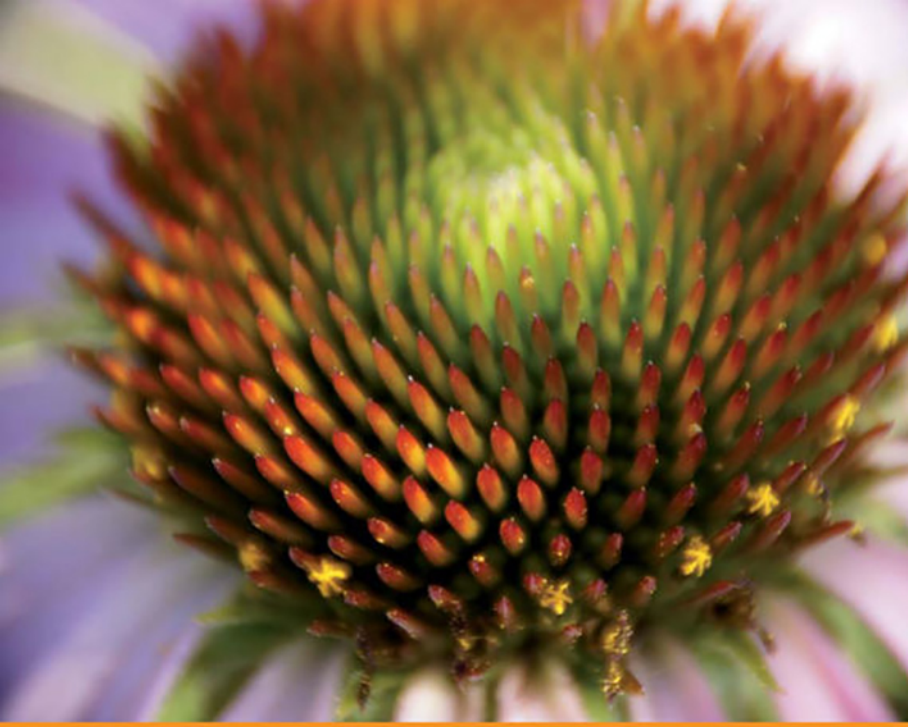


Администрирование сервера NGINX

Подробное руководство по настройке NGINX в любой ситуации, с многочисленными примерами и справочными таблицами для всех директив

Димитрий Айвалиотис

ДМК ПРЕСС
ИЗДАТЕЛЬСТВО

Димитрий Айвалиотис

Администрирование сервера NGINX

Dimitri Aivaliotis

Mastering NGINX

An in-depth guide to configuring NGINX for any situation, including numerous examples and reference tables describing each directive

[PACKT] PUBLISHING open source community experience distilled

Димитрий Айвалиотис

Администрирование сервера NGINX

Подробное руководство по настройке NGINX в любой ситуации, с многочисленными примерами и справочными таблицами для всех директив

Москва, 2013

Содержание

Об авторе

Димитрий Айвалиотис работает системным архитектором в компании, предоставляющей хостинг в Цюрихе, Швейцария. Начав карьеру с построения вычислительной сети на базе Linux для школы, он затем занимался созданием инфраструктуры высокодоступных сдвоенных центров обработки данных для банков и онлайновых порталов. Решая проблемы заказчиков - в течение десяти лет - он открыл для себя NGINX и с тех пор использует эту программу в качестве веб-сервера, прокси-сервера и для организации потоковой передачи мультимедийных данных.

Димитрий с отличием закончил бакалавриат физического факультета Политехнического института Ренсселера, а затем получил степень магистра по информационно-управляющим системам в Университете штата Флорида.

Это его первая книга.

Я благодарен Джону Блэкуэллу и Филу Марголису, прочитавшим ранние варианты рукописи. Их советы и критические замечания оказались очень полезны и позволили сделать книгу лучше. Хочу также поблагодарить технических рецензентов за конструктивную критику и указание на допущенные мной ошибки. Все оставшиеся ошибки - целиком моя вина.

Коллектив издательства Packt Publishing немало способствовал претворению этого проекта в жизнь. Их вера в меня как в писателя не давала мне впасть в отчаяние в тяжкие моменты, когда казалось, что все сроки будут сорваны.

Сотрудники компании NGINX, Inc. помогли заполнить пробелы в моем понимании внутренних механизмов работы NGINX. Без них я бы не смог написать эту кишу

Отдельная благодарность семье. Мои жена и дети вынуждены были мириться с тем, что я тратил немало времени на сочинение книги. Я высоко ценю их терпение на протяжении этого непростого периода.

О рецензентах

Ясир Аднан (Yasir Adnan) живет в столице Бангладеш Дакке. Он изучает информатику и одновременно работает вольнонаемным программистом. Ему доводилось разрабатывать мобильные и веб-приложения, но в настоящее время он занимается в основном мобильными. С ним можно связаться но адресу yasiradnan@outlook.com.

Андрей Алексеев - соучредитель высокотехнологичной компании NGINX, Inc., стоящей за разработкой веб-сервера NGINX. До прихода в NGINX , Inc. в начале 2011 года Андрей работал в Интернет-индустрии в отделах информационно-коммуникационных технологий различных предприятий. Андрей получил диплом инженера-электроника в Санкт-Петербургском государственном электротехническом университете и окончил курс по программе MBA для руководителей в школе менеджмента Университета Антверпена.

Антонио П.П. Альмейда (Antonio P.P. Almeida) (`@perusio`) увлекся NGINX и высокопроизводительными веб-технологиями еще с тех пор, как пытался разрабатывать приложения для Drupal на маломощном ноутбуке на базе процессора Centrino частотой 1,3 ГГц. Из-за прожорливости Apache он был просто вынужден обратиться к NGINX. Он научился выжимать из NGINX максимум возможного в приложениях самых разных типов и в частности освоил все тонкости языка настройки NGINX. Антонио живет в Париже. Помимо NGINX, у него есть и другие пристрастия: малоизвестная музыка позднего итальянского средневековья, кино и желание сделать Drupal еще лучше.

Райнер Даффнер (Rainer Duffner) окончил Университет прикладных наук в Констанце, Германия, по специальности «информационные системы» и в настоящее время работает системным инженером в компании EveryWare AG, где помогает заказчикам извлечь максимум пользы из выделенных серверов па платформах FreeBSD, Linux

и Solaris. Он живет в небольшом городке близ Цюриха и в свободное время катается на горном велосипеде в окрестностях Цюриха и по швейцарским горам.

Я благодарен Димитрию за возможность принять участие в рецензировании этой замечательной книги. Ее ценность невозможно переоценить.

Посвящаю своему отцу, который всегда говорил,
что я могу добиться любой цели, которую поставил перед собой.

Предисловие

NGINX - это высокопроизводительный веб-сервер, потребляющий очень мало системных ресурсов. В Сети немало руководств по его настройке и примеров конфигураций. Задача этой книги - очистить мутные воды конфигурирования NGINX. По ходу дела вы научитесь настраивать NGINX для решения различных задач, узнаете, что означают некоторые покрытые мраком параметры, и поймете, как разработать конфигурацию, отвечающую вашим целям.

Вам больше не потребуется копировать фрагменты найденного где-то конфигурационного скрипта, потому что вы будете знать, как создать файл, делающий в точности то, что нужно. Это умение достигается не сразу, по пути встретится немало ухабов, но благодаря приведенным в этой книге советам вы сможете свободно писать конфигурационные файлы NGINX самостоятельно. Л если что-то пойдет наперекосяк, то сумеете найти причину ошибки сами или по крайней мере попросить помощи без чувства вины за то, что не пытались найти ответ своими силами.

Эта книга построена по модульному принципу - так чтобы максимально облегчить поиск нужной информации. Все главы более-менее независимы. Можете сразу переходить к интересующему вас вопросу. Если складывается ощущение, что пропущено что-то важное, вернитесь назад и прочитайте предшествующие главы. Они организованы так, чтобы можно было строить конфигурационный файл постепенно.

О содержании книги

В главе 1 «Установка NGINX и сторонних модулей» объясняется, как установить NGINX в различных операционных системах и как затем добавить сторонние модули.

В главе 2 «Руководство по настройке» рассказывается о формате конфигурационного файла NGINX. Вы узнаете, для чего предназна-

чены различные контексты, как залапать глобальные параметры и что такое «местоположение».

Глава 3 «Почтовый модуль» посвящена модулю проксирования почты, здесь рассматриваются все аспекты его настройки. Включен пример службы аутентификации.

В главе 4 «NGINX как обратный прокси-сервер» вводится понятие обратного проксирования и описывается использование NGINX в этой роли.

В главе 5 «Обратное проксирование, дополнительные вопросы» более глубоко рассматривается использование NGINX в качестве обратного прокси-сервера для решения проблем масштабирования и производительности.

В главе 6 «NGINX как HTTP-сервер» описывается использование различных модулей, включенных в NGINX для решения типичных задач веб-сервера.

В главе 7 «NGINX для разработчика» показано, как интегрировать NGINX с приложением для ускорения доставки содержимого пользователям.

В главе 8 «Техника устранения неполадок» рассматриваются типичные ошибки при настройке и способы их отладки, а также даются рекомендации но оптимизации производительности.

Приложение A «Справочник директив» содержит удобный справочник по директивам настройки - как описанным в книге, так и ранее не упоминавшимся.

В приложении B «Руководство по правилам переписывания» описано, как работать с модулем переписывания URL в NGINX и приведено несколько простых шагов преобразования правил переписывания из формата Apache в формат NGINX.

Приложение C «Сообщество NGINX» включает перечень сетевых ресурсов, где можно найти дополнительные сведения.

В приложении D «Сохранение сетевых настроек в Solaris» подробно рассказано о том, что необходимо для сохранения изменений различных сетевых параметров в операционной системе Solaris версии 10 и старше.

Что необходимо для чтения этой книги

В каждой главе, где используются примеры кода, приведены инструкции по установке. По существу, требуется следующее.

- **Среда сборки:** компилятор, файлы-заголовки и еще кое-что по мелочи.

- **NGINX:** последняя версия должна подойти.
- **Ruby:** лучше всего взять дистрибутив с сайта `https://rvm.io`.
- **Perl:** стандартная версия годится.

На кого рассчитана эта книга

Книга рассчитана на опытных системных администраторов и системных инженеров, знающих, как производится установка и настройка серверов под конкретные нужды. Предварительное знакомство с NGINX не требуется.

Графические выделения

В этой книге используются различные шрифты для обозначения типа информации. Ниже приведено несколько примеров с пояснениями.

Фрагменты кода внутри абзаца выделяются следующим образом: «NGINX попытается собрать библиотеку статически, если при запуске скрипта `configure` указан параметр `--with-<library>=<path>`».

Кусок кода выглядит так:

```
$ export BUILD_DIR=`pwd`
$ export NGINX_INSTALLDIR=/opt/nginx
$ export VAR_DIR=/home/www/tmp
$ export LUAJIT_LIB=/opt/luajit/lib
$ export LUAJIT_INC=/opt/luajit/include/luajit-2.0
```

Чтобы привлечь внимание к участку внутри куска кода, он выделяется полужирным шрифтом:

```
$ export BUILD_DIR=`pwd`
$ export NGINX_INSTALLDIR=/opt/nginx
$ export VAR_DIR=/home/www/tmp
$ export LUAJIT_LIB=/opt/luajit/lib
$ export LUAJIT_INC=/opt/luajit/include/luajit-2.0
```

Входная и выходная информация командных утилит выглядит так:

```
$ mkdir $HOME/build
$ cd $HOME/build && tar xzf nginx-<version-number>.tar.gz
```

Новые термины и важные фрагменты выделяются полужирным шрифтом. Например, элементы графического интерфейса в меню

или диалоговых окнах выглядят в книге так: «Нажатие кнопки **Next** приводит к переходу на следующий экран».

Предупреждения и важные примечания выглядят так.

Советы и рекомендации выглядят так.

Отзывы

Мы всегда рады отзывам читателей. Расскажите нам, что вы думаете об этой книге - что вам понравилось или, быть может, не понравилось. Читательские отзывы важны для нас, так как помогают выпускать книги, из которых вы черпаете максимум полезного для себя.

Чтобы отправить обычный отзыв, просто пошлите письмо на адрес feedback@packtpub.com, указав название книги в качестве темы.

Если вы являетесь специалистом в некоторой области и хотели бы стать автором или соавтором книги, познакомьтесь с инструкциями для авторов по адресу www.packtpub.com/authors.

Поддержка клиентов

Счастливым обладателям книг Packt мы можем предложить ряд услуг, которые позволят извлечь из своего приобретения максимум пользы.

Загрузка кода примеров

Вы можете скачать код примеров ко всем книгам издательства Packt, купленным на сайте http://www.PacktPub.com. Если книга была куплена в другом месте, зайдите на страницу http://www.PacktPub.com/support, зарегистрируйтесь, и мы отправим файлы по электронной почте.

Опечатки

Мы проверяли содержимое книги со всем тщанием, но какие-то ошибки все же могли проскользнуть. Если вы найдете в нашей книге ошибку, в тексте или в коде, пожалуйста, сообщите нам о ней. Так

вы избавите других читателей от разочарования и поможете нам сделать следующие издания книги лучше. При обнаружении опечатки просьба зайти па страницу http://www.packtpub.com/support, выбрать книгу, щелкнуть по ссылке **errata submission form** и ввести информацию об опечатке. Проверив ваше сообщение, мы поместим информацию об опечатке на нашем сайте или добавим ее в список замеченных опечаток в разделе Errata для данной книги. Список подтвержденных опечаток можно просмотреть, выбрав название книги на странице http://www.packtpub.com/support.

Нарушение авторских прав

Незаконное размещение защищенного авторским правом материала в Интернете - проблема для всех носителей информации. В издательстве Packt мы относимся к защите прав интеллектуальной собственности и лицензированию очень серьезно. Если вы обнаружите незаконные копии наших изданий в любой форме в Интернете, пожалуйста, незамедлительно сообщите нам адрес или название вебсайта, чтобы мы могли предпринять соответствующие меры.

Просим отправить ссылку на вызывающий подозрение в пиратстве материал по адресу copyright@packtpub.com.

Мы будем признательны за помощь в защите прав наших авторов и содействие в наших стараниях предоставлять читателям полезные сведения.

Вопросы

Если вас смущает что-то в этой книге, вы можете связаться с нами по адресу questions@packtpub.com, и мы сделаем все возможное для решения проблемы.

Глава 1. Установка NGINX и сторонних модулей

Первоначально NGINX задумывалась как HTTP-сервер. Она создавалась для решения проблемы C10K, описанной Дэниэлом Кегелем (Daniel Kegel) на странице http://www.kegel.com/c10k.html, - проектирование веб-сервера, способного обрабатывать одновременно 10 000 соединений. NGINX может это делать за счет основанного на событиях механизма обработки соединений и для достижения цели использует зависящий от ОС механизм событий.

Прежде чем переходить к настройке NGINX, ее необходимо установить. В этой главе описывается, как установить саму NGINX и где взять дополнительные модули. NGINX по природе своей является модульной программой, и существует обширное сообщество разработчиков сторонних модулей, расширяющих функциональность основного сервера. Эти модули прикомпилируются к серверу и устанавливаются вместе с ним.

В этой главе:

- Установка NGINX с помощью менеджера пакетов.
- Сборка NGINX из исходного кода.
- Настройка для работы в качестве веб-сервера или почтового сервера.
- Включение модулей.
- Поиск и установка сторонних модулей.
- Полный пример.

Установка NGINX с помощью менеджера пакетов

Для установки достаточно простой команды менеджера пакетов:

- Linux (дистрибутивы на основе Debian) Linux

```
sudo apt-get install nginx
```

- Linux (дистрибутивы на основе rpm)
  ```
  sudo yum install nginx
  ```
- FreeBSD
  ```
  sudo pkg_install -r nginx
  ```

Команда `sudo` применяется для получения привилегий суперпользователя (`'root'`). Если ваша операционная система поддерживает управление доступом на основе ролей (**RBAC - Role-based access control**), то для достижения той же цели следует использовать другую команду, например `pfexec`.

Описанные команды устанавливают NGINX в стандартные места, зависящие от операционной системы. Это предпочтительный метод установки в случае, когда использование пакетов ОС - непреложное требование.

Команда, разрабатывающая ядро NGINX, предлагает также двоичные файлы стабильной версии на странице http://nginx.org/en/download.html. Если в дистрибутиве отсутствует пакет `nginx` (как, например, в CentOS), то можно установить заранее откомпилированные и протестированные двоичные файлы, как описано ниже.

CentOS

Добавьте репозиторий NGINX в конфигурацию `yum`, создав следующий файл:

```
sudo vi /etc/yum.repos.d/nginx.repo

[nginx]
name=nginx repo
baseurl=http://nginx.org/packages/centos/6/$basearch/
gpgcheck=0
enabled=1
```

Затем выполните следующую команду для установки `nginx`:

```
sudo yum install nginx
```

На странице с указанным выше URL имеются также инструкции по установке пакета `nginx-release`.

Debian

Скачайте ключ подписания NGINX - `http://nginx.org/keys/nginx_signing.key` - и добавьте его в связку ключей `apt`:

```
sudo apt-key add nginx_signing.key
```

Добавьте репозиторий `nginx.org` в конец файла `/etc/apt/sources.list`:

```
vi /etc/apt/sources.list
deb http://nginx.org/packages/debian/ squeeze nginx
deb-src http://nginx.org/packages/debian/ squeeze nginx
```

Установите `nginx`, выполнив такие команды:

```
sudo apt-get update
sudo apt-get install nginx
```

Если среди пакетов, прилагаемых к операционной системе, нет `nginx` или версия, включенная в пакет, устарела и не умеет делать того, что вам нужно, или пакеты, размещенные на сайте `nginx.org`, не отвечают вашим потребностям или вы хотите использовать версию NGINX для разработчиков, то остается единственная возможность - собрать NGINX из исходного кода.

