP-адреса, и порта:

```
uwsgi_pass localhost:9000;
uwsgi_pass uwsgi://localhost:9000;
uwsgi_pass suwsgi://[2001:db8::1]:9090;
```

или в виде пути UNIX-сокета, который указывается после слова “*unix*” и заключается в двоеточия:

```
uwsgi_pass unix:/tmp/uwsgi.socket;
```

Если доменному имени соответствует несколько адресов, то все они будут использоваться по очереди (round-robin). Кроме того, в качестве адреса можно указать группу серверов.

В значении параметра можно использовать переменные. В этом случае, если адрес указан в виде доменного имени, имя ищется среди описанных групп серверов и если не найдено, то определяется с помощью resolver’а.

uwsgi_pass_header

Синтаксис uwsgi_pass_header *поле* …;

Умолчание —

Контекст http, server, location

Разрешает передавать от uwsgi-сервера клиенту запрещённые для передачи поля заголовка.

uwsgi_pass_request_body

Синтаксис uwsgi_pass_request_body on | off;

Умолчание uwsgi_pass_request_body on;

Контекст http, server, location

Позволяет запретить передачу исходного тела запроса на uwsgi-сервер. См. также директиву uwsgi_pass_request_headers.

uwsgi_pass_request_headers

Синтаксис uwsgi_pass_request_headers on | off;

Умолчание uwsgi_pass_request_headers on;

Контекст http, server, location

Позволяет запретить передачу полей заголовка исходного запроса на uwsgi-сервер. См. также директиву uwsgi_pass_request_body.

uwsgi_read_timeout

Синтаксис uwsgi_read_timeout *время*;

Умолчание uwsgi_read_timeout 60s;

Контекст http, server, location

Задаёт таймаут при чтении ответа uwsgi-сервера. Таймаут устанавливается не на всю передачу ответа, а только между двумя операциями чтения. Если по истечении этого времени uwsgi-сервер ничего не передаст, соединение закрывается.

uwsgi_request_buffering

Синтаксис uwsgi_request_buffering on | off;

Умолчание uwsgi_request_buffering on;

Контекст http, server, location

Разрешает или запрещает использовать буферизацию тела запроса клиента.

on	тело запроса полностью читается от клиента перед отправкой запроса на uwsgi-сервер.
off	тело запроса отправляется на uwsgi-сервер сразу же по мере его поступления. В этом случае запрос не может быть передан следующему серверу, если Angie уже начал отправку тела запроса.

Если для отправки тела исходного запроса используется HTTP/1.1 и передача данных частями (chunked transfer encoding), то тело запроса буферизуется независимо от значения директивы.

uwsgi_send_timeout

Синтаксис uwsgi_send_timeout *время*;

Умолчание uwsgi_send_timeout 60s;

Контекст http, server, location

Задаёт таймаут при передаче запроса uwsgi-серверу. Таймаут устанавливается не на всю передачу запроса, а только между двумя операциями записи. Если по истечении этого времени uwsgi-сервер не примет новых данных, соединение закрывается.

uwsgi_socket_keepalive

Синтаксис uwsgi_socket_keepalive on | off;

Умолчание uwsgi_socket_keepalive off;

Контекст http, server, location

Конфигурирует поведение “TCP keepalive” для исходящих соединений к uwsgi-серверу.

""	По умолчанию для сокета действуют настройки операционной системы.
on	для сокета включается параметр SO_KEEPALIVE

uwsgi_ssl_certificate

Синтаксис uwsgi_ssl_certificate *файл*;

Умолчание —

Контекст http, server, location

Задаёт файл с сертификатом в формате PEM для аутентификации на suwsgi-сервере. В имени файла можно использовать переменные.

uwsgi_ssl_certificate_key

Синтаксис uwsgi_ssl_certificate_key *файл*;

Умолчание —

Контекст http, server, location

Задаёт файл с секретным ключом в формате PEM для аутентификации на suwsgi-сервере.

Вместо файла можно указать значение “engine:*имя*:id“, которое загружает ключ с указанным *id* из OpenSSL engine с заданным именем.

В имени файла можно использовать переменные.

uwsgi_ssl_ciphers

Синтаксис uwsgi_ssl_ciphers *шифры*;

Умолчание uwsgi_ssl_ciphers DEFAULT;

Контекст http, server, location

Описывает разрешённые шифры для запросов к suwsgi-серверу. Шифры задаются в формате, поддерживаемом библиотекой OpenSSL.

Полный список можно посмотреть с помощью команды “*openssl ciphers*”.

uwsgi_ssl_conf_command

Синтаксис uwsgi_ssl_conf_command *имя значение*;

Умолчание —

Контекст http, server, location

Задаёт произвольные конфигурационные команды OpenSSL при установлении соединения с uwsgi HTTPS-сервером.

Примечание

Директива поддерживается при использовании OpenSSL 1.0.2 и выше.

На одном уровне может быть указано несколько директив *uwsgi_ssl_conf_command*. Директивы наследуются с предыдущего уровня конфигурации при условии, что на данном уровне не описаны свои директивы *uwsgi_ssl_conf_command*.

Внимание

Следует учитывать, что изменение настроек OpenSSL напрямую может привести к неожиданному поведению.

uwsgi_ssl_crl

Синтаксис uwsgi_ssl_crl *файл*;

Умолчание —

Контекст http, server, location

Указывает файл с отозванными сертификатами (CRL) в формате PEM, используемыми при проверке сертификата suwsgi-сервера.

uwsgi_ssl_name

Синтаксис uwsgi_ssl_name *имя*;

Умолчание uwsgi_ssl_name `имя хоста из uwsgi_pass;`

Контекст http, server, location

Позволяет переопределить имя сервера, используемое при проверке сертификата uwsgi HTTPS-сервера, а также для передачи его через SNI при установлении соединения с suwsgi-сервером.

По умолчанию используется имя хоста из URL'а, заданного директивой uwsgi_pass.

uwsgi_ssl_password_file

Синтаксис uwsgi_ssl_password_file *файл*;

Умолчание —

Контекст http, server, location

Задаёт файл с паролями от секретных ключей, где каждый пароль указан на отдельной строке. Пароли применяются по очереди в момент загрузки ключа.

uwsgi_ssl_protocols

Синтаксис uwsgi_ssl_protocols [SSLv2] [SSLv3] [TLSv1] [TLSv1.1] [TLSv1.2] [TLSv1.3];

Умолчание uwsgi_ssl_protocols TLSv1 TLSv1.1 TLSv1.2;

Контекст http, server, location

Разрешает указанные протоколы для запросов к suwsgi-серверу.

uwsgi_ssl_server_name

Синтаксис uwsgi_ssl_server_name on | off;

Умолчание uwsgi_ssl_server_name off;

Контекст http, server, location

Разрешает или запрещает передачу имени сервера через расширение Server Name Indication протокола TLS (SNI, RFC 6066) при установлении соединения с suwsgi-сервером.

uwsgi_ssl_session_reuse

Синтаксис uwsgi_ssl_session_reuse on | off;

Умолчание uwsgi_ssl_session_reuse on;

Контекст http, server, location

Определяет, использовать ли повторно SSL-сессии при работе с suwsgi-сервером. Если в логах появляются ошибки “*SSL3_GET_FINISHED:digest check failed*”, то можно попробовать выключить повторное использование сессий.

uwsgi_ssl_trusted_certificate

Синтаксис uwsgi_ssl_trusted_certificate *файл*;

Умолчание —

Контекст http, server, location

Задаёт файл с доверенными сертификатами CA в формате PEM, используемыми при проверке сертификата uwsgi HTTPS-сервера.

uwsgi_ssl_verify

Синтаксис uwsgi_ssl_verify on | off;

Умолчание uwsgi_ssl_verify off;

Контекст http, server, location

Разрешает или запрещает проверку сертификата suwsgi-сервера.

uwsgi_ssl_verify_depth

Синтаксис uwsgi_ssl_verify_depth *число*;

Умолчание uwsgi_ssl_verify_depth 1;

Контекст http, server, location

Устанавливает глубину проверки в цепочке сертификатов suwsgi-сервера.

uwsgi_store

Синтаксис uwsgi_store on | off | *строка*;

Умолчание uwsgi_store off;

Контекст http, server, location

Разрешает сохранение на диск файлов.

on	сохраняет файлы в соответствии с путями, указанными в директивах alias или root
off	запрещает сохранение файлов

Имя файла можно задать явно с помощью строки с переменными:

```
uwsgi_store /data/www$original_uri;
```

Время изменения файлов выставляется согласно полученному полю “Last-Modified” в заголовке ответа. Ответ сначала записывается во временный файл, а потом этот файл переименовывается. Временный файл и постоянное место хранения ответа могут располагаться на разных файловых системах. Однако нужно учитывать, что в этом случае вместо дешёвой операции переименовывания в пределах одной файловой системы файл копируется с одной файловой системы на другую. Поэтому лучше, если сохраняемые файлы будут находиться на той же файловой системе, что и каталог с временными файлами, задаваемый директивой uwsgi_temp_path для данного *location*.

Директиву можно использовать для создания локальных копий статических неизменяемых файлов:

```
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

Синтаксис uwsgi_store_access *пользователи*:*права* …;

Умолчание uwsgi_store_access user:rw;

Контекст http, server, location

Задаёт права доступа для создаваемых файлов и каталогов, например,

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

Если заданы какие-либо права для *group* или *all*, то права для user указывать необязательно:

```
uwsgi_store_access group:rw all:r;
```

uwsgi_temp_file_write_size

Синтаксис uwsgi_temp_file_write_size *размер*;

Умолчание uwsgi_temp_file_write_size 8k|16k;

Контекст http, server, location

Ограничивает размер данных, сбрасываемых во временный файл за один раз, при включённой буферизации ответов uwsgi-сервера во временные файлы. По умолчанию размер ограничен двумя буферами, заданными директивами uwsgi_buffer_size и uwsgi_buffers. Максимальный размер временного файла задаётся директивой uwsgi_max_temp_file_size.

uwsgi_temp_path

Синтаксис uwsgi_temp_path *путь* [*уровень1* [*уровень2* [*уровень3*]]]`;

Умолчание uwsgi_temp_path uwsgi_temp;

Контекст http, server, location

Задаёт имя каталога для хранения временных файлов с данными, полученными от uwsgi-серверов. В каталоге может использоваться иерархия подкаталогов до трёх уровней. Например, при такой конфигурации

```
uwsgi_temp_path /spool/angie/uwsgi_temp 1 2;
```

временный файл будет следующего вида:

```
/spool/angie/uwsgi_temp/7/45/00000123457
```

См. также параметр *use_temp_path* директивы uwsgi_cache_path.

Модуль http_v2

Позволяет обеспечивает поддержку HTTP/2.

По умолчанию этот модуль не собирается, его сборку необходимо разрешить с помощью конфигурационного параметра *--with-http_v2_module*.

Пример конфигурации

```
server {
    listen 443 ssl http2;

    ssl_certificate server.crt;
    ssl_certificate_key server.key;
}
```

Примечание

Чтобы принимать HTTP/2-соединения по TLS, необходимо наличие поддержки расширения “Application-Layer Protocol Negotiation” (ALPN) протокола TLS, появившейся в OpenSSL версии 1.0.2.

Если директива ssl_prefer_server_ciphers установлена в значение “on”, шифры должны быть настроены таким образом, чтобы соответствовать чёрному списку RFC 7540, Appendix A а также поддерживаться клиентами.

Директивы

http2_body_preread_size

Синтаксис http2_body_preread_size *размер*;

Умолчание —

Контекст http, server

Задаёт размер буфера для каждого запроса, в который может сохраняться тело запроса до того, как оно начнёт обрабатываться.

http2_chunk_size

Синтаксис http2_chunk_size *размер*;

Умолчание http2_chunk_size 8k;

Контекст http, server, location

Задаёт максимальный размер частей, на которое будет разделяться тело ответа. Слишком маленькое значение может привести к росту накладных расходов. Слишком большое значение может негативно сказаться на приоритизации из-за блокировки очереди.

http2_max_concurrent_pushes

Синтаксис http2_max_concurrent_pushes *число*;

Умолчание http2_max_concurrent_pushes 10;

Контекст http, server

Ограничивает максимальное число параллельных push-запросов в соединении.

http2_max_concurrent_streams

Синтаксис http2_max_concurrent_streams *число*;

Умолчание http2_max_concurrent_streams 128;

Контекст http, server

Задаёт максимальное число параллельных HTTP/2-потоков в соединении.

http2_push

Синтаксис http2_push *uri* | off;

Умолчание http2_push off;

Контекст http, server, location

Заблаговременно отправляет (push) запрос к заданному uri вместе с ответом на оригинальный запрос. Будут обработаны только относительные URI с абсолютными путями, например:

```
http2_push /static/css/main.css;
```

В значении uri допустимо использование переменных.

На одном уровне конфигурации можно указать несколько *http2_push* директив. Параметр off отменяет действие унаследованных с предыдущего уровня конфигурации директив *http2_push*.

http2_push_preload

Синтаксис http2_push_preload on | off;

Умолчание http2_push_preload off;

Контекст http, server, location

Разрешает автоматическое преобразование preload links, указанных в полях “Link” заголовка ответа, в push-запросы.

http2_recv_buffer_size

Синтаксис http2_recv_buffer_size *размер*;

Умолчание http2_recv_buffer_size 256k;

Контекст http

Задаёт размер входного буфера для рабочего процесса.

Встроенные переменные

Модуль *http_v2* поддерживает следующие встроенные переменные:

$http2

согласованный идентификатор протокола:

h2	для HTTP/2 через TLS
h2c	для HTTP/2 через незашифрованный TCP
""	пустая строка для остальных случаев

Модуль http_xslt

Фильтр, преобразующий XML-ответ с помощью одного или нескольких XSLT-шаблонов.

По умолчанию этот модуль не собирается, его сборку необходимо разрешить с помощью конфигурационного параметра *--with-http_xslt_module*.

Примечание

Для сборки и работы этого модуля нужны библиотеки libxml2 и libxslt.

Пример конфигурации

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

Директивы

xml_entities

Синтаксис xml_entities *путь*;

Умолчание —

Контекст http, server, location

Задаёт файл DTD, в котором описаны символьные сущности. Этот файл компилируется на стадии конфигурации. По техническим причинам модуль не имеет возможности использовать внешнее подмножество, заданное в обрабатываемом XML, поэтому оно игнорируется, а вместо него используется специально заданный файл. В этом файле не нужно описывать структуру XML, достаточно только объявления необходимых символьных сущностей, например:

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

xslt_last_modified

Синтаксис xslt_last_modified on | off;

Умолчание xslt_last_modified off;

Контекст http, server, location

Позволяет сохранить поле заголовка “Last-Modified” исходного ответа во время XSLT-преобразований для лучшего кэширования ответов.

По умолчанию поле заголовка удаляется, так как содержимое ответа изменяется во время преобразования и может содержать динамически созданные элементы или части, которые изменились независимо от исходного ответа.

xslt_param

Синтаксис xslt_param *параметр значение*;

Умолчание —

Контекст http, server, location

Задаёт параметры для XSLT-шаблонов. Значение рассматривается как выражение XPath. В значении можно использовать переменные. Если нужно передать в шаблон строковое значение, можно воспользоваться директивой xslt_string_param.

Директив *xslt_param* может быть несколько. Директивы наследуются с предыдущего уровня конфигурации при условии, что на данном уровне не описаны свои директивы *xslt_param* и xslt_string_param.

xslt_string_param

Синтаксис xslt_string_param *параметр значение*;

Умолчание —

Контекст http, server, location

Задаёт строковые параметры для XSLT-шаблонов. Выражения XPath в значении параметра не интерпретируются. В значении можно использовать переменные.

Директив *xslt_string_param* может быть несколько. Директивы наследуются с предыдущего уровня конфигурации при условии, что на данном уровне не описаны свои директивы xslt_param и *xslt_string_param*.

xslt_stylesheet

Синтаксис xslt_stylesheet *шаблон* [*параметр* = *значение* …];

Умолчание —

Контекст location

Задаёт XSLT-шаблон и необязательные параметры для этого шаблона. Шаблон компилируется на стадии конфигурации.

Параметры можно задавать как по отдельности, так и группировать в одной строке, разделяя символом “:”. Если же в самих параметрах встречается символ “:”, то его нужно экранировать в виде “*%3A*”. Кроме того, libxslt требует, чтобы параметры, содержащие не только алфавитно-цифровые символы, были заключены в одинарные или двойные кавычки, например:

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

В описании параметров можно использовать переменные, например, целая строка параметров может быть взята из одной переменной:

```
location / {
    xslt_stylesheet /site/xslt/one.xslt
                    $arg_xslt_params
                    param1='$value1':param2=value2
                    param3=value3;
}
```

Можно указать несколько шаблонов — в этом случае они будут применяться последовательно в порядке их описания.

xslt_types

Синтаксис xslt_types mime-*тип* …;

Умолчание xslt_types text/xml;

Контекст http, server, location

Разрешает преобразования в ответах с указанными MIME-типами в дополнение к “*text/xml*”. Специальное значение * соответствует любому MIME-типу. Если в результате преобразования выдаётся HTML-ответ, то его MIME-тип меняется на “*text/html*”.

5.4 Модули stream

Модуль stream_access

Модуль позволяет ограничить доступ для определённых адресов клиентов.

Пример конфигурации

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

Правила проверяются в порядке их записи до первого соответствия. В данном примере доступ разрешён только для IPv4-сетей 10.1.1.0/16 и 192.168.1.0/24, кроме адреса 192.168.1.1, и для IPv6-сети 2001:0db8::/32.

Директивы

allow

Синтаксис allow *адрес* | *CIDR* | unix: | all;

Умолчание —

Контекст stream, server

Разрешает доступ для указанной сети или адреса. Если указано специальное значение *unix:*, разрешает доступ для всех UNIX-сокетов.

deny

Синтаксис deny *адрес* | *CIDR* | unix: | all;

Умолчание —

Контекст stream, server

Запрещает доступ для указанной сети или адреса. Если указано специальное значение *unix:*, запрещает доступ для всех UNIX-сокетов.

Модуль stream_core

Модуль по умолчанию не собирается, его сборку необходимо разрешить с помощью конфигурационного параметра *–with-stream*.

Пример конфигурации

```
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;
    }
}
```

Директивы

listen

Синтаксис listen *адрес*[:*порт*] [ssl] [udp] [proxy_protocol] [fastopen = *число*] [backlog = *число*] [rcvbuf = *размер*] [sndbuf = *размер*] [bind] [ipv6only=on|off] [reuseport] [so_keepalive=on|off|[keepidle]:[keepintvl]:[keepcnt]];

Умолчание —

Контекст server

Задаёт *адрес* и *порт* для сокета, на котором сервер будет принимать соединения. Можно указать только *порт*. Кроме того, *адрес* может быть именем хоста, например:

```
listen 127.0.0.1:12345;
listen *:12345;
listen 12345;     # то же, что и *:12345
listen localhost:12345;
```

IPv6-адреса задаются в квадратных скобках:

```
listen [::1]:12345;
listen [::]:12345;
```

UNIX-сокеты задаются префиксом *unix:*

```
listen unix:/var/run/angie.sock;
```

Диапазоны портов задаются при помощи указания первого и последнего порта через дефис:

```
listen 127.0.0.1:12345-12399;
listen 12345-12399;
```

Внимание

Разные серверы должны слушать на разных парах *адрес:порт*.

ssl	указывает на то, что все соединения, принимаемые на данном слушающем сокете, должны работать в режиме SSL.
udp	конфигурирует слушающий сокет для работы с датаграммами. Для обработки пакетов с одного адреса и порта в рамках одной сессии необходимо также указывать параметр *reuseport*.
proxy_protocol	указывает на то, что все соединения, принимаемые на данном порту, должны использовать протокол PROXY.

В директиве *listen* можно также указать несколько дополнительных параметров, специфичных для связанных с сокетами системных вызовов.

fastopen= *число*	включает “TCP Fast Open” для слушающего сокета и ограничивает максимальную длину очереди соединений, которые ещё не завершили процесс three-way handshake.

Примечание

Не включайте “TCP Fast Open”, не убедившись, что сервер может адекватно обрабатывать многократное получение одного и того же SYN-пакета с данными.

backlog= *число*	задаёт параметр *backlog* в вызове *listen()*, который ограничивает

	максимальный размер очереди ожидающих приёма соединений. По умолчанию *backlog* устанавливается равным *-1* для FreeBSD, DragonFly BSD и macOS, и *511* для других платформ.
rcvbuf= *размер*	задаёт размер буфера приёма (параметр *SO_RCVBUF*) для слушающего сокета.
sndbuf= *размер*	задаёт размер буфера передачи (параметр *SO_SNDBUF*) для слушающего сокета.
bind	указывает, что для данного слушающего сокета нужно делать *bind()* отдельно. Это нужно потому, что если описаны несколько директив *listen* с одинаковым портом, но разными адресами, и одна из директив *listen* слушает на всех адресах для данного *порта* (**:порт*), то Angie сделает *bind()* только на **:порт*. Необходимо заметить, что в этом случае для определения адреса, на который пришло соединение, делается системный вызов *getsockname()*. Если же используются параметры *backlog*, *rcvbuf*, *sndbuf*, *ipv6only*, *reuseport* или *so_keepalive*, то для данной пары *адрес:порт* всегда делается отдельный вызов *bind()*.
ipv6only=on | off	определяет (через параметр сокета *IPV6_V6ONLY*), будет ли слушающий на wildcard-адресе [::] IPv6-сокет принимать только IPv6-соединения, или же одновременно IPv6- и IPv4-соединения. По умолчанию параметр включён. Установить его можно только один раз на старте.
reuseport	указывает, что нужно создавать отдельный слушающий сокет для каждого рабочего процесса (через параметр сокета *SO_REUSEPORT* для Linux 3.9+ и DragonFly BSD или *SO_REUSEPORT_LB* для FreeBSD 12+), позволяя ядру распределять входящие соединения между рабочими процессами. В настоящий момент это работает только на Linux 3.9+, DragonFly BSD и FreeBSD 12+.

Предупреждение

Ненадлежащее использование параметра *reuseport* может быть небезопасно.

so_keepalive=on | off | [keepidle]:[keepintvl]:[keepcnt] конфигурирует для слушающего сокета поведение “TCP keepalive”.

"	если параметр опущен, для сокета будут действовать настройки операционной системы
on	для сокета включается параметр *SO_KEEPALIVE*
off	для сокета параметр *SO_KEEPALIVE* выключается

Некоторые операционные системы поддерживают настройку параметров “TCP keepalive” на уровне сокета посредством параметров *TCP_KEEPIDLE*, *TCP_KEEPINTVL* и

TCP_KEEPCNT. На таких системах их можно сконфигурировать с помощью параметров *keepidle*, *keepintvl* и *keepcnt*. Один или два параметра могут быть опущены, в таком случае для соответствующего параметра сокета будут действовать стандартные системные настройки.

Например,

```
so_keepalive=30m::10
```

установит таймаут бездействия (*TCP_KEEPIDLE*) в 30 минут, для интервала проб (*TCP_KEEPINTVL*) будет действовать стандартная системная настройка, а счётчик проб (*TCP_KEEPCNT*) будет равен 10.

preread_buffer_size

Синтаксис preread_buffer_size *размер*;

Умолчание preread_buffer_size 16k;

Контекст stream, server

Задаёт размер буфера предварительного чтения.

preread_timeout

Синтаксис preread_timeout *время*;

Умолчание preread_timeout 30s;

Контекст stream, server

Задаёт время фазы предварительного чтения.

proxy_protocol_timeout

Синтаксис proxy_protocol_timeout *время*;

Умолчание proxy_protocol_timeout 30s;

Контекст stream, server

Задаёт время для завершения операции чтения заголовка протокола PROXY. Если по истечении этого времени заголовок полностью не получен, соединение закрывается.

resolver

Синтаксис resolver *адрес* … [valid= *время*] [ipv4=on|off] [ipv6=on|off] [status_zone= *зона*];

Умолчание —

Контекст stream, server

Задаёт серверы DNS, используемые для преобразования имён вышестоящих серверов в адреса, например:

```
resolver 127.0.0.1 [::1]:5353;
```

Адрес может быть указан в виде доменного имени или IP-адреса, и необязательного порта. Если порт не указан, используется порт 53. Серверы DNS опрашиваются циклически.

По умолчанию Angie кэширует ответы, используя значение TTL из ответа. Необязательный параметр valid позволяет это переопределить:

valid	*необязательный* параметр, позволяет переопределить срок кэширования ответа

```
resolver 127.0.0.1 [::1]:5353 valid=30s;
```

По умолчанию Angie будет искать как IPv4-, так и IPv6-адреса при преобразовании имён в адреса.

ipv4=off	запрещает поиск IPv4-адресов
ipv6=off	запрещает поиск IPv6-адресов
status_zone	*необязательный* параметр, включает сбор информации о запросах и ответах сервера DNS в указанной зоне

Примечание

Для предотвращения DNS-спуфинга рекомендуется использовать DNS-серверы в защищённой доверенной локальной сети.

resolver_timeout

Синтаксис resolver_timeout *время;*

Умолчание resolver_timeout 30s;

Контекст stream, server

Задаёт таймаут для преобразования имени в адрес, например:

```
resolver_timeout 5s;
```

server

Синтаксис server { … }

Умолчание —

Контекст stream

Задаёт конфигурацию для сервера.

status_zone

Синтаксис status_zone *зона*;

Умолчание —

Конеткст *server*

Директива включает выделение зоны в разделяемой памяти для сбора метрик. Несколько *server* могут разделять одну зону для сбора данных.

stream

Синтаксис stream { … }

Умолчание —

Контекст main

Предоставляет контекст конфигурационного файла, в котором указываются директивы stream-сервера.

tcp_nodelay

Синтаксис tcp_nodelay on | off;

Умолчание tcp_nodelay on;

Контекст stream, server

Разрешает или запрещает использование параметра *TCP_NODELAY*. Параметр включается как для клиентских соединений, так и для соединений с проксируемыми серверами.

variables_hash_bucket_size

Синтаксис variables_hash_bucket_size *размер;*

Умолчание variables_hash_bucket_size 64;

Контекст stream

Задаёт размер корзины в хэш-таблице переменных. Подробнее настройка хэш-таблиц обсуждается отдельно.

variables_hash_max_size

Синтаксис variables_hash_max_size *размер;*

Умолчание variables_hash_max_size 1024;

Контекст stream

Задаёт максимальный размер хэш-таблиц переменных. Подробнее настройка хэш-таблиц обсуждается отдельно.

Встроенные переменные

Модуль *stream core* поддерживает встроенные переменные:

$angie_version

версия Angie

$binary_remote_addr

адрес клиента в бинарном виде, длина значения всегда 4 байта для IPv4-адресов или 16 байт для IPv6-адресов

$bytes_received

число байт, полученных от клиента

$bytes_sent

число байт, переданных клиенту

$connection

порядковый номер соединения

$hostname

имя хоста

$msec

текущее время в секундах с точностью до миллисекунд

$pid

номер (PID) рабочего процесса

$protocol

протокол, используемый для работы с клиентом: *TCP* или *UDP*

$proxy_protocol_addr

адрес клиента, полученный из заголовка протокола PROXY Протокол PROXY должен быть предварительно включён при помощи установки параметра proxy_protocol в директиве listen.

$proxy_protocol_port

порт клиента, полученный из заголовка протокола PROXY Протокол PROXY должен быть предварительно включён при помощи установки параметра proxy_protocol в директиве listen.

$proxy_protocol_server_addr

адрес сервера, полученный из заголовка протокола PROXY Протокол PROXY должен быть предварительно включён при помощи установки параметра proxy_protocol в директиве listen.

$proxy_protocol_server_port

порт сервера, полученный из заголовка протокола PROXY Протокол PROXY должен быть предварительно включён при помощи установки параметра proxy_protocol в директиве listen.

$proxy_protocol_tlv_ имя

TLV, полученный из заголовка протокола PROXY. *Имя* может быть именем типа TLV или его числовым значением. В последнем случае значение задаётся в шестнадцатеричном виде и должно начинаться с *0x*:

```
$proxy_protocol_tlv_alpn
$proxy_protocol_tlv_0x01
```

SSL TLV могут также быть доступны как по имени типа TLV, так и по его числовому значению, оба должны начинаться с *ssl_*:

```
$proxy_protocol_tlv_ssl_version
$proxy_protocol_tlv_ssl_0x21
```

Поддерживаются следующие имена типов TLV:

- *alpn (0x01)* - протокол более высокого уровня, используемый поверх соединения
- *authority (0x02)* - значение имени хоста, передаваемое клиентом
- *unique_id (0x05)* - уникальный идентификатор соединения
- *netns (0x30)* - имя пространства имён
- *ssl (0x20)* - структура SSL TLV в бинарном виде

Поддерживаются следующие имена типов SSL TLV:

- *ssl_version (0x21)* - версия SSL, используемая в клиентском соединении
- *ssl_cn (0x22)* - Common Name сертификата
- *ssl_cipher (0x23)* - имя используемого шифра
- *ssl_sig_alg (0x24)* - алгоритм, используемый для подписи сертификата
- *ssl_key_alg (0x25)* - алгоритм публичного ключа

Также поддерживается следующее специальное имя типа SSL TLV:

- *ssl_verify* - результат проверки клиентского сертификата: 0, если клиент предоставил сертификат и он был успешно верифицирован, либо ненулевое значение

Протокол PROXY должен быть предварительно включён при помощи установки параметра proxy_protocol в директиве listen.

$remote_addr

адрес клиента

$remote_port

порт клиента

$server_addr

адрес сервера, принявшего соединение Получение значения этой переменной обычно требует одного системного вызова. Чтобы избежать системного вызова, в директивах listen следует указывать адреса и использовать параметр *bind*.

$server_port

порт сервера, принявшего соединение

$session_time

длительность сессии в секундах с точностью до миллисекунд

$status

статус сессии, может принимать одно из следующих значений:

200	сессия завершена успешно
400	невозможно разобрать данные, полученные от клиента, например заголовок протокола PROXY
403	доступ запрещён, например при ограничении доступа для определённых адресов клиентов
500	внутренняя ошибка сервера
502	плохой шлюз, например если невозможно выбрать сервер группы или сервер недоступен
503	сервис недоступен, например при ограничении по числу соединений

$time_iso8601

локальное время в формате по стандарту ISO 8601

$time_local

локальное время в Common Log Format

Модуль stream_geo

Создаёт переменные, значения которых зависят от IP-адреса клиента.

Пример конфигурации