Сборка NGINX из исходного кода

Существуют две ветви кода NGINX - стабильная и разрабатываемая. Именно в последней находятся все новые функции, которые интегрируются и тестируются перед включением в стабильную версию. Выпуск разрабатываемой версии сопровождается таким же контролем качества и прогоном функциональных тестов, как и выпуск стабильной версии, поэтому ту и другую можно использовать в производственных системах. Основное различие между ними связано с поддержкой сторонних модулей. В разрабатываемой версии внутренний API может измениться, тогда как в стабильной он фиксирован, поэтому обратная совместимость сторонних модулей гарантируется только для стабильных версий.

Подготовка среды для сборки

Для сборки NGINX из исходного кода система должна отвечать определенным требованиям. Кроме компилятора, понадобятся библиотеки OpenSSL и **PCRE (Perl Compatible Regular Expressions)** и файлы-заголовки для них - если вы хотите иметь поддержку для SSL и модуля переписывания URL соответственно. В некоторых дистрибутивах эти требования удовлетворены изначально. Если же это не так, то вам предстоит найти и установить подходящий пакет либо скачать исходный код, распаковать его в какой-то каталог и сообщить об этом каталоге скрипту конфигурирования NGINX.

NGINX попытается собрать библиотеку статически, если при запуске скрипта `configure` указан параметр `--with-<library>=<path>`. Это бывает полезно, если вы хотите, чтобы NGINX не зависела от того, что установлено в системе, или требуется добиться от двоичного файла `nginx` максимальной производительности. Если используются функции внешних библиотек, доступные лишь начиная с некоторой версии (например, расширение TLS Next Protocol Negotiation - согласование следующего протокола, - появившееся в версии OpenSSL 1.0.1), то указывайте путь к распакованному исходному коду именно этой версии.

Существуют и другие необязательные пакеты для поддержки той или иной функциональности, в том числе алгоритмов хеширования MD5 и SHA-1, библиотеки сжатия zlib и библиотеки libatomic. Алгоритмы хеширования применяются во многих местах NGINX, в частности, для вычисления хеш-кода UR1, играющего роль ключа кэша. Библиотека `zlib` используется для сжатия отправляемого клиенту содержимого. Если доступна библиотека `atomic_ops`, то NGINX будет использовать атомарные операции обновления памяти для реализации быстрого алгоритма блокировки памяти.

Компиляция исходного кода

NGINX можно скачать со страницы http://nginx.org/en/download.html. Там вы найдете исходный код обеих ветвей в форматах `.tar.gz` и `.zip`. Распакуйте архив во временный каталог:

```
$ mkdir $HOME/build
$ cd $HOME/build && tar xzf nginx-<version-number>.tar.gz
```

Произведите конфигурирование, выполнив команду:

```
$ cd $HOHE/build/nginx-<version-number> && ./configure
```

После чего соберите программу:

```
$ make && sudo make install
```

При самостоятельной сборке двоичного файла `nginx` вы можете включить только то, что вам нужно, например, указать, от имени какого пользователя должна работать NGINX, или задать подразумеваемые по умолчанию места расположения журналов, чтобы их не нужно было явно прописывать в конфигурационном файле. В таблице ниже приведены параметры `configure`, которые позволяют изменить способ сборки двоичного файла. Эти параметры не зависят от того, какие модули NGINX подключаются.

Общие параметры `configure`

Параметр	Описание
`--prefix=<path>`	Корень дерева установки. Все остальные пути указываются относительно этого корня
`--sbin-path=<path>`	Путь к двоичному файлу `nginx`. Если не задан, то файл создается в каталоге `prefix`
`--conf-path=<path>`	Путь к каталогу, в котором `nginx` ищет свой конфигурационный файл, если тот не указан в командной строке
`--error-log-path=<path>`	Путь к каталогу, в который `nginx` записывает журнал ошибок, если не задан иным способом
`--pid-path=<path>`	Путь к каталогу, в котором создается pid-файл главного процесса, обычно `/var/run`
`--lock-path=<path>`	Путь к файлу, управляющему взаимоблокировкой разделяемой памяти
`--user=<user>`	Пользователь, от имени которого запускаются рабочие процессы
`--group=<group>`	Пользователь, от имени которой запускаются рабочие процессы
`--with-file-aio`	Включает асинхронный ввод-вывод для FreeBSD 4.3+ и Linux 2.6.22+
`--with-debug`	Включает отладочное протоколирование. Не рекомендуется для производственных систем

Можно также задать оптимизации, отсутствующие в готовом пакете. Ниже перечислены некоторые особенно полезные параметры.

Параметры `configure`, предназначенные для оптимизации

Параметр	Описание
`--with-cc=<path>`	Если требуется использовать компилятор C, путь к которому не входит в переменную окружения `PATH`
`--with-cpp=<path>`	Путь к соответствующему препроцессору
`--with-cc-opt=<options>`	Здесь можно задать путь к необходимым включаемым файлам (передается флагу `-I<path>`), уровень оптимизации (`-O4`) или указать, что нужно компилировать 64-разрядную версию
`--witn-ld-opt=<options>`	Компоновщику передается путь к библиотекам (`-L<path>`) и путь поиска во время выполнения (`-R<path>`)
`--with-cpu-opt=<cpu>`	Позволяет задать сборку для конкретного семейства процессоров

Настройка для работы в качестве веб-сервера или почтового сервера

Среди высокопроизводительных веб-серверов NGINX отличается еще и тем, что проектировалась для работы в качестве почтового прокси-сервера. В зависимости от поставленной задачи NGINX можно настроить для ускорения работы другого веб-сервера, для работы в качестве веб-сервера, для работы в качестве почтового прокси-сервера или для всего сразу. Иногда удобно иметь один пакет, который можно установить на любой сервер внутри организации и задать роль NGINX в конфигурационном файле. А иногда - для работы в высокопроизводительных средах, где каждый килобайт на учете, - лучше подготовить урезанный двоичный файл.

Параметры configure для почтового прокси-сервера

В таблице ниже перечислены параметры configure, имеющие отношение к почтовому модулю.

Параметры `configure`, относящиеся к модулю mail

Параметр	Описание
`--with-mail`	Активируется модуль `mail`, который по умолчанию выключен
`--with-mail_ssl_module`	Этот модуль необходим для проксирования почтовых сообщений, в которых используется `SSL/TLS`

Параметр	Описание
`--without-mail_pop3_module`	При включенном модуле `mail` модуль POP3 можно ОТКЛЮЧИТЬ
`--without-mail_imap_module`	При включенном модуле `mail` модуль IMAP можно ОТКЛЮЧИТЬ
`--without-mail_smtp_module`	При включенном модуле `mail` модуль SMTP можно ОТКЛЮЧИТЬ
`--without-http`	Этот параметр полностью отключает модуль `http`; используйте, только если собираетесь компилировать поддержку `mail`

Для типичного почтового прокси-сервера я рекомендую такие параметры `configure`:

```
$ ./configure --with-mail --with-mail_ssl_module
--with-openssl=$ {BUILD_DIR}/openssl-1.0.1c
```

В настоящее время протокол SSL/TLS необходим практически в любой почтовой системе, отказ от него на прокси-сервере лишит пользователей ожидаемой функциональности. Я советую прикомпилировать OpenSSL статически, чтобы не зависеть от наличия библиотеки OpenSSL в операционной системе. Разумеется, переменную `BUILD_DIR`, упоминаемую в приведенной выше команде, необходимо предварительно определить.

Параметры configure для определения путей

В таблице ниже перечислены параметры `configure`, относящиеся к модулю `http` - от активации модуля Perl до задания путей к временным каталогам.

Параметры `configure`, относящиеся к модулю HTTP

Параметр	Описание
`--without-http-cache`	Если за NGINX стоит другой сервер, то можно настроить NGINX так, чтобы содержимое кэшировалось локально. Этот параметр отключает кэш
`--with-http_perl_module`	Систему конфигурирования NGINX можно расширить, разрешив писать код на Perl. Этот параметр активирует соответствующий модуль (но при этом снижается производительность)

Параметр	Описание
`--with-perl_modules_path=<path>`	Этот параметр определяет путь к дополнительным Perl-модулям, необходимым встроенному интерпретатору Perl. Его можно задать также в конфигурационном файле
`--with-perl=<path>`	Путь к интерпретатору Perl (версии не ниже 5.6.1), если он отсутствует в списке путей по умолчанию
`--http-log-path=<path>`	Подразумеваемый по умолчанию путь к журналу доступа HTTP
`--http-client-body-temp-path=<path>`	В этом каталоге временно сохраняется тело запроса, полученного от клиента. Если включен модуль WebDAV, рекомендуется размещать этот каталог в той же файловой системе, где содержимое будет храниться в конечном итоге
`--http-proxy-temp-path=<path>`	В этом каталоге хранятся временные файлы, когда NGINX используется в качестве прокси-сервера
`--http-fastcgi-temp-path=<path>`	Каталог для временных файлов FastCGI
`--http-uwsgi-temp-path=<path>`	Каталог для временных файлов uWSGI
`--http-scgi-temp-path=<path>`	Каталог для временных файлов SCGI

Включение модулей

Помимо модулей `http` и `mail`, в дистрибутив NGINX включен еще целый ряд модулей. По умолчанию они не активированы, но это можно сделать, указав при запуске `configure` параметр `--with-<имя-модуля>_module`.

Параметры `configure`, относящиеся к прочим модулям

Параметр	Описание
`--with-http_ssl_module`	Если требуется шифровать HTTP-трафик, то есть обрабатывать URL-адреса, начинающиеся с `https`, то необходимо задать этот параметр (при этом необходима библиотека OpenSSL)
`--with-http_realip_module`	Если NGINX расположена за балансировщиком нагрузки L7 или иным устройством, которое передает IP-адрес клиента в HTTP-заголовке, то необходимо включить этот модуль. Он позволяет корректно обрабатывать ситуации, когда разные клиенты по видимости имеют один и тот же IP-адрес

Параметр	Описание
`--with-http_addition_module`	Этот модуль производит преобразование XML-ответов, основываясь на одной или нескольких таблицах стилей XSLT (при этом необходимы библиотеки `libxml2` и `libxslt`)
`--with-http_image_filter_module`	Этот модуль работает как фильтр изображений, обрабатывая их до передачи клиенту (при этом необходима библиотека `libgd`)
`--with-http_geoip_module`	При наличии этого модуля в конфигурационных блоках можно устанавливать различные переменные, которые позволяют принимать решения в зависимости от местоположения клиента, определяемого по его IP-адресу (при этом необходима библиотека MaxMind GeoIP и соответствующие откомпилированные файлы базы данных)
`--with-http_sub_module`	Этот модуль реализует подстановочный фильтр, который заменяет одну присутствующую в ответе строку другой
`--with-http_dav_module`	Если этот модуль включен, то распознаются директивы в конфигурационном файле, относящиеся к использованию WebDAV. Включайте этот модуль только при необходимости, поскольку его неправильная настройка создает угрозы безопасности
`--with-http_flv_module`	Этот модуль обеспечивает псевдопотоковую передачу видеофайлов в формате Flash
`--with-http_mp4_module`	Этот модуль обеспечивает псевдопотоковую передачу видеофайлов в формате H.264/AAC
`--with-http_gzip_static_module`	Этот модуль поддерживает передачу предварительно сжатых статических файлов для ресурсов, URL-адреса которых задаются без указания суффикса `.gz`
`--with-http_gunzip_static_module`	Этот модуль реализует распаковку сжатого содержимого для клиентов, которые не поддерживают сжатие в формате gzip
`--with-http_random_index_module`	Этот модуль следует включать, если вы хотите возвращать индексный файл, случайно выбранный среди всех файлов в некотором каталоге
`--with-http_secure_link_module`	Этот модуль реализует механизм хеширования ссылки на URL-адрес, так что вычислить реальную ссылку сможет только клиент, знающий правильный пароль
`--with-http_stub_status_module`	Этот модуль позволит собирать статистику от самой NGINX. Полученную информацию можно представить в графическом виде с помощью пакета RRDtool или аналогичной программы

Как видите, все эти модули дополняют функциональность модуля `http`. Само их включение на этапе компиляции никак не отражается на производительности во время выполнения. Снижение возможно, только если эти модули упомянуты в конфигурационном файле.

Для ускорения и проксирования веб-сервера я рекомендую задавать такие параметры `configure`:

```
$ ./configure --with-http_ssl_module --with-http_realip_module
--with-http_geoip_module --with-http_stub_status_module
--with-openssl=${BUILD_ DIR}/openssl-1.0.1c
```

А для работы в качестве веб-сервера - такие:

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

Различие в том, «каким боком» NGINX обращена к клиенту. В роли веб-ускорителя NGINX является также оконечной точкой для SSL-запросов и обрабатывает запросы от клиентов, находящихся за прокси-сервером, определяя, откуда в действительности поступил запрос и принимая соответствующие решения. В роли веб-сервера необходима только стандартная функциональность обслуживания запросов на файлы.

Я рекомендую всегда включать модуль `stub_status`, поскольку он позволяет получать статистические данные о работе NGINX.

Отключение неиспользуемых модулей

Существует также ряд относящихся к `http` модулей, которые по умолчанию включены, но могут быть деактивированы путем задания параметра `--without-<имя-модуля>_module`. Если вам эти модули не нужны, можете со спокойной душой отключить их.

Параметры `configure`, относящиеся к модулю HTTP

Параметр	Описание
`--without-http_charset_module`	Модуль `charset` отвечает за установку заголовка `Content-Type` в ответе, а также за преобразование из одной кодировки в другую
`--without-http_gzip_module`	Модуль `gzip` работает как выходной фильтр, сжимая отправляемые клиенту данные
`--without-http_ssi_module`	Этот модуль обрабатывает запросы с включением на стороне сервера (Server Side Include). Если включен модуль Perl, то становится доступна дополнительная команда SSI - `perl`

Параметр	Описание
`--without-http_userid_module`	Модуль `userid` позволяет NGINX устанавливать куки, служащие для идентификации клиента. Впоследствии для отслеживания действий клиента можно использовать переменные `$uid_set` и `$uid_got`
`--without-http_access_module`	Этот модуль контролирует доступ к ресурсам, основываясь на IP-адресе клиента
`--without-http_auth_basic_module`	Этот модуль ограничивает доступ с помощью простой схемы аутентификации (HTTP Basic Authentication)
`--without-http_autoindex_module`	Модуль `autoindex` наделяет NGINX способностью генерировать страницу с содержимым каталогов, в которых нет индексного файла
`--without-http_geo_module`	Этот модуль позволяет устанавливать значения конфигурационных переменных, исходя из IP-адреса клиента, а затем принимать на этой основе те или иные решения
`--without-http_map_module`	Модуль `map` позволяет сопоставлять одной переменной другую
`--without-http_split_clients_module`	Этот модуль создает переменные, которые можно использовать для проведения A/B-тестирования
`--without-http_referer_module`	Этот модуль позволяет NGINX блокировать запросы, исходя из HTTP-заголовка Referer
`--without-http_rewrite_module`	Модуль `rewrite` позволяет заменять URI, исходя из различных условий
`--without-http_proxy_module`	Модуль `proxy` позволяет NGINX передавать запросы другому серверу или группе серверов
`--without-http_fastcgi_module`	Этот модуль позволяет NGINX передавать запросы серверу FastCGI
`--without-http_uwsgi_module`	Этот модуль позволяет NGINX передавать запросы серверу uWSGI
`--without-http_scgi_module`	Этот модуль позволяет NGINX передавать запросы серверу SCGI
`--without-http_memcached_module`	Этот модуль позволяет NGINX взаимодействовать с сервером memcached, сохраняя ответы на запросы в переменной
`--without-http_limit_conn_module`	Этот модуль позволяет NGINX устанавливать ограничения на количество соединений, основываясь на каких-то условиях, чаще всего на IP-адресе
`--without-http_limit_req_module`	Этот модуль позволяет NGINX ограничить частоту запросов, ассоциированных с каким-то условием

Параметр	Описание
`--without-http_empty_gif_module`	Этот модуль порождает прозрачное изображение в формате GIF размером 1 x 1 пиксель
`--without-http_browser_module`	Этот модуль позволяет определять настройки в зависимости от заголовка `User-Agent` в запросе. В результате анализа номера версии в этом заголовке устанавливаются определенные переменные
`--without-http_upstream_ip_hash_module`	Этот модуль определяет группу серверов, которые можно использовать в сочетании с различными модулями проксирования

Поиск и установка сторонних модулей

Как и во многих проектах с открытым исходным кодом, вокруг NGINX сформировалось активное сообщество разработчиков. Благодаря модульной структуре NGINX сообщество имеет возможность разрабатывать и публиковать модули, обеспечивающие дополнительную функциональность. Таких модулей немало, поэтому прежде чем приступать к разработке собственного, имеет смысл познакомиться с тем, что есть.

Процедура установки стороннего модуля очень проста.

1. Найдите интересующий вас модуль (на сайте https://github.com или http://wiki.nginx.org/3rdPartyModules).
2. Скачайте модуль.
3. Распакуйте архив с исходным кодом.
A. Прочитайте файл README, если он имеется. Проверьте, ист ли дополнительных зависимостей.
5. Сконфигурируйте NGINX, так чтобы она использовала модуль `./configure -add-module=<path>`.

В результате получится двоичный файл `nginx`, в который включена функциональность добавленного модуля.

Помните, что многие сторонние модули носят экспериментальный характер. Перед тем как использовать модуль в производственной системе, протестируйте его. И не забывайте, что в разрабатываемые версии NGINX могут быть внесены такие изменения API, что сторонние модули перестанут работать.

Особо отметим модуль `ngx_lua`, который позволяет использовать в качестве скриптового языка конфигурирования Lua вместо Perl. По сравнению с `perl` у этого модуля есть важное достоинство: он не

блокирует выполнение программы и тесно интегрирован с другими сторонними модулями. Полная инструкция по установке приведена на странице http://wiki.nginx.org/HttpLuaModule#Installation. В следующем разделе мы на примере этого модуля продемонстрируем процедуру установки стороннего модуля.

Полный пример

Составив представление о том, что можно сконфигурировать, вы можете построить двоичный файл, точно отвечающий вашим потребностям. В примере ниже задается префикс путей, пользователь, группа, некоторые пути, отключаются одни модули, включаются другие и добавляются два сторонних модуля.

```
$ export BUILD_DIR=`pwd`
$ export NGINX_INSTALLDIR=/opt/nginx
$ export VAR_DIR=/home/www/tmp
$ export LUAJIT_LIB=/opt/luajit/lib
$ ./configure \
--prefix=${NGINX_INSTALLDIR} \
     --user=www \
     --group=www \
     --http-client-body-temp-path=${VAR_DIR}/client_body_temp \
     --http-proxy-temp-path=${VAR_DIR}/proxy_temp \
     --http-fastcgi-temp-path=${VAR_DIR}/fastcgi_temp \
     --without-http_uwsgi_module \
     --without-http_scgi_module \
     --without-http_browser_module \
     --with-openssl=${BUILD_DIR}/../openssl-1.0.1c \
     --with-pcre=${BUILD_DIR}/../pcre-8.32 \
     --with-http_ssl_module \
     --with-http_realip_module \
     --with-http_sub_module \
     --with-http_flv_module \
     --with-http_gzip_static_module \
     --with-http_gunzip_module \
     --with-http_secure_link_module \
     --with-http_stub_status_module \
     --add-module=${BUILD_DIR}/ngx_devel_kit-0.2.17 \
     --add-module=${BUILD_DIR}/ngx_lua-0.7.9
```

После многочисленных сообщений о том, что скрипт `configure` нашел в вашей системе, печатается такая сводка:

Configuration summary

```
+ using PCRE library: /home/builder/build/pcre-8.32
+ using OpenSSL library: /home/builder/build/openssl-1.0.1c
+ md5: using OpenSSL library
+ sha1: using OpenSSL library
+ using system zlib library
nginx path prefix: "/opt/nginx"
nginx binary file: "/opt/nginx/sbin/nginx"
nginx configuration prefix: "/opt/nginx/conf"
nginx configuration file: "/opt/nginx/conf/nginx.conf"
nginx pid file: "/opt/nginx/logs/nginx.pid"
nginx error log file: "/opt/nginx/logs/error.log"
nginx http access log file: "/opt/nginx/logs/access.log"
nginx http client request body temporary files: "/home/www/tmp/
client_body_temp"
nginx http proxy temporary files: "/home/www/tmp/proxy_temp"
nginx http fastcgi temporary files: "/home/www/tmp/fastcgi_temp"
```

Как видим, `configure` нашел все, что мы просили, и подтвердил указанные пути. Теперь можно собрать и установить `nginx`, как объяснялось в начале этой главы.

Резюме

В этой главе мы познакомились с различными модулями NGINX. Собирая двоичный файл самостоятельно, вы можете включить только ту функциональность, которая вам необходима. Сборка и установка программ - дело, вам знакомое, поэтому вы не потратите много времени на подготовку среды сборки и предварительную установку зависимостей. Подгоняя NGINX под свои нужды, не стесняйтесь включать или выключать те или иные модули.

Далее мы дадим общий обзор настройки NGINX, чтобы вы поняли, как подходить к этой задаче.

Глава 2. Руководство по настройке

Формат конфигурационного файла NGINX прост и логичен. Понимание его устройства и назначения отдельных секций - одно из условий самостоятельного составления конфигурационных файлов. В этой главе мы постараемся достичь этой цели, рассмотрев следующие вопросы.

- Основы формата конфигурационного файла.
- Глобальные конфигурационные параметры NGINX.
- Включаемые файлы.
- Секция с описанием HTTP-сервера.
- Секция с описанием виртуального сервера.
- Местоположения - где, когда и как.
- Секция с описанием почтового сервера.
- Полный пример конфигурации.

Основы формата конфигурационного файла

Конфигурационный файл NGINX состоит из секций. Секции устроены следующим образом:

```
<секция> {

    <директива> <параметры>;

    }
```

Обратите внимание, что строки, содержащие директивы, оканчиваются точкой с запятой (`;`). Это признак конца строки. Фигурные скобки вводят новый конфигурационный контекст, но мы будем называть такие контексты просто «секциями».

Глобальные конфигурационные параметры NGINX

В глобальной секции задаются параметры, оказывающие влияние на сервер в целом. Его формат отличается от описанного выше. Глобальная секция может включать как конфигурационные директивы, например `user` и `worker_processes`, так и секции, например `events`. Глобальная секция не заключается в фигурные скобки.

В таблице ниже приведены наиболее распространенные директивы, задаваемые в глобальном контексте.

Глобальные конфигурационные директивы

Директива	Описание
`user`	Пользователь и группа, от имени которых исполняются рабочие процессы. Если группа опущена, то подразумевается группа, имя которой совпадает с именем пользователя
`worker_processes`	Количество рабочих процессов, создаваемых сразу после запуска. Эти процессы обрабатывают запросы на соединения со стороны клиентов. Сколько процессов задать, зависит от сервера и в первую очередь от дисковой подсистемы и сетевой инфраструктуры. Если решаются в основном счетные задачи, то рекомендуется задавать этот параметр равным количеству процессорных ядер, а если задачи, требующие интенсивного ввода-вывода, - то умножать это количество на коэффициент от 1,5 до 2
`error_log`	Это файл, в который записываются сообщения об ошибках. Если ни в каком другом контексте директивы `error_log` нет, то в этом файле будут регистрироваться вообще все ошибки. Второй параметр директивы обозначает уровень сообщений, попадающих в журнал (`debug`, `info`, `notice`, `warn` `error`, `crit`, `alert`, `emerg`). Сообщения уровня debug выводятся, только если программа была сконфигурирована с параметром `--with-debug`
`pid`	Файл, в котором хранится идентификатор главного процесса. Переопределяет значение, заданное на этапе конфигурирования и компиляции
`use`	Определяет метод обработки соединения. Переопределяет значение, заданное при компиляции, и, если используется, то должна содержаться в контексте `events`. Обычно не нуждается в переопределении, разве что в случае, когда значение по умолчанию приводит к ошибкам

Директива	Описание
worker_connections	Эта директива задает максимальное число соединений, одновременно открытых в одном рабочем процессе. Сюда входят в частности соединения с клиентами и с проксируемыми серверами (но не только). Особенно важно это для обратных прокси-серверов - чтобы достичь указанного количества одновременно открытых соединений, может понадобиться настройка на уровне операционной системы

Ниже приведен простой пример задания описанных директив.

```
# мы хотим, чтобы nginx работала от имени пользователя 'www'
user www;

# рабочая нагрузка счетная и имеется 12 процессорных ядер
worker_processes 12;

# явно задаем путь к обязательному журналу ошибок
error_log /var/log/nginx/error.log;

# явно задаем путь к pid-файлу
pid /var/run/nginx.pid.

# создаем конфигурационный контекст для модуля 'events'
events {

    # мы работаем в системе Solaris и обнаружили, что при использовании
    # подразумеваемого по умолчанию механизма обработки соединений nginx
    # со временем перестает отвечать на запросы, поэтому переходим на
    # следующий по качеству механизм
    use /dev/poll;

    # произведение этого числа и значения worker_processes
    # показывает, сколько может быть одновременно открыто соединений
    # для одной пары IP:порт
    worker_connections 2048;
}
```

Глобальная секция должна находиться в начале конфигурационного файла `nginx.conf`.

Включаемые файлы

Включать файлы можно в любое место конфигурационного файла. Их цель - сделать файл более удобным для восприятия и обеспечить повторное использование некоторых частей. Включаемые файлы

должны быть синтаксически корректны с точки зрения записи директив и блоков. Для включения файла нужно указать путь к нему:

```
include /opt/local/etc/nginx/mime.types;
```

Метасимволы в пути позволяют включить сразу несколько файлов:

```
include /opt/local/etc/nginx/vhost/*.conf;
```

Если указан не полный путь, то NGINX считает, что путь задан относительно главного местоположения конфигурационного файла.

Проверить правильность конфигурационного файла можно, запустив NGINX следующим образом:

```
nginx -t -c <path-to-nginx.conf>
```

При этом на наличие ошибок проверяются и все включаемые файлы.

Секция с описанием HTTP-сервера

Секция (или конфигурационный контекст), описывающая HTTP-сервер, доступна, только если при конфигурировании NGINX не был задан параметр `--without-http`, отключающий модуль HTTP. В этой секции описываются все аспекты работы с модулем HTTP - именно с ним вы чаще всего будете иметь дело.

Поскольку конфигурационных директив, относящихся к работе с HTTP-соединениями, много, мы разобьем их на несколько категорий и будем рассматривать каждую категорию в отдельности.

Клиентские директивы

Директивы из этой категории относятся к самому соединению с клиентом, а также описывают некоторые аспекты поведения для клиентов разных типов.

Клиентские директивы модуля HTTP

Директива	Описание
`chunked_transfer_encoding`	Позволяет отключить специфицированный в стандарте HTTP/1.1 механизм поблочной передачи данных (chunked transfer encoding) в ответе клиенту

Директива	Описание
client_body_buffer_size	Задает размер буфера для чтения тела запроса клиента. По умолчанию для буфера выделяется две страницы памяти. Увеличение размера позволяет предотвратить запись во временный файл на диске
client_body_in_file_only	Используется для отладки или последующей обработки тела запроса клиента. Если значение равно on, то тело запроса принудительно записывается в файл
client_body_in_single_buffer	Заставляет NGINX сохранить все тело запроса клиента в одном буфере, чтобы уменьшить количество операций копирования
client_body_temp_path	Определяет путь к каталогу для сохранения файлов с телами запросов клиентов
client_body_timeout	Задает время между последовательными операциями чтения тела запроса клиента
client_header_buffer_size	Задает размер буфера для чтения заголовка запроса клиента, если он превышает подразумеваемую по умолчанию величину 1 КБ
client_header_timeout	Время, отведенное на чтение всего заголовка запроса
client_max_body_size	Максимальный размер тела запроса клиента. В случае превышения отправляется ответ 413 (Request Entity Too Large)
keepalive_disable	Запрещает соединения типа keep-alive для некоторых браузеров
keepalive_requests	Определяет, сколько запросов можно принять по одному соединению типа keep-alive, прежде чем закрывать его
keepalive_timeout	Определяет, сколько времени соединение типа keep-alive может оставаться открытым. Можно задать второй параметр, используемый для формирования заголовка ответа «Keep-Alive»
large_client_header_buffers	Задает максимальное число и размер буферов для чтения большого заголовка запроса клиента
msie_padding	Разрешает или запрещает добавлять комментарии в ответы со статусом больше 400 для увеличения размера ответа до 512 байт при работе с MSIE
msie_refresh	Разрешает или запрещает отправлять MSIE-клиентам ответ Refresh вместо перенаправления

Директивы, относящиеся к вводу-выводу

Эти директивы управляют отправкой статических файлов и порядком работы с файловыми дескрипторами.

Директивы модуля HTTP, относящиеся к вводу-выводу

Директива	Описание
`aio`	Разрешает использование асинхронного файлового ввода-вывода. Это возможно во всех современных версиях FreeBSD и дистрибутивах Linux. В FreeBSD директиву `aio` можно использовать для предварительной загрузки данных для `sendfile`. В Linux требуется директива `directio`, которая автоматически отключает `sendfile`
`directio`	Разрешает использовать зависящий от операционной системы флаг при чтении файлов, размер которых больше или равен указанному. Обязательна при использовании директивы `aio` в Linux
`directio_alignment`	Устанавливает выравнивание для `directio`. Обычно подразумеваемого по умолчанию значения `512` достаточно, но при использовании XFS в Linux рекомендуется увеличить до 4 К
`open_file_cache`	Настраивает кэш, в котором могут храниться дескрипторы открытых файлов, информация о существовании каталогов и информация об ошибках поиска файлов
`open_file_cache_errors`	Разрешает или запрещает кэширование ошибок поиска файлов в кэше `open file cache`
`open_file_cache_min_uses`	Задаёт минимальное число обращений к файлу в течение времени, заданного параметром `inactive` директивы `open_file_cache`, необходимое для того, чтобы дескриптор файла оставался в кэше открытых дескрипторов
`open_file_cache_valid`	Задает время между последовательными проверками актуальности данных, хранящихся в кэше open_file_cache
`postpone_output`	Задает минимальный размер порции данных, отправляемых клиенту. Если возможно, данные не будут отправляться, пока не накопится указанное количество
`read_ahead`	Если возможно, ядро будет сразу считывать из файла столько байтов, сколько указано в параметре `size`. Поддерживается в текущих версиях FreeBSD и Linux (в Linux параметр `size` игнорируется)

Директива	Описание
sendfile	Разрешает использовать системный вызов sendfile (2) для прямого копирования из одного файлового дескриптора в другой
sendfile_max_chunk	Задает максимальный размер данных, который можно скопировать за один вызов sendfile (2). Без этого ограничения одно быстрое соединение может целиком захватить рабочий процесс

Директивы, относящиеся к хеш-таблицам

Директивы из этой категории управляют выделением статической памяти для определенных переменных. NGINX вычисляет минимально необходимый размер при запуске и после изменения конфигурации. Как правило, достаточно настроить только один из параметров *_hash_max_size с помощью соответствующей директивы. NGINX выдает предупреждение при попытке задать сразу несколько таких параметров. Переменным вида *_hash_bucket_size по умолчанию присваивается значение, кратное размеру строки кэша процессора, чтобы минимизировать количество обращений к кэшу, необходимое для чтения записи. Поэтому изменять их не рекомендуется. Дополнительные сведения см. на странице http://ngmx.org/en/docs/hash.html.

Директивы модуля HTTP, относящиеся к хеш-таблицам

Директива	Описание
server_names_hash_bucket_size	Задает размер кластера в хеш-таблицах имен серверов
server_names_hash_max_size	Задает максимальный размер хеш-таблиц имен серверов
types_hash_bucket_size	Задает размер кластера в хеш-таблицах типов
types_names_hash_max_size	Задает максимальный размер хеш-таблиц типов
variables_hash_bucket_size	Задает размер кластера в хеш-таблицах прочих переменных
variables_names_hash_max_size	Задает максимальный размер хеш-таблиц прочих переменных

Директивы, относящиеся к сокетам

Эти директивы описывают установку различных параметров TCP-сокетов, создаваемых NGINX.

Директивы модуля HTTP, относящиеся к сокетам

Директива	Описание
`lingering_close`	Определяет, следует ли оставлять соединение открытым в ожидании дополнительных данных от клиента
`lingering_time`	Связана с директивой `lingering_close` и определяет, сколько времени держать сокет открытым для обработки дополнительных данных
`lingering_timeout`	Также связана с директивой `lingering_close` и определяет, сколько времени держать соединение открытым в ожидании дополнительных данных
`reset_timedout_connection`	Если значение этой директивы равно `on`, то сокеты, для которых истек таймаут, сбрасываются немедленно, в результате чего освобождается выделенная для них память. По умолчанию сокет остается в состоянии `FIN_WAIT1`. На соединения типа keep-alive эта директива не распространяется, они всегда закрываются обычным образом
`send_lowat`	Если значение отлично от нуля, то NGINX пытается минимизировать количество операций отправки данных через клиентские сокеты. В Linux, Solaris и Windows эта директива игнорируется
`send_timeout`	Задает таймаут между двумя последовательными операциями записи при передаче ответа клиенту
`tcp_nodelay`	Разрешает или запрещает использование параметра `TCP_NODELAY` для соединений типа keep-alive
`tcp_nopush`	Учитывается только при использовании директивы `sendfile`. Разрешает NGINX отправлять заголовки ответа одним пакетом, а также передавать файл полными пакетами

Пример конфигурации

Ниже приведен пример конфигурационной секции модуля HTTP:

```
http {
  include /opt/local/etc/nginx/mime.types;
  default_type application/octet-stream;
```

```
sendfile on;
tcp_nopush on;
tcp_nodelay on;
keepalive_timeout 65;
server_names_hash_max_size 1024;
}
```

Этот контекстный блок должен располагаться после всех глобальных директив в файле `nginx.conf`.

Секция с описанием виртуального сервера

По соглашению в контексте, начинающемся ключевым словом `server`, находится описание «виртуального сервера». Так называется логический набор ресурсов, сопоставленный со значением директивы `server_name`. Виртуальные серверы отвечают на запросы по протоколу HTTP и потому входят в состав секции `http`.

Виртуальный сервер определяется сочетанием директив `listen` и `server_name`. Директива `listen` задает комбинацию IP-адреса и номера порта либо путь к сокету в домене UNIX:

```
listen address[:port];
listen port;
listen unix:path;
```

Директива `listen` однозначно описывает привязку совета в NGINX. Дополнительно в ней могут присутствовать следующие параметры.