```
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;
}
```

Директивы

geo

Синтаксис geo [*$адрес*] *$переменная* { … }

Умолчание —

Контекст stream

Описывает для указанной переменной зависимость значения от IP-адреса клиента. По умолчанию адрес берётся из переменной $remote_addr, но его также можно получить из другой переменной, например:

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

Примечание

Поскольку переменные вычисляются только в момент использования, само по себе наличие даже большого числа объявлений переменных “*geo*” не влечёт за собой никаких дополнительных расходов на обработку соединений.

Если значение переменной не представляет из себя правильный IP-адрес, то используется адрес “255.255.255.255”.

Адреса задаются либо префиксами в формате CIDR (включая одиночные адреса), либо в виде диапазонов.

Также поддерживаются следующие специальные параметры:

delete	удаляет описанную сеть
default	значение переменной, если адрес клиента не соответствует ни одному из заданных адресов. При задании адресов в формате CIDR вместо *default* можно использовать “*0.0.0.0/0*” и “*::/0*”. Если параметр *default* не указан, значением по умолчанию будет пустая строка.
include	включает файл с адресами и значениями. Включений может быть несколько.
ranges	указывает, что адреса задаются в виде диапазонов. Этот параметр должен быть первым. Для ускорения загрузки гео-базы нужно располагать адреса в порядке возрастания.

Пример:

```
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;
}
```

В файле *conf/geo.conf* могут быть такие строки:

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

В качестве значения выбирается максимальное совпадение, например, для адреса *127.0.0.1* будет выбрано значение “*RU*”, а не “*US*”.

Пример описания диапазонов:

```
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;
}
```

Модуль stream_geoip

Создаёт переменные, значения которых зависят от IP-адреса клиента, используя готовые базы данных MaxMind.

При использовании баз данных с поддержкой IPv6 IPv4-адреса ищутся отображёнными на IPv6.

По умолчанию этот модуль не собирается, его сборку необходимо разрешить с помощью конфигурационного параметра *--with-stream_geoip_module*.

Внимание

Для сборки и работы этого модуля нужна библиотека MaxMind GeoIP.

Пример конфигурации

```
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;
    }
#   ...
}
```

Директивы

geoip_country

Синтаксис geoip_country *файл*;

Умолчание —

Контекст stream

Задаёт базу данных для определения страны в зависимости от значения IP-адреса клиента. При использовании этой базы данных доступны следующие переменные:

$geoip_country_code	двухбуквенный код страны, например, “RU”, “US”.

$geoip_country_code3	трёхбуквенный код страны, например, “RUS”, “USA”.
$geoip_country_name	название страны, например, “Russian Federation”, “United States”.

geoip_city

Синтаксис geoip_city *файл*;

Умолчание —

Контекст stream

Задаёт базу данных для определения страны, региона и города в зависимости от значения IP-адреса клиента. При использовании этой базы данных доступны следующие переменные:

$geoip_city_continent_code	двухбуквенный код континента, например, “EU”, “NA”.
$geoip_city_country_code	двухбуквенный код страны, например, “RU”, “US”.
$geoip_city_country_code3	трёхбуквенный код страны, например, “RUS”, “USA”.
$geoip_city_country_name	название страны, например, “Russian Federation”, “United States”.
$geoip_dma_code	DMA-код региона в США (также известный как “код агломерации”), согласно геотаргетингу Google AdWords API.
$geoip_latitude	широта.
$geoip_longitude	долгота.
$geoip_region	двухсимвольный код региона страны (область, край, штат, провинция, федеральная земля и тому подобное), например, “48”, “DC”.
$geoip_region_name	название региона страны (область, край, штат, провинция, федеральная земля и тому подобное), например, “Moscow City”, “District of Columbia”.
$geoip_city	название города, например, “Moscow”, “Washington”.
$geoip_postal_code	почтовый индекс.

geoip_org

Синтаксис geoip_org *файл*;

Умолчание —

Контекст stream

Задаёт базу данных для определения названия организации в зависимости от значения IP-адреса клиента. При использовании этой базы данных доступна следующая переменная:

$geoip_org	название организации, например, “The University of Melbourne”.

Модуль stream_js

Позволяет задавать обработчики на njs — подмножестве языка JavaScript.

Пример конфигурации

```
stream {
    js_import stream.js;

    js_set $bar stream.bar;
    js_set $req_line stream.req_line;

    server {
        listen 12345;

        js_preread stream.preread;
        return     $req_line;
    }

    server {
        listen 12346;

        js_access  stream.access;
        proxy_pass 127.0.0.1:8000;
        js_filter  stream.header_inject;
    }
}

http {
    server {
        listen 8000;
        location / {
            return 200 $http_foo\n;
        }
    }
}
```

Файл stream.js:

```
var line = '';

function bar(s) {
    var v = s.variables;
    s.log("hello from bar() handler!");
    return "bar-var" + v.remote_port + "; pid=" + v.pid;
}

function preread(s) {
    s.on('upload', function (data, flags) {
        var n = data.indexOf('\n');
        if (n != -1) {
            line = data.substr(0, n);
            s.done();
        }
    });
```

```
}

function req_line(s) {
    return line;
}

// Чтение строки HTTP-запроса.
// Получение байт в 'req' до того как
// будет прочитана строка запроса.
// Добавление HTTP-заголовка в запрос клиента

var my_header =  'Foo: foo';
function header_inject(s) {
    var req = '';
    s.on('upload', function(data, flags) {
        req += data;
        var n = req.search('\n');
        if (n != -1) {
            var rest = req.substr(n + 1);
            req = req.substr(0, n + 1);
            s.send(req + my_header + '\r\n' + rest, flags);
            s.off('upload');
        }
    });
}

function access(s) {
    if (s.remoteAddress.match('^192.*')) {
        s.deny();
        return;
    }

    s.allow();
}

export default {bar, preread, req_line, header_inject, access};
```

Директивы

js_access

Синтаксис js_access *функция* | *модуль.функция*;

Умолчание —

Контекст stream, server

Задаёт функцию njs, которая будет вызываться в access-фазе. Можно ссылаться на функцию модуля.

Функция вызывается однократно при первом достижении сессией access-фазе. Функция вызывается со следующими аргументами:

s	объект stream-сессии

В этой фазе может происходить инициализация, также при помощи метода s.on() может регистрироваться вызов для каждого входящего блока данных пока не будет вызван один из методов: s.done(), s.decline(), s.allow(). При вызове любого из этих методов обработка сессии переходит на следующую фазу и все текущие вызовы s.on() сбрасываются.

js_fetch_buffer_size

Синтаксис js_fetch_buffer_size *размер*;

Умолчание js_fetch_buffer_size 16k;

Контекст stream, server

Задаёт размер буфера, который будет использоваться для чтения и записи для Fetch API.

js_fetch_ciphers

Синтаксис js_fetch_ciphers *шифры*;

Умолчание js_fetch_ciphers HIGH:!aNULL:!MD5;

Контекст stream, server

Описывает разрешённые шифры для HTTPS-соединений при помощи Fetch API. Шифры задаются в формате, поддерживаемом библиотекой OpenSSL.

Полный список можно посмотреть с помощью команды “*openssl ciphers*”.

js_fetch_max_response_buffer_size

Синтаксис js_fetch_max_response_buffer_size *размер*;

Умолчание js_fetch_max_response_buffer_size 1m;

Контекст stream, server

Задаёт максимальный размер ответа, полученного при помощи Fetch API.

js_fetch_protocols

Синтаксис js_fetch_protocols [TLSv1] [TLSv1.1] [TLSv1.2] [TLSv1.3];

Умолчание js_fetch_protocols TLSv1 TLSv1.1 TLSv1.2;

Контекст stream, server

Разрешает указанные протоколы для HTTPS-соединений при помощи Fetch API.

js_fetch_timeout

Синтаксис js_fetch_timeout *время*;

Умолчание js_fetch_timeout 60s;

Контекст stream, server

Задаёт таймаут при чтении и записи при помощи Fetch API. Таймаут устанавливается не на всю передачу ответа, а только между двумя операциями чтения. Если по истечении этого времени данные не передавались, соединение закрывается.

js_fetch_trusted_certificate

Синтаксис js_fetch_trusted_certificate *файл*;

Умолчание —

Контекст stream, server

Задаёт файл с доверенными сертификатами CA в формате PEM, используемыми при проверке HTTPS-сертификата при помощи Fetch API.

js_fetch_verify

Синтаксис js_fetch_verify on | off;

Умолчание js_fetch_verify on;

Контекст stream, server

Разрешает или запрещает проверку сертификата HTTPS-сервера при помощи Fetch API.

js_fetch_verify_depth

Синтаксис js_fetch_verify_depth *число*;

Умолчание js_fetch_verify_depth 100;

Контекст stream, server

Устанавливает глубину проверки в цепочке HTTPS-сертификатов при помощи Fetch API.

js_filter

Синтаксис js_filter *функция* | *модуль.функция*;

Умолчание —

Контекст stream, server

Задаёт фильтр данных. Можно ссылаться на функцию модуля.

Функция фильтра вызывается однократно при первом достижении сессией content-фазы. Функция фильтра вызывается со следующими аргументами:

s	объект stream-сессии

В этой фазе может происходить инициализация, также при помощи метода s.on() может регистрироваться вызов для каждого входящего блока данных. Для отмены регистрации вызова и отмены фильтра можно использовать метод s.off().

Примечание

Так как обработчик *js_filter* должен сразу возвращать результат, то поддерживаются только синхронные операции. Таким образом, асинхронные операции, например ngx.fetch() или setTimeout(), не поддерживаются.

js_import

Синтаксис js_import *модуль.js* | *имя_экспорта* from *модуль.js*;

Умолчание —

Контекст stream, server

Импортирует модуль, позволяющий задавать обработчики location и переменных на njs. *Имя_экспорта* является пространством имён при доступе к функциям модуля. Если *имя_экспорта* не задано, то пространством имён будет являться имя модуля.

```
js_import stream.js;
```

В примере при доступе к экспорту в качестве пространства имён используется имя модуля *stream*. Если импортируемый модуль экспортирует *foo()*, то для доступа используется *stream.foo*.

Директив *js_import* может быть несколько.

js_path

Синтаксис js_path *путь*;

Умолчание —

Контекст stream, server

Задаёт дополнительный путь для модулей njs.

js_preload_object

Синтаксис js_preload_object *имя.json* | *имя from файл.json*;

Умолчание —

Контекст stream, server

Предварительно загружает неизменяемый объект во время конфигурации. *Имя* используется в качестве имени глобальной переменной, через которую объект доступен в коде njs. Если *имя* не указано, то будет использоваться имя файла.

```
js_preload_object map.json;
```

В примере *map* используется в качестве имени во время доступа к предварительно загруженному объекту.

Директив js_preload_object может быть несколько.

js_preread

Синтаксис js_preread *функция* | *модуль.функция*;

Умолчание —

Контекст stream, server

Задаёт функцию njs, которая будет вызываться в preread-фазе. Можно ссылаться на функцию модуля.

Функция вызывается однократно при первом достижении сессией preread-фазы. Функция вызывается со следующими аргументами:

s	объект stream-сессии

В этой фазе может происходить инициализация, также при помощи метода s.on() может регистрироваться вызов для каждого входящего блока данных пока не будет вызван один из методов: s.done(), s.decline(), s.allow(). При вызове любого из этих методов обработка сессии переходит на следующую фазу и все текущие вызовы s.on() сбрасываются.

Примечание

Так как обработчик *js_preread* должен сразу возвращать результат, то поддерживаются только синхронные операции. Таким образом, асинхронные операции, например ngx.fetch() или setTimeout(), не поддерживаются. Тем не менее асинхронные операции поддерживаются в вызовах s.on() в preread-фазе.

js_set

Синтаксис js_set *$переменная функция* | *модуль.функция*;

Умолчание —

Контекст stream, server

Задаёт функцию njs для указанной переменной. Можно ссылаться на функцию модуля.

Функция вызывается в момент первого обращения к переменной для данного запроса. Точный момент вызова функции зависит от фазы, в которой происходит обращение к переменной. Это можно использовать для реализации дополнительной логики, не относящейся к вычислению переменной. Например, если переменная указана в директиве log_format, то её обработчик не будет выполняться до фазы записи в лог. Этот обработчик также может использоваться для выполнения процедур непосредственно перед освобождением запроса.

Примечание

Так как обработчик *js_set* должен сразу возвращать результат, то поддерживаются только синхронные операции. Таким образом, асинхронные операции, например ngx.fetch() или setTimeout(), не поддерживаются.

js_var

Синтаксис js_var *$переменная* [*значение*];

Умолчание —

Контекст stream, server

Объявляет перезаписываемую переменную. В качестве значения можно использовать текст, переменные и их комбинации.

Свойства объекта сессии

Каждый stream-обработчик njs получает один аргумент, объект stream-сессии.

Модуль stream_limit_conn

Позволяет ограничить число соединений по заданному ключу, в частности, число соединений с одного IP-адреса.

Пример конфигурации

```
stream {
    limit_conn_zone $binary_remote_addr zone=addr:10m;

    ...

    server {

        ...

        limit_conn           addr 1;
        limit_conn_log_level error;
    }
}
```

Директивы

limit_conn

Синтаксис limit_conn *зона число*;

Умолчание —

Контекст stream, server

Задаёт зону разделяемой памяти и максимально допустимое число соединений для одного значения ключа. При превышении этого числа сервер закроет соединение. Например, директивы

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

server {
    ...
    limit_conn addr 1;
}
```

разрешают одновременно обрабатывать не более одного соединения с одного IP-адреса.

Допустимо одновременное указание нескольких директив *limit_conn*, при этом будет срабатывать любое из ограничений.