Параметры директивы `listen`

Параметр	Описание	Примечание
`default_server`	Означает, что данный сервер будет сервером по умолчанию для указанной пары `адрес:порт`	
`setfib`	Задает таблицу маршрутизации FIB для прослушивающего сокета	Поддерживается только в ОС FreeBSD. Игнорируется для сокетов в домене UNIX
`backlog`	Задает параметр backlog в системном вызове listen()	По умолчанию равен -1 в FreeBSD и 511 на всех остальных платформах
`rcvbuf`	Задает параметр `SO_RCVBUF` для прослушивающего сокета	

Параметр	Описание	Примечание
sndbuf	Задает параметр SO_SNDBUF для прослушивающего сокета	
accept _filter	Задает имя фильтра приема: dataready или httpready	Поддерживается только в ОС FreeBSD
deferred	Задает параметр TCP_DEFER_ACCEPT, означающий, что требуется отложить вызов accept()	Поддерживается только в ОС Linux
bind	Означает, что для данной пары адрес:порт нужен отдельный вызов bind()	Отдельный вызов bind() производится без специального указания, если для сокета заданы какие-то другие специальные параметры
ipv6only	Устанавливает значение параметра IPV6_V6ONLY	Можно установить только один раз при запуске. Игнорируется для сокетов в домене UNIX
ssl	Означает, что этот порт предназначен только для соединений по протоколу HTTPS	Позволяет построить более компактную конфигурацию
so_keepalive	Для прослушивающего сокета задается режим TCP keepalive	

Несмотря на свою простоту, директива server_name позволяет решить целый ряд задач конфигурирования. По умолчанию ее значение равно "", то есть секция server без директивы server_name сопоставляется с запросом, в котором отсутствует заголовок Host. Этим можно воспользоваться, например, для отбрасывания запросов без этого заголовка:

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

Нестандартный код ответа HTTP 444, использованный в этом примере, заставляет NGINX немедленно закрыть соединение.

Помимо обычной строки, NGINX допускает использование в директиве server_name метасимвола *:

- Метасимвол можно указывать вместо поддомена: `*.example.com`.
- Метасимвол можно указывать вместо домена верхнего уровня: `www.example.*`.
- Существует особая форма, которая соответствует поддомену или самому домену: `.example.com` (соответствует как `*.example.com`, так и `example.com`).

Параметр директивы `server_name` может быть и регулярным выражением, для этого нужно лишь предпослать имени знак тильды (~):

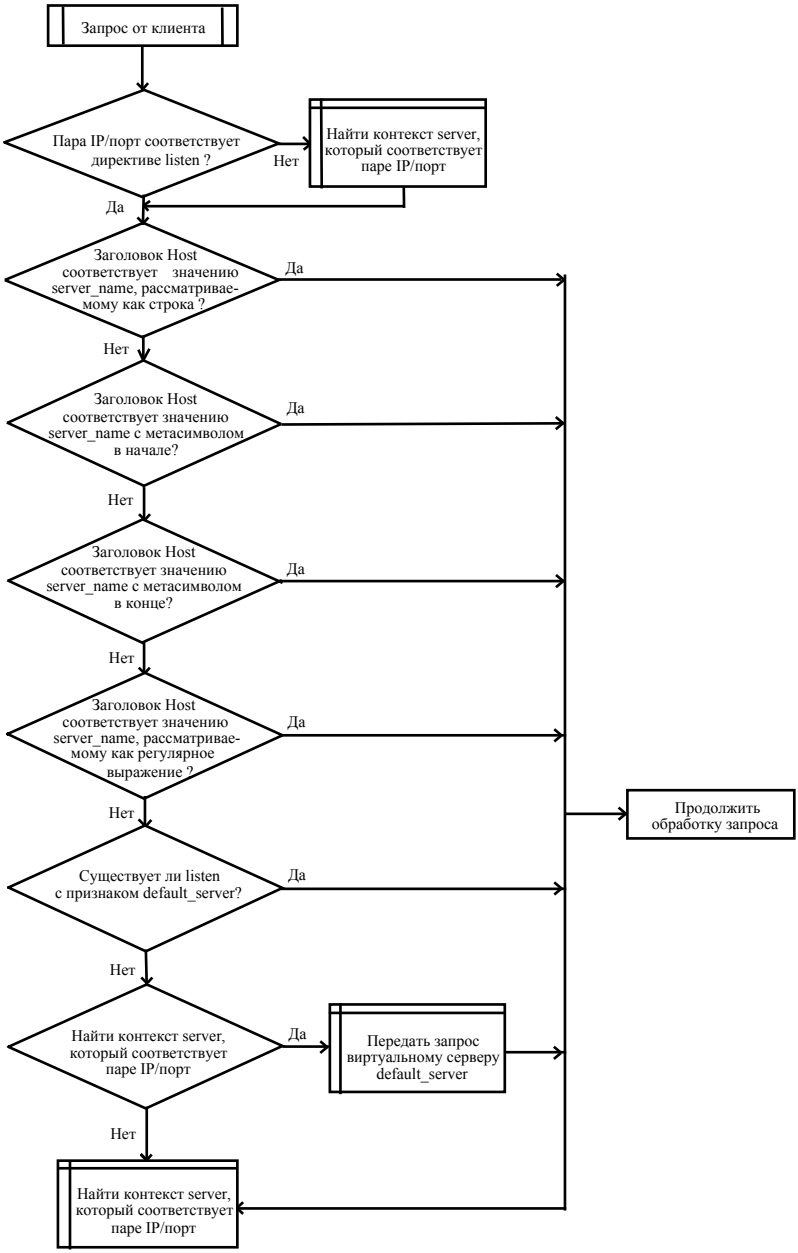
```
server_name ~^www\.example\.com$;
server_name ~^www(\d+).example\.(com)$;
```

Вторая форма применяется для запоминания подвыражений, на которые затем можно ссылаться (по номеру `$1`, `$2` и т. д.) в последующих директивах.

Чтобы определить, какой виртуальный сервер должен обслужить данный запрос, NGINX применяется следующий алгоритм.

1. Сопоставить IP-адрес и порт с указанными в директиве `listen`.
2. Сопоставить заголовок Host со значением директивы `server_name`, рассматриваемым как строка.
3. Сопоставить заголовок Host со значением директивы `server_name`, рассматриваемым как строка, считая, что в начале находится метасимвол *.
4. Сопоставить заголовок Host со значением директивы `server_name`, рассматриваемым как строка, считая, что в конце находится метасимвол *.
5. Сопоставить заголовок Host со значением директивы `server_name`, рассматриваемым как регулярное выражение.
6. Если все попытки сопоставления заголовка Host закончились неудачей, использовать ту директиву `listen`, в которой имеется признак `default_server`.
7. Если все попытки сопоставления заголовка Host закончились неудачей и директивы `listen` с признаком `default_server` не существует, использовать первый сервер, в котором директива `listen` удовлетворяет условию шага 1.

Эта логика изображена на следующей блок-схеме:


Запрос от клиента
Пара IP/порт соответствует директиве listen ?
Нет
Найти контекст server, который соответствует паре IP/порт
Да
Заголовок Host соответствует значению server_name, рассматриваемому как строка ?
Да
Нет
Заголовок Host соответствует значению server_name с метасимволом в начале?
Да
Нет
Заголовок Host соответствует значению server_name с метасимволом в конце?
Да
Нет
Заголовок Host соответствует значению server_name, рассматриваемому как регулярное выражение ?
Да
Нет
Существует ли listen с признаком default_server?
Да
Нет
Найти контекст server, который соответствует паре IP/порт
Да
Передать запрос виртуальному серверу default_server
Нет
Найти контекст server, который соответствует паре IP/порт
Продолжить обработку запроса

Признак `default_server` позволяет обработать запросы, которые иначе остались бы необработанными. Поэтому рекомендуется всегда задавать этот признак явно, чтобы не гадать потом, почему запросы обрабатываются странным образом.

Кроме того, признак `default_server` полезен, когда требуется сконфигурировать несколько виртуальных серверов с одной и той же директивой `listen`. Все описанные выше директивы будут одинаковы для всех подходящих блоков server.

Местоположения - где, когда и как

Директива `location` может встречаться в секции с описанием виртуального сервера и содержит в качестве параметра URI-адрес, поступивший от клиента или в результате внутренней переадресации. Местоположения могут быть вложенными (с несколькими исключениями). Их назначение - определить максимально специализированную конфигурацию для обработки запроса.

Местоположение задается следующим образом:

```
location [модификатор] uri {...}
```

Можно задавать также именованные местоположения:

```
location @name {...}
```

Именованное местоположение доступно только при внутренней переадресации и может быть задано только на уровне контекста сервера. При этом сохраняется тот URI, который был перед входом в блок `location`.

Модификаторы изменяют обработку местоположения следующим образом:

Модификаторы местоположения

Модификатор	Обработка
=	Сравнить буквально и завершить поиск
~	Сопоставление с регулярным выражением с учетом регистра
~*	Сопоставление с регулярным выражением без учета регистра
^~	Прекратить обработку до сопоставления этого местоположения с регулярным выражением, если это совпадение с самым длинным префиксом. Отметим, что это не сопоставление с регулярным выражением, задача данного модификатора - как раз предотвратить такое сопоставление

Когда приходит запрос, для указанного в нем URI ищется самое подходящее местоположение. Происходит это следующим образом.

- Среди местоположений без регулярного выражения ищется самое специфичное (с самым длинным совпадающим префиксом), порядок следования местоположений при этом не учитывается.
- Сопоставление с регулярными выражениями производится в том порядке, в каком они следуют в конфигурационном файле. Поиск прекращается при обнаружении первого совпадения. Далее для обработки запроса используется самое подходящее местоположение.

Описанное выше сопоставление производится после декодирования URI; например, строка `"%20"` в URI совпадает с пробелом `" "` в местоположении.

Именованные местоположения можно использовать только при внутренней переадресации запросов.

В таблице ниже перечислены директивы, которые могут встречаться только внутри местоположения.

Директивы, употребляемые внутри секции location

Директива	Описание
`alias`	Определяет путь в файловой системе, соответствующий имени местоположения. Если местоположение задано с помощью регулярного выражения, то значение `alias` должно ссылаться на запомненные подвыражения, `alias` подставляется вместо части URI, сопоставившейся с местоположением; оставшаяся часть URI не изменяется и становится частью пути в файловой системе. Применение директивы `alias` чревато ошибками в случае перемещения участков конфигурационного файла, поэтому лучше использовать директиву `root`, если только нет необходимости модифицировать URI для поиска нужного файла
`internal`	Означает, что данное местоположение можно использовать только для внутренних запросов (переадресации, определенной в других директивах, переписывании запросов, для страниц ошибок и т. д.)
`limit_except`	Ограничивает применение данного местоположения только указанными глаголами HTTP (`GET` включает также `HEAD`)

Кроме того, некоторые директивы, относящиеся к секции `http`, могут употребляться и в секции `location`. Полный перечень см. в приложении А «Справочник по директивам».

Директива `try_files` заслуживает особого упоминания. Хотя ее можно использовать в контексте `server`, чаще она встречается в описании местоположения. Ее задача - поискать перечисленные в параметрах файлы в указанном порядке; как только будет найден первый файл, поиск прекращается. Обычно этот механизм применяется, чтобы сопоставить потенциальные файлы с некоторой переменной, а затем передать обработку именованному местоположению, как показано в следующем примере:

```
location / {
   try_files $uri $uri/ @mongrel;
}

location @mongrel {
   proxy_pass http://appserver;
}
```

Здесь если переданному URI не соответствует файл, то неявно производится попытка найти каталог, а затем обработка перепоручается серверу `appserver` с помощью прокси. О том, как использовать секции `location`, директивы `try_fiies` и `proxy_pass` для решения различных задач, мы будем не раз говорить на страницах этой книги.

Местоположения могут быть вложенными за исключением следующих случаев:

- префикс равен "=";
- местоположение именованное.

Рекомендуется вкладывать местоположения, определяемые регулярными выражениями, в местоположения с простой строкой в качестве URI, например:

```
# сначала входим через root
location / {

  # затем ищем самую длинную подстроку
  # отметим, что сопоставление с регулярным выражением не производится
  location ^~ /css {

    # затем сопоставляем с этим регулярным выражением
    location ~* /css/.*\.css$ {
    }
  }
}
```

Секция с описанием почтового сервера

Секция с описанием почтового сервера, или конфигурационный контекст почты доступен, только если при конфигурировании NGINX был задан параметр `--with-mail`. В этой секции описываются все аспекты работы с почтовым модулем.

Почтовый модуль содержит директивы, определяющие поведение проксирования соединений с почтовыми серверами. В серверном контексте могут употребляться те же директивы `listen` и `server_name`, которые мы рассматривали в секции `http`.

NGINX умеет проксировать протоколы IMAP, POP3 и SMTP. В таблице ниже перечислены директивы, относящиеся к этому модулю.

Директивы для модуля mail

Директива	Описание
`auth_http`	Задает сервер, который служит для аутентификации пользователя в протоколах POP3 и IMAP. Функциональность этого сервера подробно рассматривается в главе 3
`imap_capabilities`	Определяет, какие возможности протокола IMAP4 поддерживает проксируемый сервер
`pop3_capabilities`	Определяет, какие возможности протокола POP3 поддерживает проксируемый сервер
`protocol`	Определяет, какой протокол поддерживает данный контекст виртуального сервера
`proxy`	Разрешает или запрещает проксирование почты
`proxy_buffer`	Позволяет задать размер буфера, используемого для проксирования. По умолчанию размер буфера равен размеру страницы
`proxy_pass_error_message`	Полезна, если процедура аутентификации на проксируемом сервере возвращает осмысленное сообщение клиенту
`proxy_timeout`	Используется, если таймаут должен быть больше значения по умолчанию - 24 часа
`xclient`	В протоколе SMTP предусмотрена проверка на основе параметров `IP/HELO/LOGIN`, которые передаются в команде `XCLIENT`. Данная директива позволяет NGINX передать эту информацию серверу

Если NGINX собиралась с поддержкой SSL (с параметром `--with-mail_ssl_module`), то, помимо вышеописанных, доступны следующие директивы.

Директивы для модуля mail с поддержкой SSL

Директива	Описание
`ssl`	Включает или выключает поддержку SSL-транзакций в этом контексте
`ssl_certificate`	Определяет путь к сертификату (или сертификатам) SSL в формате PEM для данного виртуального сервера
`ssl_certificate_key`	Определяет путь к секретному ключу SSL в формате PEM для данного виртуального сервера
`ssl_ciphers`	Определяет, какие шифры нужно поддерживать в этом контексте (в формате OpenSSL)
`ssl_prefer_server_ciphers`	Указывает, что при использовании протоколов SSLv3 и TLSv1 серверные шифры более приоритетны, чем клиентские
`ssl_protocols`	Определяет, какие протоколы SSL разрешить
`ssl_session_cache`	Определяет параметры кэша сеансов SSL, а также разрешает или запрещает разделение кэша всеми рабочими процессами
`ssl_session_timeout`	Определяет, сколько времени клиент может пользоваться одними и теми же параметрами SSL при условии, что они хранятся в кэше

Полный пример конфигурации

Ниже приведен пример конфигурационного файла, содержащего секции, рассмотренные в этой главе. Подчеркнем, что его не следует копировать и использовать «как есть». Скорее всего, он вам не подойдет, а включен лишь для того, чтобы дать представление о том, как выглядит полный конфигурационный файл.

```
user www;
worker_processes 12;
error_log /var/log/nginx/error.log;
pid /var/run/nginx.pid;

events {
  use /dev/poll;
  worker_connections 2048;
}

http {
  include /opt/local/etc/nginx/mime.types;
  default_type application/octet-stream;
  sendfile on;
  tcp_nopush on;
  tcp_nodelay on;
```

```
  keepalive_timeout 65;
  server_names_hash_max_size 1024;

  server {
    listen 80;
    return 444;
  }

  server {
    listen 80;
    server_name www.example.com;

    location / {
      try_files $uri $uri/ @mongrel;
    }

    location @mongrel {
      proxy_pass http://127.0.0.1:8080;
    }
  }
}
```

Резюме

В этой главе мы рассмотрели устройство конфигурационного файла NGINX. Его модульная структура в какой-то мере отражает структуру самой NG1NX. Глобальный конфигурационный блок описывает все аспекты поведения NGINX в целом. Для каждого протокола, поддерживаемого NGINX, имеется отдельная конфигурационная секция. Порядок обработки запросов можно уточнить, описав виртуальные серверы внутри контекста протокола (`http` или `mail`), таким образом, что запрос будет направляться серверу, обслуживающему конкретную пару IP-адрес/порт. В контексте `http` можно затем определить местоположения, сопоставляемые с URI запроса. Местоположения могут быть вложенными или упорядоченными иным способом, это позволяет направить запрос в нужное место файловой системы или серверу приложений.

Но пока мы ничего не сказали о конфигурировании других модулей, включаемых в двоичный файл `nginx` на этапе компиляции. Мы будем обсуждать эти дополнительные директивы при рассмотрении использования модуля для решения конкретной задачи. Мы также не объяснили, какие переменные NGINX предоставляет для конфигурирования и как ими пользоваться. Об этом также пойдет речь далее. Эта глава посвящена лишь основам настройки NGINX.

В следующей главе мы рассмотрим настройку почтового модуля NG1NX, который осуществляет проксирование электронной почты.

Глава 3. Почтовый модуль

NGINX проектировалась не только для обслуживания веб-запросов, но и как средство проксирования почтовых служб. В этой главе мы научимся настраивать NGINX в качестве почтового прокси-сервера для протоколов POP3, IMAP и SMTP. Мы рассмотрим следующие вопросы.

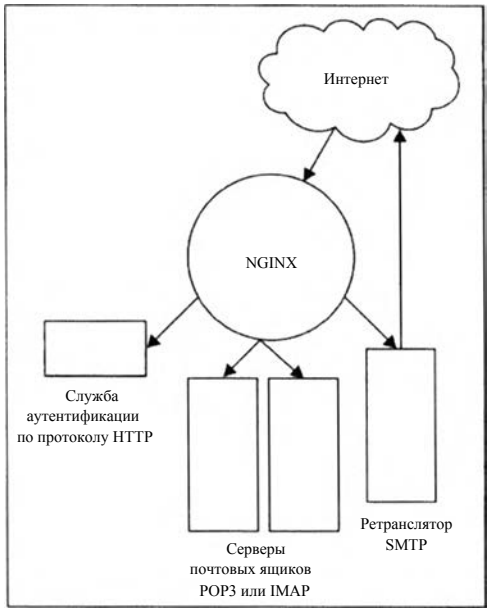
- Простая служба проксирования.
- Служба аутентификации.
- Использование в связке с memcached.
- Интерпретация журналов.
- Ограничения операционной системы.

Простая служба проксирования

Модуль почтового прокси-сервера в NGINX первоначально был разработан для компании Fast Mail. Она хотела предоставлять своим пользователям единственную оконечную точку IMAP, а размещать реальную почтовую учетную запись на одном из нескольких почтовых серверов заднего плана. В то время для проксирования использовалась классическая модель Unix, основанная на системном вызове `fork()`, то есть для каждого нового соединения порождался новый процесс. В IMAP соединения существуют очень долго, поэтому и процессы оказывались долгожителями. В результате прокси-серверы работали крайне медленно, поскольку им приходилось управлять многочисленными процессами, созданными для обслуживания соединений. Событийная модель процессов NGINX гораздо лучше отвечала таким требованиям. В качестве почтового прокси-сервера NGINX может перенаправлять трафик на сколько угодно почтовых серверов заднего плана, на которых размещаются учетные записи. Это открывает возможность предоставить клиентам единственную оконечную точку, а количество реальных серверов масштабировать с увеличением числа пользователей. На базе этой модели построены

как коммерческие, так и открытые решения для электронной почты, в том числе Atmail и Zimbra.

На рисунке ниже графически представлена описанная схема.



Поступивший запрос обрабатывается в соответствии с протоколом. Почтовый модуль можно настроить по-разному для протоколов POP3, IMAP и SMTP. Для каждого протокола NGINX обращается к службе аутентификации, передавая ей имя пользователя и пароль. В случае успешной аутентификации соединение проксируется почтовому серверу, указанному в ответе, полученном от службы аутентификации. В противном случае соединение с клиентом разрывается. Таким образом, служба аутентификации определяет, какую из служб POP3 / IMAP / SMTP должен использовать клиент и на каком сервере. Количество почтовых серверов не ограничено, и NGINX может выступать в роли прокси-сервера для всех, предоставляя клиентам единый центральный шлюз.

Прокси-сервер действует от чьего-то имени. В данном случае NGINX действует от имени почтового клиента, закрывая одно соединение и открывая другое с проксируемым сервером. Это

означает, что прямого соединения между почтовым клиентом и настоящим сервером почтовых ящиков или ретранслятором SMTP нет.

Правила, основанные на информации, содержащейся в соединении с клиентом, работать не будут, если только почтовая программа не поддерживает расширения, например XCLIENT для SMTP.

При проектировании архитектуры с участием прокси-сервера нужно иметь в виду важный момент - компьютер, на котором размещен прокси-сервер, должен поддерживать больше соединений, чем типичный проксируемый сервер. Такая же вычислительная мощность и память, как у сервера почтовых ящиков, ему не нужна, но количество одновременных соединений следует принимать во внимание.

Служба POP3

Протокол **Post Office Protocol** - это стандартный протокол Интернета для доступа к сообщениям, хранящимся на сервере почтовых ящиков. Его текущая версия имеет номер 3, отсюда и название **POP3**. Обычно почтовый клиент запрашивает у сервера почтовых ящиков все новые сообщения в одном сеансе, а затем закрывает соединение, после чего сервер почтовых ящиков удаляет все сообщения, помеченные как прочитанные.

Чтобы использовать NGINX в роли прокси для POP3, необходимо включить несколько простых директив:

```
mail {
  auth_http localhost:9000/auth;

  server {
    listen 110;
    protocol pop3;
    proxy on;
  }
}
```

В этом фрагменте мы включаем почтовый модуль и настраиваем его для работы со службой POP3, которая запущена на порту 9000 на той же машине. NGINX будет прослушивать порт 110 для всех локальных IP-адресов, играя роль прокси-сервера для POP3. Обратите внимание, что сами почтовые серверы мы здесь не настраи-

ваем - задача известить NGINX о том, с каким сервером соединить конкретного клиента, возлагается на службу аутентификации.

Если почтовый сервер поддерживает не все возможности (или вы хотите явно ограничить их перечень), то NGINX поможет в этом:

```
mail {
  pop3_capabilities TOP USER;
}
```

Возможности (capabilities) - это способ объявить о поддержке необязательных команд. В случае POP3 клиент может запросить список поддерживаемых возможностей до или после аутентификации, поэтому важно правильно настроить их в NGINX.

Можно также задать перечень поддерживаемых методов аутентификации:

```
mail {
  pop3_auth apop cram-md5;
}
```

Если поддерживается метод аутентификации APOP, то служба аутентификации должна сообщить NGINX пароль пользователя в открытом виде, чтобы та смогла сгенерировать MD5-свертку.

Служба IMAP

Протокол **Internet Message Access Protocol** также является стандартным протоколом Интернета для доступа к сообщениям, хранящимся на сервере почтовых ящиков. Он существенно расширяет функциональность более раннего протокола POP. Как правило, все сообщения остаются на сервере, поэтому к одному почтовому ящику может обращаться несколько клиентов. Это также означает, что с IMAP-сервером заднего плана может быть установлено гораздо больше долгоживущих соединений, чем с POP3-сервером.

Проксирование соединений с IMAP-сервером настраивается примерно так же, как с POP3-сервером:

```
mail {
  auth_http localhost:9000/auth;
  imap_capabilities IMAP4rev1 UIDPLUS QUOTA;
  imap_auth login cram-md5;

  server {
```

```
    listen 143;
    protocol imap;
    proxy on;
  }
}
```

Отметим, что протокол задавать необязательно, потому что `imap` подразумевается по умолчанию. Мы включили его лишь для большей ясности.

Директивы `imap_capabilities` и `imap_auth` работают так же, как их аналоги для POP3.

Служба SMTP

Simple Mail Transport Protocol - стандартный протокол Интернета для передачи почтовых сообщений от одного серверу другому или от клиента серверу. Первоначально в нем не была предусмотрена аутентификация, по теперь определено расширение SMTP-AUTH.

Выше мы уже познакомились с логикой настройки почтового модуля. Она несложна и применима также к проксированию SMTP:

```
mail {
  auth_http localhost:9000/auth;
  smtp_capabilities PIPELINING 8BITMIME DSN;
  smtp_auth login cram-md5;

  server {
    listen 25;
    protocol smtp;
    proxy on;
  }
}
```

Наш прокси-сервер объявляет о поддержке лишь тех возможностей, которые перечислены в директиве `smtp_capabilities`, в противном случае он выдаст только список поддерживаемых механизмов аутентификации, потому что список расширений отправляется клиенту в ответ на команду HELO/EHLO. Это может быть полезно при проксировании на несколько SMTP-серверов, поддерживающих различные возможности. Можно настроить NGINX, так что она будет объявлять только о возможностях, поддерживаемых всеми серверами. Важно указать в списке только расширения, которые поддерживает сам SMTP-сервер.

Поскольку SMTP-AUTH - расширение SMTP, которое необязательно поддерживается в любой конфигурации, NGINX умеет проксировать SMTP-соединения, которые вообще не аутентифицируются. В таком случае службе аутентификации доступны только команды протокола `HELO`, `MAIL FROM` и `RCPT TO`, на основании которых она должна определить, какому серверу передать соединение с данным клиентом. В этой ситуации значением директивы `smtp_auth` должно быть `none`.

Использование SSL/TLS

Если организация требует шифровать почтовый трафик или вы сами хотите обеспечить повышенную защиту, то можете настроить NGINX, так чтобы использовался протокол TLS, реализовав тем самым спецификации POP3 поверх SSL, IMAP поверх SSL или SMTP поверх SSL. Для включения поддержки TLS нужно добавить либо директиву `starttls on` для поддержки команд `STLS/STARTTLS`, либо директиву `ssl on` для поддержки чистого протокола SSL/TLS. Кроме того, необходимо добавить директивы `ssl_*`, адаптированные для вашего вычислительного центра:

```
mail {
  # разрешить команду STLS для POP3 и STARTTLS для IMAP и SMTP
  starttls on;

  # отдать приоритет списку шифров сервера, чтобы мы сами
  # определяли уровень защиты
  ssl_prefer_server_ciphers on;

  # использовать только эти протоколы
  ssl_protocols TLSv1 SSLv3;

  #использовать только стойкие наборы шифров,
  # запретив анонимные DH и MD5, и отсортировать по стойкости
  ssl_ciphers HIGH:!ADH:!MD5:@STRENGTH.

  #использовать разделяемый кэш сеансов SSL. общий для всех
  # рабочих процессов
  ssl_session_cache shared:MAIL:10m;

  #сертификат и ключ для этого компьютера
  ssl_certificate /usr/local/etc/nginx/mail.example.com.crt;
  ssl_certificate.key /usr/local/etc/nginx/mail.example.com.key;
}
```

О различиях между соединением с чистым SSL/TLS и переходом от открытого соединения к зашифрованному с помощью SSL/TLS см. статью на странице https://www.fastmail.fm/help/technology_ssl_vs_tls_starttls.html.

Генерация SSL-сертификата с помощью OpenSSL

Для тех, кто никогда раньше не генерировал SSL-сертификаты, ниже приведена последовательность шагов.

Выполнить запрос на создание сертификата:

```
$ openssl req -newkey rsa:2048 -nodes -out mail.example.com.csr
-keyout mail.example.com.key
```

В ответ должен быть напечатан такой текст:

```
.....................................................
.....................................................
....+++
.....................+++
writing new private key to 'mail.example.com.key'
-----
You are about to be asked to enter information that will
be incorporated
into your certificate request.
What you are about to enter is what is called a
Distinguished Name or a DN.
There are quite a few fields but you can leave some
blank
For some fields there will be a default value,
If you enter '.', the field will be left blank.
-----
Country Name (2 letter code) [AU]:CH
State or Province Name (full name) [Some-State]:Zurich
Locality Name (eg, city) []:ZH
Organization Name (eg, company) [Internet Widgits Pty
Ltd]:Example Company
Organizational Unit Name (eg, section) []:
Common Name (e.g. server FQDN or YOUR name) []:mail.
example.com
Email Address []:
Please enter the following 'extra' attributes
to be sent with your certificate request
A challenge password []:
An optional company name []:
```

Этот запрос на подписание сертификата (Certificate Signing Request) (файл mail.example.com.csr) можно отправить для подписания в удостоверяющий центр, например Verisign или

```
$ openssl x509 -req -days 365 -in mail.example.com.csr
-signkey mail.example.com.key -out mail.example.com.crt
```

В ответ будет напечатано:

```
Signature ok
subject=/C=CH/ST=Zurich/L=ZH/0=Example Company/CN=mail.
example.com
Getting Private key
```

Подписанный сертификат показан на рисунке ниже.

Отметим, что при использовании самоподписанного сертификата клиент, подключающийся к серверу, получит сообщение об ошибке. На производственный сервер следует устанавливать только сертификаты, подписанные признанным удостоверяющим центром.

```
..... BEGIN CERTIFICATE.....

MIIDPDCCAiQCCQDdPKFcY1X35jANBgkqhkiG9wOBAQUFADBgMQswCQYDVQQGEwJD
SDEPMAOGAlUECAwGWnVyaWNoMQswCQYDVQQHDAJaSDEYMBYGAlUECgwPRXhhbXBs
ZSBDb21wYW55MRkwFwYDVQQDDBBtYWlsLmV4YWlwbGUuY29tMB4XDTEyMDgzMTEO
MjczMloXDTEzMDgzMTEOMjczMlowYDELMAkGAlUEBhMCQOgxDzANBgNVBAgMBlp1
cmljaDELMAkGA1UEBwwCWkgxGDAWBgNVBAoMDOV4YW1wbGUgQ29tcGFueTEZMBcG
A1UEAwwQbWFpbCSleGFtcGxlLmNvbTCCASIwDQYJKoZIhvcNAQEBBQADggEPADCC
AQoCggEBAN8WUGzQIKR+iuTxtLPko/zSR+DbjDYqbMo4PdNvEN46nTFMkktvOslk
1kfk9l2jzVcmUUSZayLp3woDgxRpkpQ5eRp87yeifsZwPJlXf VPTgf XtQkktfPVn
uzOMf 70gd2Xt8uI6nOAtODAr8+CxebIpRwIwZBXPrwwFFjQvy4/qD7EXs33+x5U8
9CMxkGo2FPqCSYE39jN3JtlZ9YibnZh01NALHRvnqyw3mdzR340mu5WNFjl/NElp
MOyFL7+5wzI4ktgmAo+Mic6JnXCObSjrL1xZjWf n/5TQiYQVzUit4jdlCswWtCHw
tv67TRQ3edgvssvzfZlm7QfBbdYGjkUCAwEAATANBgkqhkiG9wOBAQUFAAOCAQEA
TDfdngMRk2w/1KCGbxrg9bVmfKXUSIfpWyt0hG02EtLx83TZajqwtOKhmPh9Q/lc
GZdF1PGscdJ2BcOeJBUGyt6mevEi2Dg4h727yVvnacnViQvzyLxQgmeC5rDEj4EC
yDzzi4nOI/rddjPeQO+cMFHz26scsKYoRemzpOyHT8JhK8AF2iOioLzwaMqxC+ll
U7lkinHdTaG6nT4WpHO5HtSBno8Xco/ujY6xIrShiPOnaOd/B4TRCmB96KYhyMdd
AyrOZgLqsskKeAlnmuSJA/7zbp1LwHarvUVFpzKed73554lfJ5kpyOciHrIfyj/2
dM/tjsDVjpE2B/meYBx8Kg==

..... END CERTIFICATE.....
```

Полный пример конфигурации почтового модуля

Доступ к почтовым службам часто осуществляется через один шлюз. В следующем примере конфигурации NGINX разрешено обслуживать трафик по протоколам POP3, IMAP и SMTP (в том числе зашифрованный) через одну службу аутентификации, предоставляя также клиентам возможность использовать команды STLS/STARTTLS на незашифрованных портах:

```
events {
  worker_connections 1024;
}

mail {
  server_name mail.example.com;
  auth_http localhost:9000/auth;

  proxy on;

  ssl_prefer_server_ciphers on;
  ssl_protocols TLSv1 SSLv3;
  ssl_ciphers HIGH:!ADH:!MD5:@STRENGTH;
  ssl_session_cache shared:MAIL:10m;
  ssl_certificate /usr/local/etc/nginx/mail.example.com.crt;
  ssl_certificate_key /usr/local/etc/nginx/mail.example.com.key;

  pop3_capabilities TOP USER;
  imap_capabilities IMAP4rev1 UIDPLUS QUOTA;
  smtp_capabilities PIPELINING 8BITMIME DSN;

  pop3_auth apop cram-md5:
  imap_auth login cram-md5;
  smtp_auth login cram-md5;
  server {
     listen 25;
     protocol smtp;
     timeout 120000;
     }

  server {
     listen 465;
     protocol smtp;
     ssl on;
     }

  server {
     listen 587;
     protocol smtp;
     starttls on;
     }

  server {
     listen 110;
     protocol pop3;
     starttls on;
     }

  server {
```

```
    listen 995;
    protocol p;
    ssl on;
  }

  server {
    listen 143;
    protocol imap;
    starttls on;
  }

  server {
    listen 993;
    protocol, imap;
    ssl on;
  }
}
```

Как видите, в начале контекста `mail` мы объявили имя сервера. Это сделано для того, чтобы ко всем нашим почтовым службам можно было обращаться по имени `mail.example.com`. Даже если истинное имя машины, на которой работает NGINX, другое и у каждого почтового сервера есть собственное имя хоста, мы хотим, чтобы этот прокси-сервер был единственной точкой, известной пользователям. Именно это имя будет использоваться всюду, где NGINX должна представиться, например в начальном приветствии по протоколу SMTP.

Директива `timeout` в контексте сервера SMTP нужна для того, чтобы удвоить значение по умолчанию, поскольку нам известно, что этот конкретный проксируемый ретранслятор SMTP добавляет искусственную задержку, чтобы воспрепятствовать рассылке через него почтового спама.

Служба аутентификации

В предыдущем разделе мы несколько раз упомянули службу аутентификации, но не сказали, что конкретно она делает. Когда NGINX получает от пользователя запрос по протоколу POP3, IMAP или SMTP, то первым делом необходимо аутентифицировать соединение. NGINX не выполняет аутентификацию самостоятельно, а отправляет запрос службе аутентификации. Ответ, полученный от этой службы, NGINX использует, чтобы установить соединение с проксируемым почтовым сервером.

Служба аутентификации может быть написана на любом языке, лишь бы она была согласована с протоколом аутентификации, установленным NGINX. Этот протокол похож на HTTP, поэтому написать свою службу аутентификации несложно.

В запросе NGINX службе аутентификации присутствуют следующие заголовки:

- Host
- Auth-Method
- Auth-User
- Auth-Pass
- Auth-Salt
- Auth-Protocol
- Auth-Login-Attempt
- Client-IP
- Client-Host
- Auth-SMTP-Helo
- Auth-SMTP-From
- Auth-SMTP-To

Смысл их не нуждается в объяснении, и не все заголовки обязательны. Подробнее мы рассмотрим их, когда будем писать собственную службу аутентификации.

Для реализации службы аутентификации мы выбрали язык Ruby. Если его нет на вашей машине, не торопитесь устанавливать. Написанные на Ruby программы очень легко читать, поэтому просто смотрите на код и комментарии к нему. Адаптация этой программы к вашей среде и порядок ее запуска выходят за рамки этой книги. Этот пример приведен лишь в качестве отправной точки для написания необходимой вам службы аутентификации.

О том, как установить Ruby, подробно написано на сайте `https://rvm.io`.

Начнем с рассмотрения запроса.