Директивы наследуются с предыдущего уровня конфигурации при условии, что на данном уровне не описаны свои директивы *limit_conn*.

limit_conn_dry_run

Синтаксис limit_conn_dry_run on | off;

Умолчание limit_conn_dry_run off;

Контекст stream, server

Включает режим пробного запуска. В данном режиме число соединений не ограничивается, однако в зоне разделяемой памяти текущее число избыточных соединений учитывается как обычно.

limit_conn_log_level

Синтаксис limit_conn_log_level info | notice | warn | error;

Умолчание limit_conn_log_level error;

Контекст stream, server

Задаёт желаемый уровень записи в лог случаев ограничения числа соединений.

limit_conn_zone

Синтаксис limit_conn_zone *ключ* zone = *название:размер*;

Умолчание —

Контекст stream

Задаёт параметры зоны разделяемой памяти, которая хранит состояние для разных значений ключа. Состояние в частности содержит текущее число соединений. В качестве ключа может использоваться текст, переменные и их комбинации. Запросы с пустым значением ключа не учитываются.

Пример использования:

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

Здесь в качестве ключа используется IP-адрес клиента, задаваемый переменной $binary_remote_addr.

Длина значения *$binary_remote_addr* равна 4 байтам для IPv4-адресов или 16 байтам для IPv6-адресов. При этом размер состояния всегда равен 32 или 64 байтам на 32-битных платформах и 64 байтам на 64-битных. В зоне размером 1 мегабайт может разместиться около 32 тысяч состояний размером 32 байта или 16 тысяч состояний размером 64 байта. При переполнении зоны сервер закроет соединение.

Встроенные переменные

$limit_conn_status

хранит результат ограничения числа соединений: PASSED, REJECTED или REJECTED_DRY_RUN

Модуль stream_log

Модуль записывает логи запросов в указанном формате.

Пример конфигурации

```
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;
```

Директивы

access_log

Синтаксис access_log *путь* [*формат* [buffer = *размер*] [gzip[= *степень*]] [flush = *время*] [if = *условие*]]; access_log off;

Умолчание access_log off;

Контекст stream, server

Задаёт путь, формат и настройки буферизованной записи в лог. На одном уровне конфигурации может использоваться несколько логов. Запись в syslog настраивается указанием префикса “syslog:” в первом параметре. Специальное значение off отменяет все директивы access_log для текущего уровня.

Если задан размер буфера с помощью параметра buffer или указан параметр gzip, то запись будет буферизованной.

Примечание

Размер буфера должен быть не больше размера атомарной записи в дисковый файл. Для FreeBSD этот размер неограничен.

При включённой буферизации данные записываются в файл:

- если очередная строка лога не помещается в буфер;
- если данные в буфере находятся дольше интервала времени, заданного параметром *flush*;
- при переоткрытии лог-файла или завершении рабочего процесса.

Если задан параметр *gzip*, то буфер будет сжиматься перед записью в файл. Степень сжатия может быть задана в диапазоне от *1* (быстрее, но хуже сжатие) до *9* (медленнее, но лучше сжатие). По умолчанию используются буфер размером *64K* байт и степень сжатия *1*. Данные сжимаются атомарными блоками, и в любой момент времени лог-файл может быть распакован или прочитан с помощью утилиты *“zcat”*.

Пример:

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

Примечание

Для поддержки gzip-сжатия логов Angie должен быть собран с библиотекой *zlib*.

В пути файла можно использовать переменные, но такие логи имеют некоторые ограничения:

- пользователь, с правами которого работают рабочие процессы, должен иметь права на создание файлов в каталоге с такими логами;
- не работает буферизация;
- файл открывается для каждой записи в лог и сразу же после записи закрывается. Следует однако иметь в виду, что поскольку дескрипторы часто используемых файлов могут храниться в кэше, то при ротации логов в течение времени, заданного параметром *valid* директивы open_log_file_cache, запись может продолжаться в старый файл.

Параметр *if* включает условную запись в лог. Сессия не будет записываться в лог, если результатом вычисления условия является *“0”* или пустая строка.

log_format

Синтаксис log_format *имя* [escape=default|json|none] *строка* ..;

Умолчание —

Контекст stream

Задаёт формат лога, например:

```
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"';
```

Параметр *escape* позволяет задать экранирование символов *json* или *default* в переменных, по умолчанию используется default. Значение none отключает экранирование символов.

При использовании default символы “"”, “\”, а также символы со значениями меньше 32 или больше 126 экранируются как “\xXX”. Если значение переменной не найдено, то в качестве значения в лог будет записываться дефис “-”.

При использовании *json* экранируются все символы, недопустимые в JSON строках: символы “"” и “\” экранируются как “\»” и “\\”, символы со значениями меньше 32 экранируются как “\n”, “\r”, “\t”, “\b”, “\f” или “\u00XX”.

open_log_file_cache

Синтаксис open_log_file_cache max = *N* [inactive = *время*] [min_uses = *N*] [valid = *время*]; open_log_file_cache off;

Умолчание open_log_file_cache off;

Контекст http, server, location

Задаёт кэш, в котором хранятся дескрипторы файлов часто используемых логов, имена которых заданы с использованием переменных. Параметры:

max	задаёт максимальное число дескрипторов в кэше; при переполнении кэша наименее востребованные (LRU) дескрипторы закрываются
inactive	задаёт время, после которого закэшированный дескриптор закрывается, если к нему не было обращений в течение этого времени; по умолчанию 10 секунд
min_uses	задаёт минимальное число использований файла в течение времени, заданного параметром *inactive*, после которого дескриптор файла будет оставаться открытым в кэше; по умолчанию 1
valid	задаёт, через какое время нужно проверять, что файл ещё существует под тем же именем; по умолчанию 60 секунд
off	запрещает кэш

Пример использования:

```
open_log_file_cache max=1000 inactive=20s valid=1m min_use
```

Модуль stream_map

Создаёт переменные, значения которых зависят от значений других переменных.

Пример конфигурации

```
map $remote_addr $limit {
    127.0.0.1     "";
    default       $binary_remote_addr;
}

limit_conn_zone $limit zone=addr:10m;
limit_conn addr 1;
```

Директивы

map

Синтаксис map *строка $переменная* { … };

Умолчание —

Контекст stream

Создаёт новую переменную, значение которой зависит от значений одной или более исходных переменных, указанных в первом параметре.

Примечание

Поскольку переменные вычисляются только в момент использования, само по себе наличие даже большого числа объявлений переменных “map” не влечёт за собой никаких дополнительных расходов на обработку запросов.

Параметры внутри блока map задают соответствие между исходными и результирующими значениями.

Исходные значения задаются строками или регулярными выражениями.

Строки проверяются без учёта регистра.

Перед регулярным выражением ставится символ “~”, если при сравнении следует учитывать регистр символов, либо символы “~*”, если регистр символов учитывать не нужно. Регулярное выражение может содержать именованные и позиционные выделения, которые могут затем использоваться в других директивах совместно с результирующей переменной.

Если исходное значение совпадает с именем одного из специальных параметров, описанных ниже, перед ним следует поставить символ “”.

В качестве результирующего значения можно указать текст, переменную и их комбинации.

Также поддерживаются следующие специальные параметры:

default *значение*	задаёт результирующее значение, если исходное значение не совпадает ни с одним из перечисленных. Если параметр *default* не указан, результирующим значением по умолчанию будет пустая строка.
hostnames	указывает, что в качестве исходных значений можно использовать маску для первой или последней части имени хоста. Этот параметр следует указывать перед списком значений.

Например,

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

Вместо двух записей

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

можно использовать одну:

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

include *файл*	включает файл со значениями. Включений может быть несколько.
volatile	указывает, что переменная не кэшируется.

Если исходному значению соответствует несколько из указанных вариантов, например, одновременно подходят и маска, и регулярное выражение, будет выбран первый подходящий вариант в следующем порядке приоритета:

1. строковое значение без маски
2. самое длинное строковое значение с маской в начале, например “*.example.com”
3. самое длинное строковое значение с маской в конце, например “mail.*”
4. первое подходящее регулярное выражение (в порядке следования в конфигурационном файле)
5. значение по умолчанию (*default*)

map_hash_bucket_size

Синтаксис map_hash_bucket_size *размер*;

Умолчание map_hash_bucket_size 32|64|128;

Контекст stream

Задаёт размер корзины в хэш-таблицах для переменных map. Значение по умолчанию зависит от размера строки кэша процессора. Подробнее настройка хэш-таблиц обсуждается отдельно.

map_hash_max_size

Синтаксис map_hash_max_size *размер*;

Умолчание map_hash_max_size 2048;

Контекст stream

Задаёт максимальный размер хэш-таблиц для переменных map. Подробнее настройка хэш-таблиц обсуждается отдельно.

Модуль stream_proxy

Позволяет проксировать потоки данных по TCP, UDP и UNIX-сокетам.

Пример конфигурации

```
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;
}
```

Директивы

proxy_bind

Синтаксис proxy_bind *адрес* [transparent] | off;

Умолчание —

Контекст stream, server

Задаёт локальный IP-адрес, который будет использоваться в исходящих соединениях с проксируемым сервером. В значении параметра допустимо использование переменных. Специальное значение off отменяет действие унаследованной с предыдущего уровня конфигурации директивы *proxy_bind*, позволяя системе самостоятельно выбирать локальный IP-адрес.

Параметр transparent позволяет задать нелокальный IP-адрес, который будет использоваться в исходящих соединениях с проксируемым сервером, например, реальный IP-адрес клиента:

```
proxy_bind $remote_addr transparent;
```

Для работы параметра обычно требуется запустить рабочие процессы Angie с привилегиями суперпользователя. В Linux этого не требуется, так как если указан параметр *transparent*, то рабочие процессы наследуют capability *CAP_NET_RAW* из главного процесса.

Внимание

Необходимо настроить таблицу маршрутизации ядра для перехвата сетевого трафика с проксируемого сервера.

proxy_buffer_size

Синтаксис proxy_buffer_size *размер*;

Умолчание proxy_buffer_size 16k;

Контекст stream, server

Задаёт размер буфера, в который будут читаться данные, получаемые от проксируемого сервера. Также задаёт размер буфера, в который будут читаться данные, получаемые от клиента.

proxy_connect_timeout

Синтаксис proxy_connect_timeout *время*;

Умолчание proxy_connect_timeout 60s;

Контекст stream, server

Задаёт таймаут для установления соединения с проксируемым сервером.

proxy_download_rate

Синтаксис proxy_download_rate *скорость*;

Умолчание proxy_download_rate 0;

Контекст stream, server

Ограничивает скорость чтения данных от проксируемого сервера. Скорость задаётся в байтах в секунду.

0	отключает ограничение скорости

Примечание

Ограничение устанавливается на соединение, поэтому, если Angie одновременно откроет два соединения к проксируемому серверу, суммарная скорость будет вдвое выше заданного ограничения.

В значении параметра можно использовать переменные. Это может быть полезно в случаях, когда скорость нужно ограничивать в зависимости от какого-либо условия:

```
map $slow $rate {
    1     4k;
    2     8k;
}

proxy_download_rate $rate
```

proxy_half_close

Синтаксис proxy_half_close on | off;

Умолчание proxy_half_close off;

Контекст stream, server

Разрешает или запрещает независимое закрытие каждой из сторон проксируемого соединения TCP (“TCP half-close”). Если разрешено, то проксирование по TCP будет продолжаться, пока обе стороны не закроют соединение.

proxy_next_upstream

Синтаксис proxy_next_upstream on | off;

Умолчание proxy_next_upstream on;

Контекст stream, server

При невозможности установить соединение с проксируемым сервером определяет, будет ли клиентское соединение передано следующему серверу.

Передача соединения следующему серверу может быть ограничена по количеству попыток и по времени.

proxy_next_upstream_timeout

Синтаксис proxy_next_upstream_timeout *время*;

Умолчание proxy_next_upstream_timeout 0;

Контекст stream, server

Ограничивает время, в течение которого возможна передача запроса следующему серверу.

0	отключает это ограничение

proxy_next_upstream_tries

Синтаксис proxy_next_upstream_tries *число*;

Умолчание proxy_next_upstream_tries 0;

Контекст stream, server

Ограничивает число допустимых попыток для передачи запроса следующему серверу.

0	отключает это ограничение

proxy_pass

Синтаксис proxy_pass *адрес*;

Умолчание —

Контекст server

Задаёт адрес проксируемого сервера. Адрес может быть указан в виде доменного имени или IP-адреса, и порта:

```
proxy_pass localhost:12345;
```

или в виде пути UNIX-сокета:

```
proxy_pass unix:/tmp/stream.socket;
```

Если доменному имени соответствует несколько адресов, то все они будут использоваться по очереди (round-robin). Кроме того, в качестве адреса можно указать группу серверов.

Адрес можно также задать с помощью переменных:

```
proxy_pass $upstream;
```

В этом случае имя сервера ищется среди описанных групп серверов и если не найдено, то определяется с помощью resolver'а.

proxy_protocol

Синтаксис proxy_protocol on | off;

Умолчание proxy_protocol off;

Контекст stream, server

Включает протокол PROXY для соединений с проксируемым сервером.

proxy_requests

Синтаксис proxy_requests *число*;

Умолчание proxy_requests 0;

Контекст stream, server

Задаёт число датаграмм, полученных от клиента, по достижении которого удаляется привязка между клиентом и существующей UDP-сессией. После получения указанного количества датаграмм следующая датаграмма, полученная от того же клиента, начинает новую сессию. Сессия завершится после отправки всех принятых датаграмм на проксируемый сервер и получения указанного количества ответов или после таймаута.

proxy_responses

Синтаксис proxy_responses *число*;

Умолчание —

Контекст stream, server

Задаёт количество датаграмм, ожидаемых от проксируемого сервера в ответ на датаграмму клиента в случае, если используется протокол UDP. Задаваемое число служит подсказкой для завершения сессии. По умолчанию количество датаграмм не ограничено.

Если указано нулевое значение, то ответ не ожидается. Однако если ответ получен и сессия ещё не завершилась, то ответ будет обработан.

proxy_socket_keepalive

Синтаксис proxy_socket_keepalive on | off;

Умолчание proxy_socket_keepalive off;

Контекст stream, server

Конфигурирует поведение “TCP keepalive” для исходящих соединений к проксируемому серверу.

""	По умолчанию для сокета действуют настройки операционной системы.
on	для сокета включается параметр *SO_KEEPALIVE*

proxy_ssl

Синтаксис proxy_ssl on | off;

Умолчание proxy_ssl off;

Контекст stream, server

Включает протоколы SSL/TLS для соединений с проксируемым сервером.

proxy_ssl_certificate

Синтаксис proxy_ssl_certificate *файл*;

Умолчание —

Контекст stream, server

Задаёт файл с сертификатом в формате PEM для аутентификации на проксируемом сервере. В имени файла можно использовать переменные.

proxy_ssl_certificate_key

Синтаксис proxy_ssl_certificate_key *файл*;

Умолчание —

Контекст stream, server

Задаёт файл с секретным ключом в формате PEM для аутентификации на проксируемом сервере. В имени файла можно использовать переменные.

proxy_ssl_ciphers

Синтаксис proxy_ssl_ciphers *шифры*;

Умолчание proxy_ssl_ciphers DEFAULT;

Контекст stream, server

Описывает разрешённые шифры для запросов к проксируемому серверу. Шифры задаются в формате, поддерживаемом библиотекой OpenSSL.