Сначала мы извлекаем из полученных от NGINX заголовков интересующие нас значения.

```
# механизм аутентификации
meth = @env['HTTP_AUTH_METHOD']
# имя пользователя (login)
user = @env[' HTTP_AUTH_USER']
# пароль, открытый или зашифрованный в зависимости от используемого
```

```
# механизма аутентификации
pass = @env['HTTP_AUTH_PASS']
# в некоторых механизмах аутентификации для шифрования открытого пароля
# необходима затравка, но не в нашем примере
salt = @env['HTTP_AUTH_SALT']
# проксируемый протокол
proto = @env['HTTP_AUTH_PROTOCOL']
# количество попыток должно быть целым числом
attempt = @env['HTTP_AUTH_LOGIN_ATTEMPT'].to_i
# в нашей реализации не используется, приведено только для справки
client = @env['HTTP_CLIENT_IP']
host = @env['HTTP_CLIENT_HOST']
```

Что означает символ `@`?
Символ `@` в Ruby обозначает переменную класса. Мы используем такие переменные, чтобы было проще передавать значения из одного места программы в другое. В коде выше переменная `@env` ссылается на окружение, передаваемое в Rack-запрос. Помимо нужных нам HTTP-заголовков, окружение содержит дополнительную информацию о том, как запущена служба.

Разобравшись с тем, как обработать пришедшие от NGINX заголовки, мы теперь должны что-то сделать с содержащейся в них информацией и отправить NGINX ответ. Ожидается, что ответ от службы аутентификации будет содержать следующие заголовки:

- `Auth-Status`: любое значение, кроме `OK`, считается ошибкой.
- `Auth-Server`: IP-адрес, па который проксируется соединение.
- `Auth-Port`: порт, на который проксируется соединение.
- `Auth-User`: имя пользователя, которое должно использоваться для аутентификации на почтовом сервере.
- `Auth-Pass`: пароль в открытом виде для команды APOP.
- `Auth-wait`: сколько секунд ждать перед повторной попыткой аутентификации.
- `Auth-Error-Code`: альтернативный код ошибки для возврата клиенту.

Чаще всего используются заголовки `Auth-Status`, `Auth-Server` и `Auth-Port`. Их присутствия в ответе достаточно, чтобы аутентификации считалась состоявшейся.

Как видно из показанного ниже фрагмента кода, в зависимости от ситуации могут включаться и другие заголовки. Сам ответ содержит только заголовки с подставленными в них значениями.

Сначала проверим, не исчерпано ли количество попыток:

```
# вернуть ошибку, если превышено максимальное количество попыток входа if
attempt > @max_attempts
   @res["Auth-Status"] = "Maximum login attempts exceeded"
   return
```

Если это не так, возвращаем подходящие заголовки, записывая в них значения, полученные от нашего механизма аутентификации:

```
@res["Auth-Status"] = "OK"
@res["Auth-Server"] = @mailhost
# возвращаем порт, соответствующий этому протоколу
@res["Auth-Port"] = MailAuth::Port[proto]
# если используется APOP, то пароль нужно вернуть в открытом виде
if meth == 'apop' && proto == 'pop3'
   @res["Auth-User"] = user
   @res["Auth-Pass"] = pass
end
```

Если попытка аутентификации не удалась, мы должны сообщить об этом NGINX.

```
#в случае ошибки аутентификации возвращаем соответствующий ответ
@res["Auth-Status"] = «Invalid login of password»
#и устанавливаем время в секундах, которое клиент должен
# ждать перед следующей попыткой
@res["Auth-Wait"] = "3"
# Можно также установить код ошибки, который следует вернуть SMTP-клиету
@res["Auth-Error-Code"] = "535 5.7.8"
```

Не все заголовки обязательны; какие именно включать, зависит от состояния запроса на аутентификацию и (или) возникших ошибок.

Интересно, что в заголовке `Auth-User` можно вернуть не то имя пользователя, которое указано в запросе. Это может оказаться полезным, например, в случае, когда производится переход от старого проксируемого сервера, который принимал имя пользователя без домена, к новому, который требует указания домена. Благодаря указанной возможности NGINX может передавать новому проксируемому серверу модифицированное имя.

База данных для аутентификации может быть произвольной: плоский текстовый файл, каталог LDAP, реляционная СУБД и т. д. Не-

обязательно использовать то же хранилище, к которому обращается сама почтовая служба, однако данные в обоих хранилищах должны быть синхронизированы во избежание ошибок из-за их расхождения.

В нашем примере данные для аутентификации хранятся в простом хеше:

```
@auths = { "test:1234" => '127.0.1.1' }
```

Для проверки пользователя нужно просто произвести поиск в этом хеше:

```
# возвращается значение, соответствующее ключу 'user:pass'

  if @auths.key?("#{user}:#{pass}")
  @mailhost = @auths["#{user}:#{pass}"]
return true
# если такого ключа нет, метод возвращает false
  else
  return false
  end
```

Объединяя все три части, получаем полный код службы аутентификации:

```
#!/usr/bin/env rackup
# Это простой HTTP-сервер, согласованный с протоколом аутентификации,
# который определен почтовым модулем NGINX.

require 'logger'
require 'rack'

module MailAuth

# настраиваем соответствие протоколов и портов
Port = {
   'smtp' => '25',
   'pop3' => '110',
   'imap' => '143'

}
  class Handler
    def initialize
    # настраиваем протоколирование, как для почтовой службы
    @log = Logger.new("| logger -p mail.info")
    # заменяем обычную временную метку именем службы и pid'ом
    @log.datetime_format = "nginx_mail_proxy_auth pid: "
    # заголовок "Auth-Server" должен содержать IP-адрес
```

```
@N@ilOGAI = '127.0.0.1'
# задаем максимальное число попыток входа
@max_attempts = 3
# Для этого примера в качестве 'базы данных' аутентификации мы
# используем фиксированный хеш.
# В реальной программе нужно будем заменить его методом,
# который подключается к каталогу LDAP или к базе данных
@auths = { "test:1234" => '127.0.1.1'}
end
```

После описанной выше инициализации модуля мы сообщаем Rack, какие запросы хотим обрабатывать и определяем метод `get`, который будет отвечать на запросы от NGINX.

```
def call(env)
  # интересующие нас заголовки находятся в окружении
  @env = env
  # настраиваем объекты запроса и ответа
  @req = Rack::Request.new(env)
  @res = Rack::Response.new
  # передаем управление методу, называющемуся так же, как
  # HTTP-метод, которым был отправлен запрос
  self. send(@req. request_method.downcase)
  # возвращаемся сюда, чтобы завершить ответ
  @res.finish
end

def get
  # механизм аутентификации
  meth = @env['HTTP_AUTH_METHOD']
  # имя пользователя (login)
  user = @env['HTTP_AUTH_USER']
  # пароль, открытый или зашифрованный в зависимости от
  # используемого механизма аутентификации
  pass = @env['HTTP_AUTH_PASS']
  # в некоторых механизмах аутентификации для шифрования
  # открытого пароля необходима затравка, но не в нашем примере
  salt = @env['HTTP_AUTH_SALT']
  # проксируемый протокол
  proto = @env['HTTP_AUTH_PROTOCOL']
  # количество попыток должно быть целым числом
  attempt = @env['HTTP_AUTH_LOGIN_ATTEMPT'].to_i
  # в нашей реализации не используется, приведено только для справки
  client = @env['HTTP_CLIENT_IP']
  host = @env['HTTP_CLIENT_HOST']
  # вернуть ошибку, если превышено максимальное количество попыток
  # входа
  if attempt > @max_attempts
      @res["Auth-Status"] = "Maximum login attempts exceeded"
```

```
    return
end

# для особого случая smtp-транзакций, не требующих
# аутентификации, в файле nginx.conf содержится такой фрагмент:
# smtp_auth none;
# возможно, понадобится справочная таблица для переадресации
# некоторых отправителей на определенные SMTP-серверы
if meth == 'none' && proto == 'smtp'
    helo = @env['HTTP_AUTH_SMTP_HELO']
    # мы хотим выделить адреса из следующих двух заголовков
    from = @env['HTTP_AUTH_SMTP_FROM'].split(/: /)[1]
    to = @env['HTTP_AUTH_SMTP_TO'].split(/: /)[1]
    @res["Auth-Status"] = "OK"
    @res["Auth-Server"] = @mailhost
    # возвращаем порт, соответствующий этому протоколу
    @res["Auth-Port"] = MailAuth::Port[proto]
    @log.info("a mail from #{from} on #{helo} for #{to}")
#пытаемся выполнить аутентификацию с помощью информации
# из полученных заголовков
elsif auth(user. pass)
    @res["Auth-Status"] = "OK"
    @res["Auth-Server"] = @mailhost
    # возвращаем порт, соответствующий этому протоколу
    @res["Auth-Port"] = MailAuth::Port[proto]
    # если используется APOP, то пароль нужно вернуть в открытом виде
    if meth == 'apop' && proto == 'pop3'
        @res["Auth-User"] = user
        @res["Auth-Pass"] = pass
    end
    @log.info("+ #{user} from #{client}")
#       попытка аутентификации не удалась
else
  # в случае ошибки аутентификации возвращаем соответствующий ответ
  @res["Auth-Status"] = "Invalid login or password"
  # и устанавливаем время в секундах, которое клиент должен
  # ждать перед следующей попыткой
  @res["Auth-Wait"] = "3"
  # Можно также установить код ошибки, который следует вернуть
  # SMTP-клиенту
  @res["Auth-Error-Code"] = "535 5.7.8"
  @log.info("! #{user} from #{client}")
end
end
```

Следующая секция объявлена закрытой (`private`), поскольку находящиеся в ней методы может вызывать только сам содержащий их класс. Метод `auth` - рабочая лошадка службы аутентификации, он проверяет корректность имени пользователя и пароля. Метод

method_missing служит для обработки недопустимых HTTP-методов и возвращает ошибку Not Found (Не найдено):

```
  private

    # наш метод аутентификации; измените в соответствии со своим окружением
    def auth(user. pass)
        # просто возвращает значение, соответствующее ключу
        if @auths.key?("#{user}:#{pass}")
            @mailhost = @auths["#{user}:#{pass}"]
            return @mailhost
        # если такого ключа нет, метод возвращает false
        else
            return false
        end
    end

    # на случай, если какой-нибудь другой процесс попытается обратиться к
    # этой службе и отправит запрос методом, отличным от GET

        def method_missing(env)
    @res.status = 404

  end
end # класс MailAuthHandler
end # модуль MailAuth
```

И в последней секции настраиваем сервер, так чтобы запросы к URI /auth маршрутизировались правильному обработчику:

```
# настраиваем каркас Rack
use Rack::ShowStatus
# сопоставляем URI /auth наш обработчик аутентификации

    map "/auth" do
    run MailAuth::Handler.new
    end
```

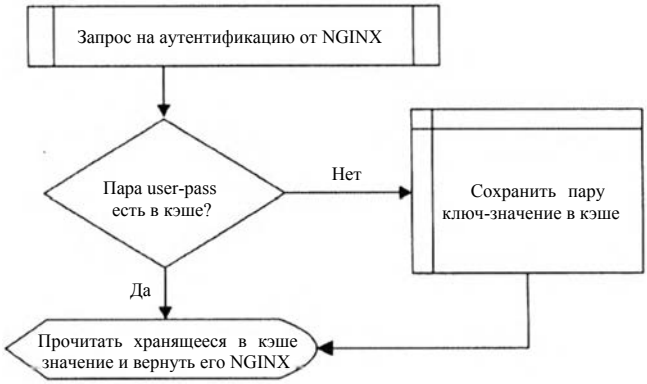
Этот код нужно сохранить в файле nginx_mail_proxy_auth.ru и вызывать с флагом -p <port>, который говорит, какой порт он должен прослушивать. Дополнительные сведения о каркасе Rack, реализующем интерфейс к веб-серверу, смотрите на сайте http://rack.github.com.

Использование в связке с memcached

Если клиенты часто обращаются к почтовым службам через ваш прокси-сервер или службе аутентификации доступно не слишком много ресурсов, то имеет смысл включить в инфраструктуру слой кэ-

ширования. Для этой цели мы воспользуемся программой `memcached`, которая будет кэшировать результаты аутентификации в памяти.

NGINX может искать ключ в `memcached`, но только в контексте определенного местоположения в модуле `http`. Поэтому мы должны реализовать собственный слой кэширования вне NGINX.



Как видно из блок-схемы, мы сначала проверяем, есть ли данная комбинация имени пользователя и пароля в кэше. Если нет, мы запрашиваем информацию у службы аутентификации и помещаем пару ключ-значение в кэш. В противном случае эти данные можно взять прямо из кэша.

Компания Zimbra разработала модуль `memcached` для NGINX, который делает все то же самое непосредственно в контексте NGINX. Но пока этот код не включен в официальный дистрибутив NGINX.

Показанный ниже код дополняет нашу исходную службу аутентификации слоем кэширования (в данном случае это, пожалуй, перебор, но этот код может служить основой для работы с базой данных для аутентификации, размещенной в сети).

```
# gem install memcached (зависит от библиотек libsasl2 и gettext)
require 'memcached'

# здесь задается IP адрес и порт, на котором работает memcached
@cache = Memcached.new("localhost:11211")
```

```
def get_cache_value(user, pass)
  resp = ''
  begin
    # сначала смотрим, есть ли ключ в кэше
  resp = @cache.get("#{user}:#{pass}")
    rescue Memcached::NotFound
    # если нет, вызываем метод auth
    resp = auth(user, pass)
    # и сохраняем полученный результат в кэше с ключом 'user:pass'
  @cache.set("#{user}:#{pass}",resp)
  end
  # явно возвращаем ответ вызывающей программе
  return resp
  end
```

Чтобы использовать этот код, необходимо, конечно, установить и запустить `memcached`. Для вашей операционной системы, скорее всего, имеется готовый пакет:

- Linux (на основе Debian)
  `sudo apt-get install memcached`
- Linux (на основе rpm)
  `sudo yum install memcached`
- FreeBSD
  `sudo pkg_add -r memcached`

Для настройки `memcached` нужно просто указать параметры при запуске. Никакого конфигурационного файла, читаемого непосредственно `memcached`, не существует, хотя операционная система может включать файл, который служит для упрощения передачи параметров.

Ниже перечислены наиболее важные параметры `memcached`:

- `-l`: задает адрес (или адреса), прослушиваемый `memcached` (по умолчанию любой). Важно отметить, что в целях безопасности `memcached` не должен прослушивать адрес, доступный из Интернета, потому что никакой аутентификации не предусмотрено.
- `-m`: задаем объем оперативной памяти, отводимой под кэш (в мегабайтах).
- `-c`: задает максимальное количество одновременных соединений (по умолчанию 1024).
- `-p`: задает порт, прослушиваемый `memcached` (по умолчанию 11211).

Задав разумные значения этих параметров, остается запустить `memcached`.

Теперь, заменив строку `elsif auth(user, pass)` строкой `elsif get_ cache_ value(user, pass)` в файле `nginx_mail_proxy_auth.ru`, мы получим службу аутентификации со слоем кэширования, который ускоряет обработку запросов.

Интерпретация журналов

Файлы журналов - лучшее средство понять, что происходит, когда система работает не так, как ожидается. Объем записываемой в журнал информации зависит от указанного в конфигурационном файле уровня подробности и от того, была ли NGINX собрана с поддержкой отладки (параметр `--enable-debug`).

В каждой строке журнала указывается один из уровней серьезности, сконфигурированных в директиве `error_log`. Определены следующие уровни (в порядке возрастания серьезности): `debug`, `info`, `notice`, `warn`, `error`, `crit`, `alert` и `emerg`. Если задан некоторый уровень, то в журнал помещаются сообщения этого и более высоких уровней. По умолчанию предполагается уровень `error`.

Для модуля `mail` обычно задается уровень протоколирования `info`, чтобы можно было получать столько информации, сколько возможно без вывода отладочных сообщений. Отладочное протоколирование в данном случае дало бы только сведения о точках входа в функции и паролях, указанных для соединений.

Поскольку почта сильно зависит от правильности функционирования DNS, многие ошибки могут быть связаны с некорректными записями DNS или устареванием информации в кэше. Если вы подозреваете, что ошибка вызвана именно неправильным разрешением имен, то можете узнать у NGINX, какой IP-адрес был сопоставлен конкретному доменному имени, включив отладочное протоколирование. К сожалению, для этого понадобится перекомпилировать двоичный файл `nginx`, если он изначально не был скомпилирован с поддержкой отладки.

Ниже описана типичная последовательность записей журнала, относящихся к сеансу по протоколу POP3.

Сначала клиент устанавливает соединение с прокси-сервером:

```
<timestamp> [info] <worker pid>#0: *<connection id> client <ip
address> connected to 0.0.0.0:110
```

После успешного входа выводится сообщение, содержащее все сведения о соединении:

```
<timestamp> [info] <worker pid>#0: *<connection id> client logged
in, client: <ip address>, server: 0.0.0.0:110, login: "<username>",
upstream: <upstream ip>:<upstream port>, [<client ip>:<client port>-
<local ip>:110] <=> [<local ip:<high port>-<upstream ip>:<upstream
port>]
```

Отметим, что часть перед двусторонней стрелкой `<=>` относится к стороне клиент-прокси, а часть после стрелки - к стороне прокси-почтовый сервер. Эта информация выводится еще раз при завершении сеанса.

```
<timestamp> [info] <worker pid>#0: *<connection id> proxied session
done, client: <ip address>, server: 0.0.0.0:110, login: "<username>",
upstream: <upstream ip>:<upstream port>, [<client ip>:<client port>-
<local ip>:110] <=> [<local ip:<high port>-<upstream ip>:<upstream
port>]
```

Таким образом, мы видим, какие порты использованы на всех сторонах соединения, что помогает отлаживать потенциальные ошибки и при необходимости коррелировать записи в этом журнале и в журнале брандмауэра.