Полный список можно посмотреть с помощью команды “*openssl ciphers*”.

proxy_ssl_conf_command

Синтаксис proxy_ssl_conf_command *имя значение*;

Умолчание —

Контекст stream, server

Задаёт произвольные конфигурационные команды OpenSSL при установлении соединения с проксируемым сервером.

Примечание

Директива поддерживается при использовании OpenSSL 1.0.2 и выше.

На одном уровне может быть указано несколько директив *proxy_ssl_conf_command*. Директивы наследуются с предыдущего уровня конфигурации при условии, что на данном уровне не описаны свои директивы *proxy_ssl_conf_command*.

Внимание

Следует учитывать, что изменение настроек OpenSSL напрямую может привести к неожиданному поведению.

proxy_ssl_crl

Синтаксис proxy_ssl_crl *файл*;

Умолчание —

Контекст stream, server

Указывает файл с отозванными сертификатами (CRL) в формате PEM, используемыми при проверке сертификата проксируемого сервера.

proxy_ssl_name

Синтаксис proxy_ssl_name *имя*;

Умолчание proxy_ssl_name *хост из* proxy_pass;

Контекст stream, server

Позволяет переопределить имя сервера, используемое при проверке сертификата проксируемого сервера, а также для передачи его через SNI при установлении соединения с проксируемым сервером. Имя сервера можно также задать с помощью переменных.

По умолчанию используется имя хоста из адреса, заданного директивой proxy_pass.

proxy_ssl_password_file

Синтаксис proxy_ssl_password_file *файл*;

Умолчание —

Контекст stream, server

Задаёт файл с паролями от секретных ключей, где каждый пароль указан на отдельной строке. Пароли применяются по очереди в момент загрузки ключа.

proxy_ssl_protocols

Синтаксис proxy_ssl_protocols [SSLv2] [SSLv3] [TLSv1] [TLSv1.1] [TLSv1.2] [TLSv1.3];

Умолчание proxy_ssl_protocols TLSv1 TLSv1.1 TLSv1.2;

Контекст stream, server

Разрешает указанные протоколы для соединений с проксируемым сервером.

proxy_ssl_server_name

Синтаксис proxy_ssl_server_name on | off;

Умолчание proxy_ssl_server_name off;

Контекст stream, server

Разрешает или запрещает передачу имени сервера через расширение Server Name Indication протокола TLS (SNI, RFC 6066) при установлении соединения с проксируемым сервером.

proxy_ssl_session_reuse

Синтаксис proxy_ssl_session_reuse on | off;

Умолчание proxy_ssl_session_reuse on;

Контекст stream, server

Определяет, использовать ли повторно SSL-сессии при работе с проксируемым сервером. Если в логах появляются ошибки “*SSL3_GET_FINISHED:digest check failed*”, то можно попробовать выключить повторное использование сессий.

proxy_ssl_trusted_certificate

Синтаксис proxy_ssl_trusted_certificate *файл*;

Умолчание —

Контекст stream, server

Задаёт файл с доверенными сертификатами CA в формате PEM, используемыми при проверке сертификата проксируемого HTTPS-сервера.

proxy_ssl_verify

Синтаксис proxy_ssl_verify on | off;

Умолчание proxy_ssl_verify off;

Контекст stream, server

Разрешает или запрещает проверку сертификата проксируемого сервера.

proxy_ssl_verify_depth

Синтаксис proxy_ssl_verify_depth *число*;

Умолчание proxy_ssl_verify_depth 1;

Контекст stream, server

Устанавливает глубину проверки в цепочке сертификатов проксируемого сервера.

proxy_timeout

Синтаксис proxy_timeout *время*;

Умолчание proxy_timeout 10m;

Контекст stream, server

Задаёт таймаут между двумя идущими подряд операциями чтения или записи на клиентском соединении или соединении с проксируемым сервером. Если по истечении этого времени данные не передавались, соединение закрывается.

proxy_upload_rate

Синтаксис proxy_upload_rate *скорость*;

Умолчание proxy_upload_rate 0;

Контекст stream, server

Ограничивает скорость чтения данных от клиента. Скорость задаётся в байтах в секунду.

0	отключает ограничение скорости

Примечание

Ограничение устанавливается на соединение, поэтому, если клиент одновременно откроет два соединения, суммарная скорость будет вдвое выше заданного ограничения.

В значении параметра можно использовать переменные. Это может быть полезно в случаях, когда скорость нужно ограничивать в зависимости от какого-либо условия:

```
proxy_upload_rate $rate;
map $slow $rate {
    1     4k;
    2     8k;
}

proxy_upload_rate $rate;
```

Модуль stream_realip

Позволяет менять адрес и порт клиента на переданные в заголовке протокола PROXY. Протокол PROXY должен быть предварительно включён при помощи установки параметра *proxy_protocol* в директиве listen.

По умолчанию этот модуль не собирается, его сборку необходимо разрешить с помощью конфигурационного параметра *--with-stream_realip_module*.

Пример конфигурации

```
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;
```

Директивы

set_real_ip_from

Синтаксис set_real_ip_from *адрес* | *CIDR* | unix:;

Умолчание —

Контекст stream, server

Задаёт доверенные адреса, которые передают верный адрес для замены. Если указано специальное значение “*unix:*“, доверенными будут считаться все UNIX-сокеты.

Встроенные переменные

$realip_remote_addr

хранит исходный адрес клиента

$realip_remote_port

хранит исходный порт клиента

Модуль stream_return

Позволяет отправить заданное значение клиенту и после этого закрыть соединение.

Пример конфигурации

```
server {
    listen 12345;
    return $time_iso8601;
}
```

Директивы

return

Синтаксис return *значение*;

Умолчание —

Контекст server

Задаёт значение, отправляемое клиенту. В качестве значения можно использовать текст, переменные и их комбинации.

Модуль stream_set

Позволяет устанавливать значение переменной.

Пример конфигурации

```
server {
    listen 12345;
    set    $true 1;
}
```

Директивы

set

СинтаксисЁ set *$переменная значение*;

Умолчание —

Контекст server

Устанавливает значение указанной переменной. В качестве значения можно использовать текст, переменные и их комбинации.

Модуль stream_split_clients

Создаёт переменные для A/B тестирования (также известного как “split-тестирование”).

Пример конфигурации

```
stream {
    ...
    split_clients "${remote_addr}AAA" $upstream {
                  0.5%                feature_test1;
                  2.0%                feature_test2;
                  *                   production;
    }

    server {
        ...
        proxy_pass $upstream;
    }
}
```

Директивы

split_clients

Синтаксис split_clients *строка $переменная* { … }

Умолчание —

Контекст stream

Создаёт переменную для A/B тестирования, например:

```
split_clients "${remote_addr}AAA" $variant {
               0.5%               .one;
               2.0%               .two;
               *                  "";
}
```

Значение исходной строки хэшируется с помощью MurmurHash2. В приведённом примере при значениях хэша от 0 до 21474835 (0.5%) переменная *$variant* получит значение «*.one*». При значениях хэша от 21474836 до 107374180 (2%) — «*.two*». И при значениях хэша от 107374181 до 4294967295 — «» (пустая строка).

Модуль stream_ssl

Обеспечивает необходимую поддержку для работы прокси-сервера по протоколу SSL/TLS.

По умолчанию этот модуль не собирается, его сборку необходимо разрешить с помощью конфигурационного параметра *--with-stream_ssl_module*.

Примечание

Для сборки и работы этого модуля нужна библиотека OpenSSL.

Пример конфигурации

Для уменьшения загрузки процессора рекомендуется

- установить число рабочих процессов равным числу процессоров,
- включить разделяемый кэш сессий,
- выключить встроенный кэш сессий
- и, возможно, увеличить время жизни сессии (по умолчанию 5 минут):

```
worker_processes auto;

stream {

    #...

    server {
        listen              12345 ssl;

        ssl_protocols       TLSv1 TLSv1.1 TLSv1.2;
        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;

    #     ...
    }
```

Директивы

ssl_alpn

Синтаксис ssl_alpn *протокол* …;

Умолчание —

Контекст stream, server

Задаёт список поддерживаемых протоколов ALPN. Один из протоколов должен быть согласован, если клиент использует ALPN:

```
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

Синтаксис ssl_certificate *файл*;

Умолчание —

Контекст stream, server

Указывает файл с сертификатом в формате PEM для данного сервера. Если вместе с основным сертификатом нужно указать промежуточные, то они должны находиться в этом же файле в следующем порядке — сначала основной сертификат, а затем промежуточные. В этом же файле может находиться секретный ключ в формате PEM.

Директива может быть указана несколько раз для загрузки сертификатов разных типов, например RSA и 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;

#    ...
}
```

Возможность задавать отдельные цепочки сертификатов для разных сертификатов есть только в OpenSSL 1.0.2 и выше. Для более старых версий следует указывать только одну цепочку сертификатов.

Примечание

В имени файла можно использовать переменные при использовании OpenSSL 1.0.2 и выше:

```
ssl_certificate     $ssl_server_name.crt;
ssl_certificate_key $ssl_server_name.key;
```

При использовании переменных сертификат загружается при каждой операции SSL handshake, что может отрицательно влиять на производительность.

Вместо файла можно указать значение “data:*$переменная*“, при котором сертификат загружается из переменной без использования промежуточных файлов.

Ненадлежащее использование подобного синтаксиса может быть небезопасно, например данные секретного ключа могут попасть в лог ошибок.

ssl_certificate_key

Синтаксис ssl_certificate_key *файл*;

Умолчание —

Контекст stream, server

Указывает файл с секретным ключом в формате PEM для данного виртуального сервера.

Примечание

В имени файла можно использовать переменные при использовании OpenSSL 1.0.2 и выше.

Вместо файла можно указать значение “engine:*имя*:id“, которое загружает ключ с указанным *id* из OpenSSL engine с заданным именем.

Вместо файла можно указать значение “data:*$переменная*“, при котором секретный ключ загружается из переменной без использования промежуточных файлов. При этом следует учитывать, что ненадлежащее использование подобного синтаксиса может быть небезопасно, например данные секретного ключа могут попасть в лог ошибок.

ssl_ciphers

Синтаксис ssl_ciphers *шифры*;

Умолчание ssl_ciphers HIGH:!aNULL:!MD5;

Контекст stream, server

Описывает разрешённые шифры. Шифры задаются в формате, поддерживаемом библиотекой OpenSSL, например:

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

Полный список можно посмотреть с помощью команды “*openssl ciphers*”.

ssl_client_certificate

Синтаксис ssl_client_certificate *файл*;

Умолчание —

Контекст stream, server

Указывает файл с доверенными сертификатами CA в формате PEM, которые используются для проверки клиентских сертификатов.

Список сертификатов будет отправляться клиентам. Если это нежелательно, можно воспользоваться директивой ssl_trusted_certificate.

ssl_conf_command

Синтаксис ssl_conf_command *имя значение*;

Умолчание —

Контекст stream, server

Задаёт произвольные конфигурационные команды OpenSSL.

Примечание

Директива поддерживается при использовании OpenSSL 1.0.2 и выше.

На одном уровне может быть указано несколько директив *ssl_conf_command*:

```
ssl_conf_command Options PrioritizeChaCha;
ssl_conf_command Ciphersuites TLS_CHACHA20_POLY1305_SHA256;
```

Директивы наследуются с предыдущего уровня конфигурации при условии, что на данном уровне не описаны свои директивы *ssl_conf_command*.

Внимание

Изменение настроек OpenSSL напрямую может привести к неожиданному поведению.

ssl_crl

Синтаксис ssl_crl *файл*;

Умолчание —

Контекст stream, server

Указывает файл с отозванными сертификатами (CRL) в формате PEM, используемыми для проверки клиентских сертификатов.

ssl_dhparam

Синтаксис ssl_dhparam *файл*;

Умолчание —

Контекст stream, server

Указывает файл с параметрами для DHE-шифров.

Внимание

По умолчанию параметры не заданы, и соответственно DHE-шифры не будут использоваться.

ssl_ecdh_curve

Синтаксис ssl_ecdh_curve *кривая*;

Умолчание ssl_ecdh_curve auto;

Контекст stream, server

Задаёт кривую для ECDHE-шифров.

Примечание

При использовании OpenSSL 1.0.2 и выше можно указывать несколько кривых, например:

```
ssl_ecdh_curve prime256v1:secp384r1;
```

Специальное значение auto соответствует встроенному в библиотеку OpenSSL списку кривых для OpenSSL 1.0.2 и выше, или *prime256v1* для более старых версий.

Примечание

При использовании OpenSSL 1.0.2 и выше директива задаёт список кривых, поддерживаемых сервером. Поэтому для работы ECDSA-сертификатов важно, чтобы список включал кривые, используемые в сертификатах.

ssl_handshake_timeout

Синтаксис ssl_handshake_timeout *время*;

Умолчание ssl_handshake_timeout 60s;

Контекст stream, server

Задаёт таймаут для завершения операции SSL handshake.

ssl_password_file

Синтаксис ssl_password_file *файл*;

Умолчание —

Контекст stream, server

Задаёт файл с паролями от секретных ключей, где каждый пароль указан на отдельной строке. Пароли применяются по очереди в момент загрузки ключа.

Пример:

```
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;

        # вместо файла можно указать именованный канал
        ssl_password_file /etc/keys/fifo;
        ssl_certificate_key /etc/keys/second.key;
    }
}
```

ssl_prefer_server_ciphers

Синтаксис ssl_prefer_server_ciphers on | off;

Умолчание ssl_prefer_server_ciphers off;

Контекст stream, server

При использовании протоколов SSLv3 и TLS устанавливает приоритет серверных шифров над клиентскими.

ssl_protocols

Синтаксис ssl_protocols [SSLv2] [SSLv3] [TLSv1] [TLSv1.1] [TLSv1.2] [TLSv1.3];

Умолчание ssl_protocols TLSv1 TLSv1.1 TLSv1.2;

Контекст stream, server

Разрешает указанные протоколы.

Примечание

Параметры TLSv1.1 и TLSv1.2 работают только при использовании OpenSSL 1.0.1 и выше.

Параметр TLSv1.3 работает только при использовании OpenSSL 1.1.1 и выше.

ssl_session_cache

Синтаксис ssl_session_cache off | none | [builtin[:*размер*]] [shared:*название*:*размер*];

Умолчание ssl_session_cache none;

Контекст stream, server

Задаёт тип и размеры кэшей для хранения параметров сессий. Тип кэша может быть следующим:

off	жёсткое запрещение использования кэша сессий: Angie явно сообщает клиенту, что сессии не могут использоваться повторно.
none	мягкое запрещение использования кэша сессий: Angie сообщает клиенту, что сессии могут использоваться повторно, но на самом деле не хранит параметры сессии в кэше.
builtin	встроенный в OpenSSL кэш, используется в рамках только одного рабочего процесса. Размер кэша задаётся в сессиях. Если размер не задан, то он равен 20480 сессиям. Использование встроенного кэша может вести к фрагментации памяти.
shared	кэш, разделяемый между всеми рабочими процессами. Размер кэша задаётся в байтах, в 1 мегабайт может поместиться около 4000 сессий. У каждого разделяемого кэша должно быть произвольное название. Кэш с одинаковым названием может использоваться в нескольких серверах. Также он используется для автоматического создания, хранения и периодического обновления ключей TLS session tickets, если они не указаны явно с помощью директивы ssl_session_ticket_key.

Можно использовать одновременно оба типа кэша, например:

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

однако использование только разделяемого кэша без встроенного должно быть более эффективным.

ssl_session_ticket_key

Синтаксис ssl_session_ticket_key *файл*;

Умолчание —

Контекст stream, server

Задаёт файл с секретным ключом, применяемым при шифровании и расшифровании TLS session tickets. Директива необходима, если один и тот же ключ нужно использовать на нескольких серверах. По умолчанию используется случайно сгенерированный ключ.

Если указано несколько ключей, то только первый ключ используется для шифрования TLS session tickets. Это позволяет настроить ротацию ключей, например:

```
ssl_session_ticket_key current.key;
ssl_session_ticket_key previous.key;
```

Файл должен содержать 80 или 48 байт случайных данных и может быть создан следующей командой:

```
openssl rand 80 > ticket.key
```

В зависимости от размера файла для шифрования будет использоваться либо AES256 (для 80-байтных ключей), либо AES128 (для 48-байтных ключей).

ssl_session_tickets

Синтаксис ssl_session_tickets on | off;

Умолчание ssl_session_tickets on;

Контекст stream, server

Разрешает или запрещает возобновление сессий при помощи TLS session tickets.

ssl_session_timeout

Синтаксис ssl_session_timeout *время*;

Умолчание ssl_session_timeout 5m;

Контекст stream, server

Задаёт время, в течение которого клиент может повторно использовать параметры сессии.

ssl_trusted_certificate

Синтаксис ssl_trusted_certificate *файл*;

Умолчание —

Контекст stream, server

Задаёт файл с доверенными сертификатами CA в формате PEM, которые используются для проверки клиентских сертификатов.

В отличие от ssl_client_certificate, список этих сертификатов не будет отправляться клиентам.

ssl_verify_client

Синтаксис ssl_verify_client on | off | optional | optional_no_ca;

Умолчание ssl_verify_client off;

Контекст stream, server

Разрешает проверку клиентских сертификатов. Результат проверки доступен через переменную $ssl_client_verify. Если при проверке клиентского сертификата произошла ошибка или клиент не предоставил требуемый сертификат, соединение закрывается.

optional	запрашивает клиентский сертификат, и если сертификат был предоставлен, проверяет его
optional_no_ca	запрашивает сертификат клиента, но не требует, чтобы он был подписан доверенным сертификатом CA. Это предназначено для случаев, когда фактическая проверка сертификата осуществляется внешним по отношению к Angie сервисом.

ssl_verify_depth

Синтаксис ssl_verify_depth *число*;

Умолчание ssl_verify_depth 1;

Контекст stream, server

Устанавливает глубину проверки в цепочке клиентских сертификатов.

Встроенные переменные

Модуль stream_ssl поддерживает встроенные переменные:

$ssl_alpn_protocol

возвращает протокол, выбранный при помощи ALPN во время операции SSL handshake, либо пустую строку.

$ssl_cipher

возвращает название используемого шифра для установленного SSL-соединения.

$ssl_ciphers

возвращает список шифров, поддерживаемых клиентом. Известные шифры указаны по имени, неизвестные указаны в шестнадцатеричном виде, например:

```
AES128-SHA:AES256-SHA:0x00ff
```

Примечание

Переменная полностью поддерживается при использовании OpenSSL версии 1.0.2 и выше. При использовании более старых версий переменная доступна только для новых сессий и может содержать только известные шифры.

$ssl_client_cert

возвращает клиентский сертификат для установленного SSL-соединения в формате PEM перед каждой строкой которого, кроме первой, вставляется символ табуляции.

$ssl_client_fingerprint

возвращает SHA1-отпечаток клиентского сертификата для установленного SSL-соединения.

$ssl_client_i_dn

возвращает строку “issuer DN” клиентского сертификата для установленного SSL-соединения согласно RFC 2253.

$ssl_client_raw_cert

возвращает клиентский сертификат для установленного SSL-соединения в формате PEM.

$ssl_client_s_dn

возвращает строку “subject DN” клиентского сертификата для установленного SSL-соединения согласно RFC 2253.

$ssl_client_serial

возвращает серийный номер клиентского сертификата для установленного SSL-соединения.

$ssl_client_v_end

возвращает дату окончания срока действия клиентского сертификата.

$ssl_client_v_remain

возвращает число дней, оставшихся до истечения срока действия клиентского сертификата.

$ssl_client_v_start

возвращает дату начала срока действия клиентского сертификата.

$ssl_client_verify

возвращает результат проверки клиентского сертификата: “SUCCESS”, “FAILED:reason” и, если сертификат не был предоставлен, “NONE”.

$ssl_curve

возвращает согласованную кривую, использованную для обмена ключами во время операции SSL handshake. Известные кривые указаны по имени, неизвестные указаны в шестнадцатеричном виде, например:

```
prime256v1
```

Примечание

Переменная поддерживается при использовании OpenSSL версии 3.0 и выше. При использовании более старых версий значением переменной будет пустая строка.

$ssl_curves

возвращает список кривых, поддерживаемых клиентом. Известные кривые указаны по имени, неизвестные указаны в шестнадцатеричном виде, например:

```
0x001d:prime256v1:secp521r1:secp384r1
```

Примечание

Переменная поддерживается при использовании OpenSSL версии 1.0.2 и выше. При использовании более старых версий значением переменной будет пустая строка.

Переменная доступна только для новых сессий.

$ssl_protocol

возвращает протокол установленного SSL-соединения.

$ssl_server_name

возвращает имя сервера, запрошенное через SNI.

$ssl_session_id

возвращает идентификатор сессии установленного SSL-соединения.

$ssl_session_reused

возвращает “*r*”, если сессия была использована повторно, иначе “.”.

Модуль stream_ssl_preread

Позволяет извлекать информацию из сообщения ClientHello без терминирования SSL/TLS, например имя сервера, запрошенное через SNI или протоколы, указанные в ALPN.

По умолчанию этот модуль не собирается, его сборку необходимо разрешить с помощью конфигурационного параметра *--with-stream_ssl_preread_module*.

Пример конфигурации

Выбор сервера по имени

```
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;
}
```

Выбор сервера по протоколу

```
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;
}
```

Выбор сервера по версии протокола SSL

```
map $ssl_preread_protocol $upstream {
    ""        ssh.example.com:22;
    "TLSv1.2" new.example.com:443;
    default   tls.example.com:443;
}

# ssh и https на одном порту
server {
    listen      192.168.0.1:443;
    proxy_pass  $upstream;
    ssl_preread on;
}
```

Директивы

ssl_preread

Синтаксис ssl_preread on | off;

Умолчание ssl_preread off;

Контекст stream, server

Разрешает извлечение информации из сообщения ClientHello во время фазы предварительного чтения.

Встроенные переменные

$ssl_preread_protocol

максимальная версия протокола SSL, поддерживаемая клиентом

$ssl_preread_server_name

имя сервера, запрошенное через SNI

$ssl_preread_alpn_protocols

список протоколов, переданный клиентом через ALPN. Значения разделяются запятыми.

Модуль stream_upstream

Предоставляет контекст для описания группы серверов, которые могут использоваться в директиве proxy_pass.

Пример конфигурации

```
upstream backend {
    hash $remote_addr consistent;

    server backend1.example.com:12345  weight=5;
    server backend2.example.com:12345;
    server unix:/tmp/backend3;

    server backup1.example.com:12345   backup;
    server backup2.example.com:12345   backup;
}

server {
    listen 12346;
    proxy_pass backend;
}
```

Директивы

upstream

Синтаксис upstream *имя* { … }

Умолчание —

Контекст stream

Описывает группу серверов. Серверы могут слушать на разных портах. Кроме того, можно одновременно использовать серверы, слушающие на TCP- и UNIX-сокетах.

Пример:

```
upstream backend {
    server backend1.example.com:12345 weight=5;
    server 127.0.0.1:12345            max_fails=3 fail_timeout=30s;
    server unix:/tmp/backend2;
    server backend3.example.com:12345 resolve;

    server backup1.example.com:12345  backup;
}
```

По умолчанию соединения распределяются по серверам циклически (в режиме round-robin) с учётом весов серверов. В вышеприведённом примере каждые 7 соединений будут распределены так: 5 соединений на backend1.example.com:12345 и по одному соединению на второй и третий серверы.

Если при попытке работы с сервером происходит ошибка, то соединение передаётся следующему серверу, и так далее до тех пор, пока не будут опробованы все работающие серверы. Если связь с серверами не удалась, соединение будет закрыто.

server

Синтаксис server *адрес* [*параметры*];

Умолчание —

Контекст upstream

Задаёт адрес и другие параметры сервера. Адрес может быть указан в виде доменного имени или IP-адреса, и обязательного порта, или в виде пути UNIX-сокета, который указывается после префикса “*unix:*”. Доменное имя, которому соответствует несколько IP-адресов, задаёт сразу несколько серверов.

Могут быть заданы следующие параметры:

weight= *число*	задаёт вес сервера по умолчанию 1.
max_conns= *число*	ограничивает максимальное число одновременных активных соединений к проксируемому серверу. Значение по умолчанию равно 0 и означает, что ограничения нет. Если группа не находится в зоне разделяемой памяти, то ограничение работает отдельно для каждого рабочего процесса.

max_fails= *число* — задаёт число неудачных попыток работы с сервером, которые должны произойти в течение времени, заданного параметром fail_timeout, чтобы сервер считался недоступным на период времени, также заданный параметром fail_timeout.

В данном случае неудачной попыткой считается ошибка или таймаут при установке соединения с сервером.

max_fails=1	число попыток по умолчанию
max_fails=0	отключает учёт попыток

fail_timeout= *время* — задаёт:

- время, в течение которого должно произойти заданное число неудачных попыток работы с сервером для того, чтобы сервер считался недоступным;
- и время, в течение которого сервер будет считаться недоступным. По умолчанию параметр равен 10 секундам.

backup	помечает сервер как запасной. На него будут передаваться запросы в случае, если не работают основные серверы.

down	помечает сервер как постоянно недоступный.

Внимание

Параметр backup нельзя использовать совместно с методами балансировки нагрузки hash и random.

zone

Синтаксис zone *имя* [*размер*];

Умолчание —

Контекст upstream

Задаёт имя и размер зоны разделяемой памяти, в которой хранятся конфигурация группы и её рабочее состояние, разделяемые между рабочими процессами. В одной и той же зоне могут быть сразу несколько групп. В этом случае достаточно указать размер только один раз.

hash

Синтаксис hash *ключ* [consistent];

Умолчание —

Контекст upstream

Задаёт метод балансировки нагрузки для группы, при котором соответствие клиента серверу определяется при помощи хэшированного значения ключа. В качестве ключа может использоваться текст, переменные и их комбинации. Пример использования:

```
hash $remote_addr;
```

Метод совместим с библиотекой Perl Cache::Memcached.

Если задан параметр consistent, то вместо вышеописанного метода будет использоваться метод консистентного хэширования ketama. Метод гарантирует, что при добавлении сервера в группу или его удалении на другие серверы будет перераспределено минимальное число ключей. Применение метода для кэширующих серверов обеспечивает больший процент попаданий в кэш. Метод совместим с библиотекой Perl Cache::Memcached::Fast при значении параметра ketama_points равным 160.

least_conn

Синтаксис least_conn;

Умолчание —

Контекст upstream

Задаёт для группы метод балансировки нагрузки, при котором соединение передаётся серверу с наименьшим числом активных соединений, с учётом весов серверов. Если подходит сразу несколько серверов, они выбираются циклически (в режиме round-robin) с учётом их весов.

random

Синтаксис random [two];

Умолчание —

Контекст upstream

Задаёт для группы метод балансировки нагрузки, при котором соединение передаётся случайно выбранному серверу, с учётом весов серверов.

Если указан необязательный параметр two, Angie случайным образом выбирает два сервера, из которых выбирает сервер, используя указанный метод. Методом по умолчанию является least_conn, при котором соединение передаётся на сервер с наименьшим количеством активных соединений.

Встроенные переменные

Модуль stream_upstream поддерживает следующие встроенные переменные:

$upstream_addr

хранит IP-адрес и порт или путь к UNIX-сокету сервера группы. Если при проксировании были сделаны обращения к нескольким серверам, то их адреса разделяются запятой, например:

```
192.168.1.1:12345, 192.168.1.2:12345, unix:/tmp/sock”
```

Если сервер не может быть выбран, то переменная хранит *имя* группы серверов.

$upstream_bytes_received

число байт, полученных от сервера группы. Значения нескольких соединений разделяются запятыми и двоеточиями подобно адресам в переменной $upstream_addr.

$upstream_bytes_sent

число байт, переданных на сервер группы. Значения нескольких соединений разделяются запятыми и двоеточиями подобно адресам в переменной $upstream_addr.

$upstream_connect_time

хранит время, затраченное на установление соединения с сервером группы; время хранится в секундах с точностью до миллисекунд. Времена нескольких соединений разделяются запятыми и двоеточиями подобно адресам в переменной $upstream_addr.

$upstream_first_byte_time

время получения первого байта данных; время хранится в секундах с точностью до миллисекунд. Времена нескольких соединений разделяются запятыми подобно адресам в переменной $upstream_addr.

$upstream_session_time

длительность сессии в секундах с точностью до миллисекунд. Времена нескольких соединений разделяются запятыми подобно адресам в переменной $upstream_addr.

5.5 Модули mail

Модуль mail_core

Модуль по умолчанию не собирается, его сборку необходимо разрешить с помощью конфигурационного параметра *–with-mail*.

Пример конфигурации

```
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;
    }
}
```

Директивы

listen

Синтаксис listen *адрес*[:*порт*] [ssl] [proxy_protocol] [backlog = *число*] [rcvbuf = *размер*] [sndbuf = *размер*] [bind] [ipv6only=on|off] [reuseport] [so_keepalive=on|off|[keepidle]:[keepintvl]:[keepcnt]];

Умолчание —

Контекст server

Задаёт *адрес* и *порт* для сокета, на котором сервер будет принимать соединения. Можно указать только *порт*. Кроме того, *адрес* может быть именем хоста, например:

```
listen 127.0.0.1:110;
listen *:110;
listen 110;      # то же, что и *:110
listen localhost:110;
```

IPv6-адреса задаются в квадратных скобках:

```
listen [::1]:110;
listen [::]:110;
```

UNIX-сокеты задаются префиксом *unix:*

```
listen unix:/var/run/angie.sock;
```

Внимание

Разные серверы должны слушать на разных парах *адрес:порт*.

ssl	указывает на то, что все соединения, принимаемые на данном слушающем сокете, должны работать в режиме SSL.
proxy_protocol	указывает на то, что все соединения, принимаемые на данном порту, должны использовать протокол PROXY. Полученная информация передаётся серверу аутентификации и может быть использована для изменения адреса клиента.