Остальные записи журнала на уровне `info` относятся к таймаутам и некорректным командам либо ответам, полученным от клиента либо от проксируемого сервера.

На уровне `warn` обычно выводятся сообщения об ошибках конфигурации:

```
<timestamp> [warn] <worker pid>#0: *<connection id> "starttls"
directive conflicts with "ssl on"
```

Многие сообщения, выводимые на уровне `error`, свидетельствуют об ошибках при работе со службой аутентификации. Из следующих сообщений понятно, в каком состоянии находилось соединение в момент ошибки:

```
<timestamp> [error] <worker pid>#0: *<connection id> auth http
server 127.0.0.1:9000 timed out while in http auth state, client:
<client ip>, server: 0.0.0.0:25
<timestamp> [error] <worker pid>#0: *<connection id> auth http server
127.0.0.1:9000 sent invalid response while in http auth state, client:
<client ip>, server: 0.0.0.0:25
```

Если по какой-либо причине на запрос об аутентификации по получен ответ, соединение разрывается. NGINX не знает, какому серверу передать запрос клиента, поэтому может только закрыть соединение с сообщением `Internal server error` и кодом ответа, зависящим от протокола.

В журнал помещается запись, которая может содержать или не содержать имя пользователя. Ниже показано сообщение для SMTP-соединения, требующего аутентификации:

```
<timestamp> [error] <worker pid>#0: *<connection id> auth http server
127.0.0.1:9000 did not send server or port while in http auth state,
client: <client ip>, server: 0.0.0.0:25, login: "<login>"
```

Отметим, что в двух предыдущих сообщениях информация об имени пользователя (`login`) отсутствовала.

Сообщение уровня `alert` означает, что NGINX не смогла установить ожидаемое значение параметра, но в остальном работает нормально.

Напротив, сообщения уровня `emerg` означают, что NGINX не смогла запуститься; необходимо либо устранить причину ошибки, либо изменить конфигурацию. Если NGINX уже запущена, то она не станет перезапускать рабочие процессы, пока не будет произведено изменение:

```
<timestamp> [error] <worker pid>#0: *<connection id> no "http_auth" is
defined for server in /opt/nginx/conf/nginx.conf:32
```

В данном случае мы должны определить службу аутентификации с помощью директивы `http_auth`.

Ограничения операционной системы

Вы можете оказаться в ситуации, когда NGINX работает не так, как ожидается - то соединения рвутся, то в журнале появляются предупреждения. В этих случаях необходимо хорошо понимать, какие ограничения налагает на NGINX операционная система и как настроить ее для оптимальной работы сервера.

У почтового прокси-сервера проблемы чаще всего связаны с ограничением количества соединений. Чтобы понять, что это значит, следует разобраться, как NGINX обрабатывает запросы на установление соединения от клиентов. Главный процесс NGINX запускает

несколько отдельных рабочих процессов. Каждый процесс способен обработать фиксированное число соединений, определяемое директивой `worker_connections`. Для каждого проксируемого соединения NGINX открывает новое соединение с почтовым сервером. Для любого соединения необходим файловый дескриптор и по одному TCP-порту из диапазона эфемерных портов для каждой пары (IP-адрес, порт почтового сервера) (см. объяснение ниже).

Количество открытых файловых дескрипторов в зависимости от операционной системы настраивается в файле ресурсов или путем отправки сигнала демону, управляющему ресурсами. Чтобы узнать текущее значение, введите следующую команду оболочки:

```
ulimit -n
```

Если ваши расчеты показывают, что этот порог слишком мал или в журнале ошибок появляется сообщение `worker_connections exceed open file resource limit`, значит, значение необходимо увеличить. Сначала настройте максимальное число открытых файловых дескрипторов на уровне операционной системы - только для пользователя, от имени которого работает NGINX, или глобально. Затем в директиве `worker_rlimit_nofile` в главном контексте файла `nginx.conf` укажите новое значение. Чтобы измененное значение вступило в силу, достаточно послать `nginx` сигнал перезагрузки конфигурации (HUP); перезапускать главный процесс при этом не нужно.

Если вы столкнулись с нехваткой TCP-портов, то следует расширить диапазон эфемерных портов. Из этого диапазона операционная система выбирает порты для исходящих соединений. По умолчанию может быть всего 5000 портов, но обычно их бывает 16384. Хорошее описание процедуры расширения этого диапазона в различных операционных системах приведено па странице http://www.ncftp.com/ncftpd/doc/misc/ephemeral_ports. html.

Резюме

В этой главе мы показали, как настраивать NGINX для проксирования соединений по протоколам POP3, IMAP и SMTP. Каждый протокол можно настроить по отдельности, объявив о поддержке тех или иных возможностей проксируемым сервером. Можно шифровать трафик по протоколу TLS при наличии у сервера подходящего SSL-сертификата.

Для функционирования почтового модуля необходима служба аутентификации, поскольку без нее никакое проксирование невозможно. Мы привели подробный пример такой службы, описав, что ожидается в запросе и как следует формировать ответ. Опираясь на этот фундамент, вы сможете написать службу аутентификации, отвечающую вашим потребностям.

Умение интерпретировать журналы - один из самых важных навыков администратора. NGINX формирует достаточно детальные, хотя и не сразу понятные, журналы. Но зная, в какие моменты жизни соединения выводятся различные сообщения и в каком состоянии находится NGINX в каждый такой момент, нетрудно понять, что происходит.

NGINX, как и любая другая программа, работает в контексте операционной системы. Поэтому крайне важно знать, как ослабить ограничения, которые ОС налагает на NGINX. Если дальнейшее ослабление невозможно, то следует искать другое архитектурное решение - увеличить количество серверов, па которых работает NGINX, или применить какую-то другую технику, позволяющую уменьшить количество соединений, обрабатываемых одним экземпляром.

В следующей главе мы узнаем, как настроить NGINX для проксирования HTTP-соединений.

Глава 4. NGINX как обратный прокси-сервер

Обратным прокси-сервером называется веб-сервер, который является оконечной точкой соединения с клиентом и открывает новое соединение с проксируемым сервером от имени клиента. **Проксируемый сервер** определяется, как сервер, с которым NGINX устанавливает соединение для выполнения запроса клиента. Проксируемые серверы могут принимать разные формы, и для каждой NGINX можно настроить по-разному.

Настройка NGINX, которую мы как раз и изучаем, не всегда бывает простой. Для решения внешне похожих задач применяются различные директивы. Некоторые параметры вообще не следует использовать, поскольку это может привести к неожиданным результатам.

Иногда проксируемый сервер не может выполнить запрос. NGINX умеет доставлять клиенту сообщения об ошибках - непосредственно от проксируемого сервера, со своего локального диска или путем переадресации на страницу, размещенную совсем на другом сервере.

В силу самой природы обратного прокси-сервера проксируемый сервер не получает информацию непосредственно от клиента. Часть этой информации, например истинный IP-адрес клиента, важна для отладки и для отслеживания. Ее можно передать проксируемому серверу в виде заголовков.

Мы рассмотрим эти вопросы, а также дадим обзор некоторых директив из модуля прокси-сервера в следующих разделах.

- Введение в технологию обратного проксирования.
- Типы проксируемых серверов.
- Преобразование конфигурации с «if» в более современную форму.

- Использование документов с описанием ошибок для обработки ошибок проксирования.
- Определение истинного IP-адреса клиента.

Введение в технологию обратного проксирования

NGINX может выступать в роли обратного прокси-сервера, который является оконечной точкой соединения с клиентом и открывает новое соединение с проксируемым сервером. По ходу дела запрос можно проанализировать, исходя из URI-адреса, параметров клиента или иной логики, чтобы лучше обслужить его. Любая часть исходного URL запроса при прохождении через обратный прокси-сервер может быть преобразована.

С той зрения проксирования, наиболее важна директива `proxy_ pass`. Она принимает один параметр - URI-адрес, на который следует передать запрос. Использование `proxy_pass` с указанием URI заменяет `request_uri` этой частью. Так, в следующем примере `/uri` будет заменено на `/newuri` при передаче запроса проксируемому серверу:

```
location /uri {
    proxy_pass http://localhost:8080/newuri;
}
```

Но из этого правила есть два исключения. Во-первых, если местоположение определено с помощью регулярного выражения, то преобразование URI не производится. В примере ниже URI `/local` будет передан далее без изменения, а не заменен на `/foreign`, как, вероятно, хотел автор:

```
location ~ ^/local {
    proxy_pass http://localhost:8080/foreign;
}
```

Второе исключение состоит в том, что если внутри местоположения есть правило переписывания, которое изменяет URI, и затем NGINX использует этот URI для обработки запроса, то преобразование не производится. В примере ниже проксируемому серверу передается URI `/index.php?page=<match>`, где `<match>` - запомненное подвыражение в скобках, а не `/index`, как указано в части URI директивы `proxy_pass`:

```
location / {
  rewrite /(.*)$ /index.php?page=$1 break;
  proxy_pass http://localhost:8080/index;
}
```

Здесь флаг `break` используется для того, чтобы немедленно прекратить обработку директив модуля `rewrite`.

В обоих случаях часть URI после доменного имени в директиве `proxy_pass` несущественна, поэтому ее можно опустить без ущерба для полноты конфигурации:

```
location ~ ^/local {
  proxy_pass http://localhost:8080;
}

location / {
  rewrite /(.*)$ /index.php?page=$1 break;
  proxy_pass http://localhost:8080;
}
```

Модуль proxy

В следующей таблице перечислены некоторые наиболее употребительные директивы модуля `proxy`.

Директивы модуля proxy

Директива	Описание
`proxy_connect_timeout`	Максимальное время ожидания соединения с проксируемым сервером
`proxy_cookie_domain`	Подменяет атрибут `domain` в заголовке `Set-Cookie` от проксируемого сервера; можно указать строку, регулярное выражение или ссылку на переменную
`proxy_cookie_path`	Подменяет атрибут path в заголовке `Set-Cookie` от проксируемого сервера; можно указать строку, регулярное выражение или ссылку на переменную
`proxy_headers_hash_bucket_size`	Максимальный размер имен заголовков
`proxy_headers_hash_max_size`	Общий размер заголовков, полученных от проксируемого сервера
`proxy_hide_header`	Список заголовков, которые не следует передавать клиенту

Директива	Описание
proxy_http_version	Версия протокола HTTP, которой следует придерживаться при взаимодействии с проксируемым сервером (для соединений типа keepalive должна быть 1.1)
proxy_ignore_client_abort	Если значение равно on, то NGINX не станет разрывать соединение с проксируемым сервером в случае, когда клиент разрывает свое соединение
proxy_ignore_headers	Какие заголовки можно игнорировать при обработке ответа от проксируемого сервера
proxy_intercept_errors	Если значение равно on, то NGINX будет отображать страницу, заданную директивой error_page, вместо ответа, полученного от проксируемого сервера
proxy_max_temp_file_size	Максимальный размер временного файла, в который записывается часть ответа в случае, когда он не умещается целиком в буферах памяти
proxy_pass	Задает URL проксируемого сервера, которому передается запрос
proxy_pass_header	Отменяет сокрытие заголовков, определенных в директиве proxy_hide_header, разрешая передавать их клиенту
proxy_pass_request_body	Если значение равно off, то тело запроса не передается проксируемому серверу
proxy_pass_request_headers	Если значение равно off, то заголовки запроса не передаются проксируемому серверу
proxy_read_timeout	Сколько времени может пройти между двумя последовательными операциями чтения данных от проксируемого сервера, прежде чем соединение будет закрыто. Значение следует увеличить, если проксируемый сервер обрабатывает запросы медленно
proxy_redirect	Перезаписывает заголовки Location и Refresh, полученные от проксируемого сервера: полезно для обхода допущений, принятых каркасом разработки приложений
proxy_send_timeout	Сколько времени может пройти между двумя последовательными операциями записи данных на проксируемый сервер, прежде чем соединение будет закрыто
proxy_set_body	Эта директива позволяет изменить тело запроса, отправляемое проксируемому серверу

Директива	Описание
`proxy_set_header`	Перезаписывает заголовки, отправляемые проксируемому серверу. Может также применяться для подавления некоторых заголовков (если в качестве значения указать пустую строку)
`proxy_temp_file_write_size`	Ограничивает размер данных в одной операции записи во временный файл, чтобы NGINX не слишком долго блокировала исполнение программы при обработке одного запроса
`proxy_temp_path`	Каталог для хранения временных файлов, получаемых от проксируемого сервера. Может быть многоуровневым

Ниже многие из перечисленных выше директив собраны воедино. Этот фрагмент можно вставить в конфигурационный файл в то место, где находится директива `proxy_pass`.

Содержимое файла `proxy.conf`:

```
proxy_redirect off;
proxy_set_header Host $host;
proxy_set_header X-Real-IP $remote_addr;
proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
client_max_body_size 10m;
client_body_buffer_size 128k;
proxy_connect_timeout 30;
proxy_send_timeout 15;
proxy_read_timeout 15;
proxy_send_lowat 12000;
proxy_buffer_size 4k;
proxy_buffers 4 32k;
proxy_busy_buffers_size 64k;
proxy_temp_file_write_size 64k;
```

Мы задали в этих директивах значения, которые, на наш взгляд, полезны для настройки обратного прокси-сервера.

- В директиве `proxy_redirect` задано значение `off`, потому что переписывать заголовок `Location` в большинстве случаев не нужно.
- Заголовок `Host` устанавливается так, чтобы проксируемый сервер мог сопоставить запрос с виртуальным сервером или как-то иначе использовать часть переданного клиентом URL-адреса, содержащую имя хоста.

- Заголовки `X-Real-IP` и `X-Forwarded-For` служат аналогичным целям - передать проксируемому серверу информацию об IP-адресе подключившегося клиента.
  - Переменная `$remote_addr`, подставляемая в заголовок `x-Real-IP`, содержит IP-адрес клиента, определенный NGINX.
  - Переменная `$proxy_add_x_forwarded_for` содержит значение заголовка `x-Forwarded-For` из запроса клиента, за которым следует значение переменной `$remote_addr`.
- Директива `client_max_body_size`, строго говоря, не является директивой модуля `proxy`, но упомянута здесь, поскольку имеет прямое отношение к конфигурации прокси-сервера. Если ее значение слишком мало, то загруженные клиентом файлы не попадут проксируемому серверу. Задавая значение этой переменной, помните, что размер файла, загружаемого с помощью веб-формы, обычно оказывается больше, чем видно в файловой системе.
- Директива `proxy_connect_timeout` определяет, как долго NGINX будет ждать установления соединения с проксируемым сервером.
- Директивы `proxy_read_timeout` и `proxy_send_timeout` задают максимальное время между последовательными операциями чтения от проксируемого сервера и записи на него.
- Директива `proxy_send_lowat` применима только в ОС FreeBSD и определяет, сколько байт должно скопиться в буфере отправки сокета, чтобы данные были переданы протоколу.
- Директивы `proxy_buffer_size`, `proxy_buffers` и `proxy_busy_buffers_size` будут подробно рассмотрены в следующей главе. Пока скажем лишь, что они определяют видимую быстроту реакции NGINX на запросы пользователя.
- Директива `proxy_temp_file_write_size` определяет, сколько времени рабочий процесс может блокировать исполнение при записи данных во временный файл: чем больше значение, тем дольше процесс блокируется.

Эти директивы можно включить в конфигурационный файл, как показано ниже, и использовать многократно:

```
location / {
  include proxy.conf;
  proxy_pass http://localhost:8080;
}
```

Если какая-то директива в конкретном случае должна иметь другое значение, то его можно переопределить после включения файла:

```
location /uploads {
  include proxy.conf;
  client_max_body_size 500m;
  proxy_connect_timeout 75;
  proxy_send_timeout 90;
  proxy_read_timeout 90;
  proxy_pass http://localhost:8080;
}
```

В данном случае порядок важен. Если одна и та же директива встречается в конфигурационном файле (с учетом включаемых файлов) несколько раз, то NGINX возьмет последнее значение.

Унаследованные серверы с куками

Бывает, что несколько унаследованных приложений необходимо разместить за одной общей оконечной точкой. Унаследованные приложения написаны в предположении, что они взаимодействуют непосредственно с клиентом. Они устанавливают в куках собственное доменное имя и считают, что к ним всегда возможен доступ по адресу /. Если поместить перед этими серверам новую оконечную точку, то эти предположения окажутся неверными. В следующем фрагменте конфигурационного файла атрибуты `domain` и `path` в куках переписываются с учетом новой оконечной точки:

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

  location /legacy1 {
    proxy_cookie_domain legacy1.example.com app.example.com;
    proxy_cookie_path $uri /legacy1$uri;
    proxy_redirect default;
    proxy_pass http://legacy1.example.com/;
}
```

Значение переменной `$uri` уже включает начальную косую черту (/), так что дублировать ее не нужно.