В директиве *listen* можно также указать несколько дополнительных параметров, специфичных для связанных с сокетами системных вызовов.

backlog= *число*	задаёт параметр *backlog* в вызове *listen()*, который ограничивает максимальный размер очереди ожидающих приёма соединений. По умолчанию *backlog* устанавливается равным *-1* для FreeBSD, DragonFly BSD и macOS, и *511* для других платформ.
rcvbuf= *размер*	задаёт размер буфера приёма (параметр *SO_RCVBUF*) для слушающего сокета.
sndbuf= *размер*	задаёт размер буфера передачи (параметр *SO_SNDBUF*) для слушающего сокета.

bind	указывает, что для данного слушающего сокета нужно делать *bind()* отдельно. Это нужно потому, что если описаны несколько директив *listen* с одинаковым портом, но разными адресами, и одна из директив *listen* слушает на всех адресах для данного *порта* (**:порт*), то Angie сделает *bind()* только на **:порт*. Необходимо заметить, что в этом случае для определения адреса, на который пришло соединение, делается системный вызов *getsockname()*. Если же используются параметры *backlog*, *rcvbuf*, *sndbuf*, *ipv6only*, *reuseport* или *so_keepalive*, то для данной пары *адрес:порт* всегда делается отдельный вызов *bind()*.
ipv6only=on | off	определяет (через параметр сокета *IPV6_V6ONLY*), будет ли слушающий на wildcard-адресе [::] IPv6-сокет принимать только IPv6-соединения, или же одновременно IPv6- и IPv4-соединения. По умолчанию параметр включён. Установить его можно только один раз на старте.

`so_keepalive=on | off | [keepidle]:[keepintvl]:[keepcnt]`

конфигурирует для слушающего сокета поведение “TCP keepalive”.

"	если параметр опущен, для сокета будут действовать настройки операционной системы
on	для сокета включается параметр *SO_KEEPALIVE*
off	для сокета параметр *SO_KEEPALIVE* выключается

Некоторые операционные системы поддерживают настройку параметров “TCP keepalive” на уровне сокета посредством параметров *TCP_KEEPIDLE*, *TCP_KEEPINTVL* и *TCP_KEEPCNT*. На таких системах их можно сконфигурировать с помощью параметров *keepidle*, *keepintvl* и *keepcnt*. Один или два параметра могут быть опущены, в таком случае для соответствующего параметра сокета будут действовать стандартные системные настройки.

Например,

```
so_keepalive=30m::10
```

установит таймаут бездействия (*TCP_KEEPIDLE*) в 30 минут, для интервала проб (*TCP*_KEEPINTVL) будет действовать стандартная системная настройка, а счётчик проб (*TCP_KEEPCNT*) будет равен 10.

Разные серверы должны слушать на разных парах *адрес:порт*.

mail

Синтаксис mail { … }

Умолчание —

Контекст main

Предоставляет контекст конфигурационного файла, в котором указываются директивы почтового сервера.

max_errors

Синтаксис max_errors *число*;

Умолчание max_errors 5;

Контекст mail, server

Задаёт число ошибок протокола, по достижении которого соединение закрывается.

protocol

Синтаксис protocol *imap* | *pop3* | *smtp;*

Умолчание —

Контекст server

Задаёт протокол проксируемого сервера. Поддерживаются протоколы IMAP, POP3 и SMTP.

Если директива не указана, то протокол может быть определён автоматически по общеизвестному порту, указанному в директиве *listen*:

```
imap: 143, 993
pop3: 110, 995
smtp: 25, 587, 465
```

Поддержку ненужных протоколов можно отключить с помощью конфигурационных параметров *--without-mail_imap_module*, *--without-mail_pop3_module* и *--without-mail_smtp_module*.

resolver

Синтаксис resolver *адрес* … [valid= *время*] [ipv4=on|off] [ipv6=on|off] [status_zone= *зона*];

Умолчание resolver off;

Контекст mail, server

Задаёт серверы DNS, используемые для преобразования имени хоста клиента для передачи его на сервер аутентификации и в команде XCLIENT при проксировании SMTP, например:

```
resolver 127.0.0.1 [::1]:5353;
```

Адрес может быть указан в виде доменного имени или IP-адреса, и необязательного порта. Если порт не указан, используется порт 53. Серверы DNS опрашиваются циклически.

По умолчанию Angie кэширует ответы, используя значение TTL из ответа. Необязательный параметр valid позволяет это переопределить:

valid	*необязательный* параметр, позволяет переопределить срок кэширования ответа

```
resolver 127.0.0.1 [::1]:5353 valid=30s;
```

По умолчанию Angie будет искать как IPv4-, так и IPv6-адреса при преобразовании имён в адреса.

ipv4=off	запрещает поиск IPv4-адресов
ipv6=off	запрещает поиск IPv6-адресов
status_zone	*необязательный* параметр, включает сбор информации о запросах и ответах сервера DNS в указанной зоне

Примечание

Для предотвращения DNS-спуфинга рекомендуется использовать DNS-серверы в защищённой доверенной локальной сети.

resolver_timeout

Синтаксис resolver_timeout *время;*

Умолчание resolver_timeout 30s;

Контекст mail, server

Задаёт таймаут для преобразования имени в адрес, например:

```
resolver_timeout 5s;
```

server

Синтаксис server { … }

Умолчание —

Контекст mail

Задаёт конфигурацию для сервера.

server_name

Синтаксис server_name *имя*;

Умолчание server_name hostname;

Контекст mail, server

Задаёт имя сервера, используемое:

- в начальном приветствии POP3/SMTP-сервера;
- в salt при аутентификации SASL-методом CRAM-MD5;
- в команде EHLO при подключении к SMTP-бэкенду, если разрешена передача команды XCLIENT.

Если директива не указана, используется имя хоста (*hostname*) машины.

timeout

Синтаксис timeout *время;*

Умолчание timeout 60s;

Контекст mail, server

Задаёт таймаут, который используется до начала проксирования на бэкенд.

Модуль mail_auth_http

Директивы

auth_http

Синтаксис auth_http URL;

Умолчание —

Контекст mail, server

Задаёт URL HTTP-сервера аутентификации. Протокол описан ниже.

auth_http_header

Синтаксис auth_http_header *заголовок значение*;

Умолчание —

Контекст mail, server

Добавляет указанный заголовок к запросам, посылаемым на сервер аутентификации. Заголовок можно использовать в качестве shared secret для проверки, что запрос поступил от Angie. Например:

```
auth_http_header X-Auth-Key "secret_string";
```

auth_http_pass_client_cert

Синтаксис auth_http_pass_client_cert on | off;

Умолчание auth_http_pass_client_cert off;

Контекст mail, server

Добавляет заголовок “Auth-SSL-Cert” с клиентским сертификатом в формате PEM (закодирован в формате *urlencode*) к запросам, посылаемым на сервер аутентификации.

auth_http_timeout

Синтаксис auth_http_timeout *время*;

Умолчание auth_http_timeout 60s;

Контекст mail, server

Задаёт таймаут общения с сервером аутентификации.

Протокол

Для общения с сервером аутентификации используется протокол HTTP. Данные в теле ответа игнорируются, информация передаётся только в заголовках.

Примеры запросов и ответов:

Запрос:

```
GET /auth HTTP/1.0
Host: localhost
Auth-Method: plain # plain/apop/cram-md5/external
Auth-User: user
Auth-Pass: password
Auth-Protocol: imap # imap/pop3/smtp
Auth-Login-Attempt: 1
Client-IP: 192.0.2.42
Client-Host: client.example.org
```

Хороший ответ:

```
HTTP/1.0 200 OK
Auth-Status: OK
Auth-Server: 198.51.100.1
Auth-Port: 143
Плохой ответ:
HTTP/1.0 200 OK
Auth-Status: Invalid login or password
Auth-Wait: 3
```

Если заголовка “Auth-Wait” нет, то после выдачи ошибки соединение будет закрыто. В текущей реализации на каждую попытку аутентификации выделяется память, которая освобождается только при завершении сессии. Поэтому число неудачных попыток аутентификации в рамках одной сессии должно быть ограничено — после 10-20 попыток (номер попытки передаётся в заголовке “Auth-Login-Attempt”) сервер должен выдать ответ без заголовка “Auth-Wait”.

При использовании APOP или CRAM-MD5 запрос и ответ будут выглядеть так:

```
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
```

Хороший ответ:

```
HTTP/1.0 200 OK
Auth-Status: OK
```

```
Auth-Server: 198.51.100.1
Auth-Port: 143
Auth-Pass: plain-text-pass
```

Если в ответе есть заголовок “Auth-User”, то он переопределяет имя пользователя, используемое для аутентификации с бэкендом.

Для SMTP в ответе дополнительно учитывается заголовок “Auth-Error-Code” — если он есть, то используется как код ответа в случае ошибки. Если его нет, то по умолчанию к “Auth-Status” будет добавлен код 535 5.7.0.

Например, если от сервера аутентификации будет получен ответ:

```
HTTP/1.0 200 OK
Auth-Status: Temporary server problem, try again later
Auth-Error-Code: 451 4.3.0
Auth-Wait: 3
```

то по SMTP клиенту будет выдана ошибка

```
451 4.3.0 Temporary server problem, try again later
```

Если при проксировании SMTP не требуется аутентификация, запрос будет выглядеть так:

```
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>
```

Для клиентского соединения по протоколу SSL/TLS добавляется заголовок “Auth-SSL”, и если директива ssl_verify_client включена, заголовок “Auth-SSL-Verify” содержит результат проверки клиентского сертификата: “*SUCCESS*”, “*FAILED:reason*” и, если сертификат не был предоставлен, “*NONE*”.

Если клиентский сертификат был предоставлен, информация о нём передаётся в следующих заголовках запроса: “Auth-SSL-Subject”, “Auth-SSL-Issuer”, “Auth-SSL-Serial” и “Auth-SSL-Fingerprint”. Если директива auth_http_pass_client_cert включена, сам сертификат передаётся в заголовке “Auth-SSL-Cert”. Протокол и шифр установленного соединения передаются в заголовках “Auth-SSL-Protocol” и “Auth-SSL-Cipher”. Запрос будет выглядеть так:

```
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
```

При использовании протокола PROXY, информация о нём передаётся в следующих заголовках запроса: “Proxy-Protocol-Addr”, “Proxy-Protocol-Port”, “Proxy-Protocol-Server-Addr” и “Proxy-Protocol-Server-Port”.

Модуль mail_proxy

Директивы

proxy_buffer

Синтаксис proxy_buffer *размер*;

Умолчание proxy_buffer 4k|8k;

Контекст mail, server

Задаёт размер буфера, используемого при проксировании. По умолчанию размер одного буфера равен размеру страницы. В зависимости от платформы это или 4K, или 8K.

proxy_pass_error_message

Синтаксис proxy_buffer on | off;

Умолчание proxy_pass_error_message off;

Контекст mail, server

Определяет, передавать ли клиенту сообщение об ошибке, полученное при аутентификации на бэкенде.

Обычно, если аутентификация в Angie прошла успешно, бэкенд не может вернуть ошибку. Если же он всё-таки возвращает ошибку, это значит, что произошла ошибка внутри системы. В таких случаях сообщение бэкенда может содержать информацию, которую нельзя показывать клиенту. Однако для некоторых POP3-серверов ошибка в ответ на правильный пароль является штатным поведением. В этом случае директиву стоит включить.

proxy_protocol

Синтаксис proxy_protocol on | off;

Умолчание proxy_protocol off;

Контекст mail, server

Включает протокол PROXY для соединений с бэкендом.

proxy_smtp_auth

Синтаксис proxy_smtp_auth on | off;

Умолчание proxy_smtp_auth off;

Контекст mail, server

Разрешает или запрещает аутентификацию пользователей на SMTP-бэкенде при помощи команды *AUTH*.

Если также включён XCLIENT, то команда *XCLIENT* не будет отправлять параметр *LOGIN*.

proxy_smtp_auth

Синтаксис proxy_timeout *время*;

Умолчание proxy_timeout 24h;

Контекст mail, server

Задаёт таймаут между двумя идущими подряд операциями чтения или записи на клиентском соединении или соединении с проксируемым сервером. Если по истечении этого времени данные не передавались, соединение закрывается.

xclient

Синтаксис xclient on | off;

Умолчание xclient on;

Контекст mail, server

Разрешает или запрещает передачу команды XCLIENT с параметрами клиента при подключении к SMTP-бэкенду.

При помощи *XCLIENT* MTA может писать в лог информацию о клиенте и применять различные ограничения на основе этих данных.

Если команда *XCLIENT* разрешена, то при подключении к бэкенду Angie посылает ему следующие команды:

- *EHLO* с именем сервера
- *XCLIENT*
- *EHLO* или *HELO*, как её передал клиент

Если найденное по IP-адресу клиента имя указывает на тот же адрес, оно передаётся в параметре *NAME* команды *XCLIENT*. Если имя не может быть найдено, указывает на другой адрес, или не задан resolver, то в параметре *NAME* передаётся *[UNAVAILABLE]*. Если же в процессе поиска имени или адреса произошла ошибка, передаётся *[TEMPUNAVAIL]*.

Если команда *XCLIENT* запрещена, то при подключении к бэкенду Angie передаёт команду *EHLO* с именем сервера, если клиент передал *EHLO*, иначе *HELO* с именем сервера.

Модуль mail_realip

Позволяет менять адрес и необязательный порт клиента на переданные в указанном поле заголовка адрес и порт клиента на переданные в заголовке протокола PROXY. Протокол PROXY должен быть предварительно включён при помощи установки параметра *proxy_protocol* в директиве listen.

Пример конфигурации

```
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;
```

Директивы

set_real_ip_from

Синтаксис set_real_ip_from *адрес* | *CIDR* | unix:;

Умолчание —

Контекст mail, server

Задаёт доверенные адреса, которые передают верный адрес для замены. Если указано специальное значение “*unix:*“, доверенными будут считаться все UNIX-сокеты.

Модуль mail_ssl

Обеспечивает работу почтового прокси-сервера по протоколу SSL/TLS.

По умолчанию этот модуль не собирается, его сборку необходимо разрешить с помощью конфигурационного параметра *--with-mail_ssl_module*.

Примечание

Для сборки и работы этого модуля нужна библиотека OpenSSL.

Пример конфигурации

Для уменьшения загрузки процессора рекомендуется

- установить число рабочих процессов равным числу процессоров,
- включить разделяемый кэш сессий,
- выключить встроенный кэш сессий
- и, возможно, увеличить время жизни сессии (по умолчанию 5 минут):

```
worker_processes auto;

mail {
```

```
    ...

    server {
        listen              993 ssl;

        ssl_protocols       TLSv1 TLSv1.1 TLSv1.2;
        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;

    #    ...
    }
```

Директивы

ssl_certificate

Синтаксис ssl_certificate *файл*;

Умолчание —

Контекст mail, server

Указывает файл с сертификатом в формате PEM для данного сервера. Если вместе с основным сертификатом нужно указать промежуточные, то они должны находиться в этом же файле в следующем порядке — сначала основной сертификат, а затем промежуточные. В этом же файле может находиться секретный ключ в формате PEM.

Директива может быть указана несколько раз для загрузки сертификатов разных типов, например RSA и 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;

#    ...
}
```

Возможность задавать отдельные цепочки сертификатов для разных сертификатов есть только в OpenSSL 1.0.2 и выше. Для более старых версий следует указывать только одну цепочку сертификатов.

Вместо файла можно указать значение “data:*сертификат*“, при котором сертификат загружается без использования промежуточных файлов.

Ненадлежащее использование подобного синтаксиса может быть небезопасно, например данные секретного ключа могут попасть в лог ошибок.

ssl_certificate_key

Синтаксис ssl_certificate_key *файл*;

Умолчание —

Контекст mail, server

Указывает файл с секретным ключом в формате PEM для данного виртуального сервера.

Вместо файла можно указать значение “engine:*имя*:id“, которое загружает ключ с указанным *id* из OpenSSL engine с заданным именем.

Вместо файла можно указать значение “data:*ключ*“, при котором секретный ключ загружается без использования промежуточных файлов. При этом следует учитывать, что ненадлежащее использование подобного синтаксиса может быть небезопасно, например данные секретного ключа могут попасть в лог ошибок.

ssl_ciphers

Синтаксис ssl_ciphers *шифры*;

Умолчание ssl_ciphers HIGH:!aNULL:!MD5;

Контекст mail, server

Описывает разрешённые шифры. Шифры задаются в формате, поддерживаемом библиотекой OpenSSL, например:

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

Полный список можно посмотреть с помощью команды “*openssl ciphers*”.

ssl_client_certificate

Синтаксис ssl_client_certificate *файл*;

Умолчание —

Контекст mail, server

Указывает файл с доверенными сертификатами CA в формате PEM, которые используются для проверки клиентских сертификатов.

Список сертификатов будет отправляться клиентам. Если это нежелательно, можно воспользоваться директивой ssl_trusted_certificate.

ssl_conf_command

Синтаксис ssl_conf_command *имя значение*;

Умолчание —

Контекст mail, server

Задаёт произвольные конфигурационные команды OpenSSL.

Примечание

Директива поддерживается при использовании OpenSSL 1.0.2 и выше.

На одном уровне может быть указано несколько директив *ssl_conf_command*:

```
ssl_conf_command Options PrioritizeChaCha;
ssl_conf_command Ciphersuites TLS_CHACHA20_POLY1305_SHA256;
```

Директивы наследуются с предыдущего уровня конфигурации при условии, что на данном уровне не описаны свои директивы *ssl_conf_command*.

Внимание

Изменение настроек OpenSSL напрямую может привести к неожиданному поведению.

ssl_crl

Синтаксис ssl_crl *файл*;

Умолчание —

Контекст mail, server

Указывает файл с отозванными сертификатами (CRL) в формате PEM, используемыми для проверки клиентских сертификатов.

ssl_dhparam

Синтаксис ssl_dhparam *файл*;

Умолчание —

Контекст mail, server

Указывает файл с параметрами для DHE-шифров.

Внимание

По умолчанию параметры не заданы, и соответственно DHE-шифры не будут использоваться.

ssl_ecdh_curve

Синтаксис ssl_ecdh_curve *кривая*;

Умолчание ssl_ecdh_curve auto;

Контекст mail, server

Задаёт кривую для ECDHE-шифров.

Примечание

При использовании OpenSSL 1.0.2 и выше можно указывать несколько кривых, например:

```
ssl_ecdh_curve prime256v1:secp384r1;
```

Специальное значение auto соответствует встроенному в библиотеку OpenSSL списку кривых для OpenSSL 1.0.2 и выше, или *prime256v1* для более старых версий.

Примечание

При использовании OpenSSL 1.0.2 и выше директива задаёт список кривых, поддерживаемых сервером. Поэтому для работы ECDSA-сертификатов важно, чтобы список включал кривые, используемые в сертификатах.

ssl_password_file

Синтаксис ssl_password_file *файл*;

Умолчание —

Контекст mail, server

Задаёт файл с паролями от секретных ключей, где каждый пароль указан на отдельной строке. Пароли применяются по очереди в момент загрузки ключа.

Пример:

```
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;

        # вместо файла можно указать именованный канал
        ssl_password_file /etc/keys/fifo;
        ssl_certificate_key /etc/keys/second.key;
    }
}
```

ssl_prefer_server_ciphers

Синтаксис ssl_prefer_server_ciphers on | off;

Умолчание ssl_prefer_server_ciphers off;

Контекст mail, server

При использовании протоколов SSLv3 и TLS устанавливает приоритет серверных шифров над клиентскими.

ssl_protocols

Синтаксис ssl_protocols [SSLv2] [SSLv3] [TLSv1] [TLSv1.1] [TLSv1.2] [TLSv1.3];

Умолчание ssl_protocols TLSv1 TLSv1.1 TLSv1.2;

Контекст mail, server

Разрешает указанные протоколы.

Примечание

Параметры TLSv1.1 и TLSv1.2 работают только при использовании OpenSSL 1.0.1 и выше.

Параметр TLSv1.3 работает только при использовании OpenSSL 1.1.1 и выше.

ssl_session_cache

Синтаксис ssl_session_cache off | none | [builtin[:*размер*]] [shared:*название*:*размер*];

Умолчание ssl_session_cache none;

Контекст mail, server

Задаёт тип и размеры кэшей для хранения параметров сессий. Тип кэша может быть следующим:

off	жёсткое запрещение использования кэша сессий: Angie явно сообщает клиенту, что сессии не могут использоваться повторно.
none	мягкое запрещение использования кэша сессий: Angie сообщает клиенту, что сессии могут использоваться повторно, но на самом деле не хранит параметры сессии в кэше.
builtin	встроенный в OpenSSL кэш, используется в рамках только одного рабочего процесса. Размер кэша задаётся в сессиях. Если размер не задан, то он равен 20480 сессиям. Использование встроенного кэша может вести к фрагментации памяти.
shared	кэш, разделяемый между всеми рабочими процессами. Размер кэша задаётся в байтах, в 1 мегабайт может поместиться около 4000 сессий. У каждого разделяемого кэша должно быть произвольное название. Кэш с одинаковым названием может использоваться в нескольких серверах. Также он используется для автоматического создания, хранения и периодического обновления ключей TLS session tickets, если они не указаны явно с помощью директивы ssl_session_ticket_key.

Можно использовать одновременно оба типа кэша, например:

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

однако использование только разделяемого кэша без встроенного должно быть более эффективным.

ssl_session_ticket_key

Синтаксис ssl_session_ticket_key *файл*;

Умолчание —

Контекст mail, server

Задаёт файл с секретным ключом, применяемым при шифровании и расшифровании TLS session tickets. Директива необходима, если один и тот же ключ нужно использовать на нескольких серверах. По умолчанию используется случайно сгенерированный ключ.

Если указано несколько ключей, то только первый ключ используется для шифрования TLS session tickets. Это позволяет настроить ротацию ключей, например:

```
ssl_session_ticket_key current.key;
ssl_session_ticket_key previous.key;
```

Файл должен содержать 80 или 48 байт случайных данных и может быть создан следующей командой:

```
openssl rand 80 > ticket.key
```

В зависимости от размера файла для шифрования будет использоваться либо AES256 (для 80-байтных ключей), либо AES128 (для 48-байтных ключей).

ssl_session_tickets

Синтаксис ssl_session_tickets on | off;

Умолчание ssl_session_tickets on;

Контекст mail, server

Разрешает или запрещает возобновление сессий при помощи TLS session tickets.

ssl_session_timeout

Синтаксис ssl_session_timeout *время*;

Умолчание ssl_session_timeout 5m;

Контекст mail, server

Задаёт время, в течение которого клиент может повторно использовать параметры сессии.

ssl_trusted_certificate

Синтаксис ssl_trusted_certificate *файл*;

Умолчание —

Контекст mail, server

Задаёт файл с доверенными сертификатами CA в формате PEM, которые используются для проверки клиентских сертификатов.

В отличие от ssl_client_certificate, список этих сертификатов не будет отправляться клиентам.

ssl_verify_client

Синтаксис ssl_verify_client on | off | optional | optional_no_ca;

Умолчание ssl_verify_client off;

Контекст mail, server

Разрешает проверку клиентских сертификатов. Результат проверки передаётся в заголовке “Auth-SSL-Verify” в запросе аутентификации. Если при проверке клиентского сертификата произошла ошибка или клиент не предоставил требуемый сертификат, соединение закрывается.

optional	запрашивает клиентский сертификат, и если сертификат был предоставлен, проверяет его
optional_no_ca	запрашивает сертификат клиента, но не требует, чтобы он был подписан доверенным сертификатом CA. Это предназначено для случаев, когда фактическая проверка сертификата осуществляется внешним по отношению к Angie сервисом. Содержимое сертификата доступно в запросах, посылаемых на сервер аутентификации.

ssl_verify_depth

Синтаксис ssl_verify_depth *число*;

Умолчание ssl_verify_depth 1;

Контекст mail, server

Устанавливает глубину проверки в цепочке клиентских сертификатов.

starttls

Синтаксис starttls on | off | only;

Умолчание starttls off;

Контекст mail, server

on	разрешить использование команд STLS для POP3 и STARTTLS для IMAP и SMTP;
off	запретить использование команд STLS и STARTTLS;
only	требовать предварительного перехода на TLS.

Модуль mail_imap

Директивы

imap_auth

Синтаксис imap_auth *метод* …;

Умолчание imap_auth plain;

Контекст mail, server

Задаёт разрешённые методы аутентификации IMAP-клиентов. Поддерживаемые методы:

plain	LOGIN, AUTH=PLAIN
login	AUTH=LOGIN
cram-md5	AUTH=CRAM-MD5. Для работы этого метода пароль должен храниться в незашифрованном виде.
external	AUTH=EXTERNAL

Методы аутентификации с передачей пароля открытым текстом (команда *LOGIN*, *AUTH=PLAIN* и *AUTH=LOGIN*) включены всегда, однако если методы plain и login не указаны, то *AUTH=PLAIN* и *AUTH=LOGIN* не будут автоматически добавляться в imap_capabilities.

imap_capabilities

Синтаксис imap_capabilities *расширение* …;

Умолчание imap_capabilities IMAP4 IMAP4rev1 UIDPLUS;

Контекст mail, server

Позволяет указать список расширений протокола IMAP, выдаваемый клиенту по команде CAPABILITY. В зависимости от значения директивы starttls к этому списку автоматически добавляются методы аутентификации, указанные в директиве imap_auth, и STARTTLS.

В данной директиве имеет смысл указать расширения, поддерживаемые IMAP-бэкендами, на которые проксируются клиенты (если эти расширения относятся к командам, используемым после аутентификации, когда Angie прозрачно проксирует подключение клиента на бэкенд).

imap_client_buffer

Синтаксис imap_client_buffer *размер*;

Умолчание imap_client_buffer 4k|8k;

Контекст mail, server

Задаёт размер буфера для чтения IMAP-команд. По умолчанию размер одного буфера равен размеру страницы. В зависимости от платформы это или 4K, или 8K.

Модуль mail_pop3

Директивы

pop3_auth

Синтаксис pop3_auth *метод* …;

Умолчание ipop3_auth plain;

Контекст mail, server

Задаёт разрешённые методы аутентификации POP3-клиентов. Поддерживаемые методы:

plain	USER/PASS, AUTH PLAIN, AUTH LOGIN
apop	APOP. Для работы этого метода пароль должен храниться в незашифрованном виде.
cram-md5	AUTH=CRAM-MD5. Для работы этого метода пароль должен храниться в незашифрованном виде.
external	AUTH=EXTERNAL

Методы аутентификации с передачей пароля открытым текстом (*USER/PASS*, *AUTH PLAIN* и *AUTH LOGIN*) включены всегда, однако если метод plain не указан, то *AUTH PLAIN* и *AUTH LOGIN* не будут автоматически добавляться в pop3_capabilities.

pop3_capabilities

Синтаксис pop3_capabilities *расширение* …;

Умолчание pop3_capabilities TOP USER UIDL;

Контекст mail, server

Позволяет указать список расширений протокола POP3, выдаваемый клиенту по команде CAPA. В зависимости от значения директивы starttls к этому списку автоматически добавляются методы аутентификации, указанные в директиве pop3_auth (расширение SASL), и STLS.

В данной директиве имеет смысл указать расширения, поддерживаемые POP3-бэкендами, на которые проксируются клиенты (если эти расширения относятся к командам, используемым после аутентификации, когда Angie прозрачно проксирует подключение клиента на бэкенд).

Модуль mail_smtp

Директивы

smtp_auth

Синтаксис smtp_auth *метод* …;

Умолчание smtp_auth plain login;

Контекст mail, server

Задаёт разрешённые методы аутентификации SASL-аутентификации SMTP-клиентов. Поддерживаемые методы:

plain	AUTH PLAIN
login	AUTH LOGIN
cram-md5	AUTH CRAM-MD5. Для работы этого метода пароль должен храниться в незашифрованном виде.
external	AUTH EXTERNAL

none	Аутентификация не требуется

Методы аутентификации с передачей пароля открытым текстом (*AUTH PLAIN* и *AUTH LOGIN*) включены всегда, однако если методы plain и login не указаны, то *AUTH PLAIN* и *AUTH LOGIN* не будут автоматически добавляться в smtp_capabilities.

smtp_capabilities

Синтаксис smtp_capabilities *расширение* …;

Умолчание —

Контекст mail, server

Позволяет указать список расширений протокола SMTP, выдаваемый клиенту в ответе на команду EHLO. В зависимости от значения директивы starttls к этому списку автоматически добавляются методы аутентификации, указанные в директиве smtp_auth, и STARTTLS.

В данной директиве имеет смысл указать расширения, поддерживаемые MTA, на который проксируются клиенты (если эти расширения относятся к командам, используемым после аутентификации, когда Angie прозрачно проксирует подключение клиента на бэкенд).

smtp_client_buffer

Синтаксис smtp_client_buffer *размер*;

Умолчание smtp_client_buffer 4k|8k;

Контекст mail, server

Задаёт размер буфера для чтения SMTP-команд. По умолчанию размер одного буфера равен размеру страницы. В зависимости от платформы это или 4K, или 8K.

smtp_greeting_delay

Синтаксис smtp_greeting_delay *размер*;

Умолчание smtp_greeting_delay 0;

Контекст mail, server

Позволяет задать задержку перед отправкой SMTP-приветствия, чтобы отклонить клиентов, не дожидающихся приветствия до начала отправки SMTP-команд.