```
  location /legacy2 {
    proxy_cookie_domain legacy2.example.org app.example.com;
    proxy_cookie_path $uri /legacy2$uri;
```

```
    proxy_redirect default;
    proxy_pass http://legacy2.example.org/;
  }

  location / {
    proxy_pass http://localhost:8080;
  }
```

Модуль upstream

С модулем `proxy` тесно связан модуль `upstream`. Директива `upstream` начинает новый контекст, в котором определяется группа проксируемых серверов. Этим серверам можно приписать разные веса (чем выше вес, тем больше соединений будет передано конкретному проксируемому серверу), они могут иметь разные типы (TCP или в домене UNIX). Их даже можно пометить как остановленные на техническое обслуживание.

В следующей таблице перечислены директивы, допустимые в контексте `upstream`.

Директивы модуля upstream

Директива	Описание
`ip_hash`	Обеспечивает равномерное распределение клиентских соединений по всем серверам за счет хеширования IP-адреса по его сети класса C
`keepalive`	Количество соединений с проксируемыми серверами, кэшируемых в одном рабочем процессе. При использовании с HTTP-соединениями значение `proxy_http_version` должно быть равно `1.1`, а значение `proxy_set_header - Connection ""`
`least_conn`	Активирует алгоритм балансировки нагрузки, согласно которому очередной запрос передается серверу с наименьшим числом активных соединений
`server`	Определяет адрес (доменное имя или IP-адрес с необязательным номером TCP-порта или путь к сокету в домене UNIX) и необязательные параметры проксируемого сервера, а именно: • `weight`: относительный вес сервера: • `max_fails`: максимальное число неудачных попыток установления связи с сервером в течение времени `fail_timeout`, после которого сервер помечается как неработоспособный; • `fail_timeout`: время, в течение которого сервер должен ответить на запрос, и время, в течение которого сервер считается неработоспособным; • `backup`: такой сервер получает запрос, только если все остальные неработоспособны: • `down`: помечает сервер как непригодный для обработки запросов

Кэширование соединений

Директива `keepalive` заслуживает особого упоминания. Она говорит, сколько соединений с проксируемым сервером NGINX должна держать открытыми в каждом рабочем процессе. Кэш соединений полезен в случае, когда NGINX должна постоянно держать некоторое количество открытых соединений с проксируемым сервером. Если проксируемый сервер работает по протоколу HTTP, то NGINX может задействовать механизм постоянных соединений, специфицированный в версии HTTP/1.1, для поддержания таких открытых соединений.

```
upstream apache {
  server 127.0.0.1:8080;
  keepalive 32;
}
  location    /    {
  proxy_http_version 1.1.
  proxy_set_header Connection ""
  proxy_pass http://apache;
  }
```

Здесь мы говорим, что хотели бы держать 32 открытых соединения с Apache, работающим на порту 8080 сервера `localhost`. NGINX должна будет выполнить процедуру квитирования TCP только для первых 32 соединений в каждом рабочем процессе, а затем может оставить эти соединения открытыми, не посылая заголовок `Connection` со значением `close`. В директиве `proxy_http_version` мы указываем, что хотели бы работать с проксируемым сервером по протоколу HTTP/1.1. Мы также очищаем значение заголовка `Connection` с помощью директивы `proxy_set_header`, чтобы не проксировать соединения с клиентом самостоятельно.

Если для обслуживания запросов потребуется больше 32 соединений, то NGINX их, конечно, откроет. Но по достижении порогового значения NGINX будет закрывать редко используемые соединения, чтобы общее число открытых соединений оставалось равным 32, как указано в директиве `keepalive`.

Этот механизм применим и для проксирования соединений по протоколам, отличным от HTTP. В примере ниже говорится, что NGINX должна поддерживать 64 соединения с двумя экземплярами `memcached`:

```
upstream memcaches {
  server 10.0.100.10:11211;
  server 10.0.100.20:11211;
  keepalive 64;
}
```

Изменение алгоритма балансировки нагрузки с подразумеваемого по умолчанию циклического (round-robin) на `ip_hash` или `least_conn` должно быть задано до директивы `keepalive`:

```
upstream apaches {
  least_conn;
  server 10.0.200.10 80;
  server 10.0.200.20:80;
  keepalive 32;
}
```

Алгоритмы балансировки нагрузки

Для выбора проксируемого сервера, которому передается очередной запрос, модуль upstream может применять один из трех алгоритмов балансировки нагрузки: циклический, по хеш-коду IP-адреса или с наименьшим количеством соединений. По умолчанию подразумевается **циклический** (round-robin) алгоритм, для его активации никакой директивы не нужно. В этом случае выбирается сервер, следующий за тем, который был выбран для обслуживания предыдущего запроса, - с учетом следования серверов в конфигурационном блоке и их весов. Циклический алгоритм пытается обеспечить справедливое распределение трафика, основываясь на понятии очередности.

Алгоритм хеширования IP-адреса, активируемый директивой `ip_hash`, основан па предположении, что запросы от клиентов с некоторыми IP-адресами должны попадать одному и тому же проксируемому серверу. В качестве ключа хеширования NGINX берет первые три октета IPv4-адреса или весь IPv6-адрес. Таким образом, множеству близких IP-адресов всегда сопоставляется один и тот же проксируемый сервер. Цель этого механизма - обеспечить не справедливое распределение, а постоянство связи между клиентом и обслуживающим его сервером.

Третий из поддерживаемых модулем `upstream` алгоритмов балансировки нагрузки, **с наименьшим количеством соединений**, выбирается директивой `least_conn`. Он ставит целью равномерное распределе-

ние нагрузки между проксирусмыми серверами путем выбора того, у которого количество активных соединений наименьшее. Различия в вычислительной мощности проксируемых серверов можно учесть с помощью параметра `weight` директивы `server`. При выборе сервера с наименьшим количеством соединений алгоритм принимает во внимание вес.

Типы проксируемых серверов

Проксируемым называется сервер, которому NGINX передает запрос на соединение. Он может находиться на другой физической или виртуальной машине, но это необязательно. Проксируемый сервер может быть демоном, прослушивающим сокет в домене UNIX на локальной машине, или одним из многих демонов, прослушивающих порты TCP па другой машине. Это может быть сервер Apache с модулями для обработки запросов различных типов или сервер промежуточного уровня Rack, предоставляющий HTTP-интерфейс к приложениям, написанным на Ruby. В любом случае NGINX можно настроить как прокси-сервер.

Единственный проксируемый сервер

Веб-сервер Apache часто применяется для обслуживания как статических файлов, так и интерпретируемых скриптов разных типов. Наличие в Сети обширной документации и пособий помогает пользователям быстро настроить свою любимую CMS (систему управления содержимым). К сожалению, из-за ограничений на ресурсы Apache в типичной конфигурации не может обслужить много одновременных запросов. NGINX, напротив, спроектирована для обслуживания именно такого трафика и потребляет очень мало ресурсов. Поскольку большинство CMS заранее сконфигурированы для работы с Apache и опираются на использование файла `.htaccess` для дополнительной настройки, то задействовать сильные стороны NGINX проще всего, используя ее в качестве прокси-сервера к экземпляру Apache:

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

Это самая простая из всех возможных конфигураций прокси-сервера. NGINX служит оконечной точкой для всех клиентских соединений и проксирует запросы на порт 8080 на локальном компьютере. Предполагается, что Apache настроен на прослушивание порта `localhost:8080`.

Подобная конфигурация как правило дополняется директивами, позволяющими NGINX самостоятельно обслуживать запросы на статические файлы, а остальные передавать Apache:

```
server {
  location / {
    try_files $uri @apache;
  }

  location @apache {
    proxy_pass http://127.0.0.1:8080;
  }
}
```

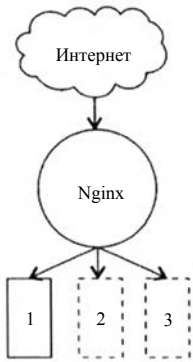
Директива `try_fiies` (относящая к базовому модулю `http`) по очереди проверяет файлы, пока не найдет совпадение. Так, в показанном выше примере NGINX сама доставит файлы, которые соответствуют URI в запросе клиента и находятся в ее корневом каталоге. Если файл не найден, то NGINX передаст запрос Apache для дальнейшей обработки. Мы воспользовались именованным местоположением, чтобы проксировать запрос после неудачной попытки найти файл локально.

Несколько проксируемых серверов

Есть также возможность настроить NGINX так, чтобы передавать запросы нескольким проксируемым серверам. Для этого следует объявить контекст `upstream`, определить в нем несколько серверов и сослаться на этот контекст в директиве `proxy_pass`:

```
upstream app {
  server 127.0.0.1:9000;
  server 127.0.0.1:9001;
  server 127.0.0.1:9002;


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

При такой конфигурации NGINX будет передавать поступающие запросы циклически трем проксируемым серверам. Это полезно, когда приложение способно в каждый момент времени обрабатывать только один запрос, и мы хотим поручить NGINX взаимодействие с клиентами, чтобы ни один из серверов приложений не был перегружен. Эта конфигурация изображена на следующем рисунке:



Проксируемые серверы

В разделе «Алгоритмы балансировки нагрузки» выше в этой главе описаны и другие алгоритмы балансировки нагрузки. Какой из них выбрать, зависит от конкретных обстоятельств.

Если некоторый клиент должен всегда попадать на один и тот же проксируемый сервер, то есть требуется обеспечить некое слабое подобие липких сеансов, то следует воспользоваться директивой `ip_hash`. В случае, когда распределение запросов характеризуется широким разбросом времени обработки одного запроса, то лучше выбрать алгоритм `least_conn`. Подразумеваемый по умолчанию циклический алгоритм хорош для общего случая, когда не требуется учитывать особенности клиента или проксируемого сервера.

Проксируемые серверы, работающие по протоколу, отличному от HTTP

До сих пор мы говорили только о проксируемых серверах, работающих по протоколу HTTP. Для них используется директива `proxy_pass`. Но как отмечалось выше в разделе «Кэширование соеди-

нений», NGINX может проксировать запросы серверам различных типов. Каждому типу соответствует своя директива `*_pass`.

Проксируемые серверы memcached

Модуль NGINX `memcached` (по умолчанию включенный) отвечает за взаимодействие с демоном `memcached`. При этом клиент не взаимодействует с `memcached` напрямую, то есть NGINX выступает в роли обратного прокси-сервера. Модуль `memcached` наделяет NGINX способностью общаться по протоколу `memcached`, то есть производить поиск ключа до передачи запроса серверу приложений.

```
upstream memcaches {
  server 10.0.100.10:11211;
  server 10.0.100.20:11211;
  }
  server {
    location / {
    set $memcached_key "$uri?$args";
    memcached_pass memcaches;
      errorpage 404 = @appserver;
    }
    location @appserver {
      proxy_pass http://127.0.0.1:8080;
    }
  }
```

В директиве `memcached.pass` переменная `$memcached_key` используется для формирования ключа поиска. Если соответствующего значения не найдено (`error_page 404`), то запрос передается компьютеру `localhost`, на котором предположительно работает сервер, способный обработать этот запрос и вставить пару ключ-значение в кэш, обслуживаемый экземпляром `memcached`.

Проксируемые серверы FastCGI

Использование сервера FastCGI - широко распространенный способ запуска PHP-приложеиий, работающих за сервером NGINX. Модуль `fastcgi` компилируется по умолчанию и активируется директивой `fastcgi_pass`. В результате NGINX получает возможность взаимодействовать по протоколу FastCGI с одним или несколькими проксируемыми серверами. Набор таких серверов определяется следующим образом:

```
upstream fastcgis {
  server 10.0.200.10:9000
  server 10.0.200.20: 9000
  server 10.0.200.30:9000
  }
```

А вот как им передаются запросы к корню:

```
location / {
  fastcgi_pass fastcgis;
}
```

Это очень примитивная конфигурация, но принцип использования FastCGI она демонстрирует. Модуль `fastcgi` располагает многочисленными директивами и возможностями настройки, которые мы будем обсуждать в главе 6.

Проксируемые серверы SCGI

NGINX поддерживает также протокол SCGI за счет встроенного модуля `scgi`. Принцип такой же, как для модуля `fastcgi`. NGINX передает запросы проксируемому серверу, указанному в директиве `scgi_pass`.

Проксируемые серверы uWSGI

Протокол uWSGI когда-то был очень популярен в среде разработчиков на языке Python. NGINX поддерживает подключение к проксируемому серверу с приложениями па Python с помощью модуля `uwsgi`. Конфигурируется он так же, как модуль `fastcgi`, только для указания проксируемого сервера служит директива `uwsgi_pass`. Пример приведен в главе 6.

Преобразование конфигурации с «if» в более современную форму

Применение директивы `if` внутри секции `location` считается оправданным только в некоторых ситуациях. Ее можно использовать в сочетании с директивами `return` и `rewrite` с флагом `last` или `break`, но в остальных случаях лучше избегать. Объясняется это возможностью получить неожиданные результаты. Рассмотрим такой пример:

```
location / {
  try_files /img /static @imageserver;
  if ($request_uri ~ "/blog") {
    proxy_pass http://127.0.0.1:9000;
    break;
  }

  if ($request_uri ~ "/tickets") {
    proxy_pass http://tickets.exanple.com;
    break;
  }
}
  location @imageserver {
  proxy_pass http://127.0.0.1:8080;
}
```

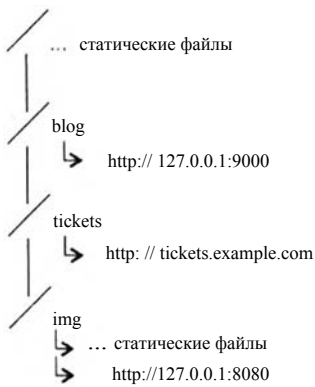
Здесь мы пытаемся определить, какому проксируемому серверу передать запрос, основываясь на значении переменной `$request_uri`. На первый взгляд, конфигурация кажется естественной и разумной, потому что в простых тестах работает правильно. Однако изображения не обслуживаются ни из местоположения `/img` в файловой системе, ни из местоположения /static там же, ни из именованного местоположения `@imageserver`. Директива `try_fiies` попросту не работает, когда в том же самом местоположении присутствует директива `if`. Директива `if` создает собственное неявное местоположение со своим обработчиком содержимого, в данном случае - модулем `proxy`. Таким образом, внешний обработчик содержимого, в котором находится директива `try_files`, вообще никогда не вызывается. Чтобы добиться желаемого, эту конфигурацию следует переписать иначе.

Подумаем, как NGINX обрабатывает запрос. Найдя соответствие IP-адресу и порту, NGINX сначала выбирает виртуальный сервер, основываясь на заголовке Host. Затем она просматривает все определенные для него местоположения в поисках подходящего URI. Поэтому, чтобы сконфигурировать селектор па основе URI, лучше просто определить несколько местоположений, как показано в примере ниже:

```
location /blog {
  proxy_pass http://127.0.0.1:9000;
}

location /tickets {
  proxy_pass http://tickets.example.com;
```

```
}

location /img {
  try_files /static @imageserver;
}

location / {
  root /static;
}
  location  @imageserver  {
proxy_pass http://127.0.0.1 8080;
}
```

Эта конфигурация иллюстрируется на рисунке ниже.



Вот еще один пример конфигурации с директивами `if`:

```
server {

  server_name marketing.example.com communication.example.com
  marketing.example.org communication.example.org
  marketing.example. net communication.example.net;

  if ($host ~* (marketing\.example\.com|marketing\.example\.
  org|marketing\.example\.net)) {
    rewrite ^/$ http://www.example.com/marketing/application.do
  redirect;
  }
  if ($host ~* (communication\.example\.com|communication\.example\.
```

```
  org|communication\.example\.net)) {
    rewrite ^/$ http://www.example.com/comms/index.cgi redirect;
  if ($host ~* (www\.example\.org|www\.example\.net)) {
    rewrite ^/(.*)$ http://www.example.com/$1 redirect;
  }
}
```

Здесь имеется ряд директив `if`, сравнивающих значение заголовка Host (или, если его нет, значение `server_name`) с различными регулярными выражениями. После обнаружения соответствия URI переписывается с целью перехода на соответствующий компонент приложения. Мало того что сопоставление каждого URI с несколькими регулярными выражениями крайне неэффективно, так еще и нарушается правило «не должно быть if внутри location».

Такую конфигурацию лучше переписать в виде нескольких последовательных контекстов `server`, в каждом их которых URL подменяется адресом соответствующего компонента:

```
server {
  server_name marketing.example.com marketing.example.org
  marketing. example.net;
  rewrite ^ http://www.example.com/marketing/application.do permanent;
}
server {
  server_name communication.example.com communication.example.org
  communication.example.net;

  rewrite ^ http://www.example.com/comms/index.cgi permanent;
}
server {
  server_name www.example.org www.example.net;

  rewrite ^ http://www.example.com$request_uri permanent;
}
```

В каждом блоке мы оставили только те имена серверов, которые относятся к соответствующей директиве `rewrite`, поэтому директивы `if` не понадобились. Во всех правилах `rewrite` мы заменили флаг redirect флагом permanent, чтобы сообщить браузеру, что это полный URL-адрес, который нужно запомнить и автоматически использо-

вать при следующем обращении к этому домену. В последнем правиле `rewrite` мы также заменили сопоставление с регулярным выражением `(^/(.*)$)` сравнением с готовой переменной `$request_uri`, которая содержит ту же информацию, но избавляет от необходимости сопоставлять строку с регулярным выражением и запоминать полвыражение в переменной.

Использование документов с описанием ошибок для обработки ошибок проксирования

Бывает, что проксируемый сервер не может обработать запрос. В таких случаях NGINX можно настроить так, чтобы возвращался документ, хранящийся на локальном диске.

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

  location = /50x.html {
    root share/examples/nginx/html;
  }
}
```

Или на внешнем сайте:

```
server {
  error_page 500 http://www.example.com/maintenance.html;
}
```

При проксировании на несколько серверов можно определить дополнительный сервер «последней надежды» (fallback), который обрабатывает запросы, которые оказались не по зубам другим серверам. Это полезно, когда сервер последней надежды может вернуть ответ, зависящий от запрошенного URI:

```
upstream app {
  server 127.0.0.1:9000;
  server 127.0.0.1:9001;
  server 127.0.0.1:9002;
 }
server {
  location / {
```

```
    error_page 500 502 503 504 = @fallback;
    proxy_pass http://app;
  }

  location @fallback {
    proxy_pass http://127.0.0.1:8080;
  }
}
```

Знак "=" в строке `error_page` означает, что мы хотим вернуть код состояния, полученный от последнего параметра, в данном случае от местоположения `@fallback`.

В этих примерах рассматриваются случаи, когда код ошибки равен 500 или больше. NGINX может также возвращать `error_page` для кодов ошибки 400 или больше, если значение директивы `proxy_intercept_errors` равно `on`, как в следующем примере:

```
server {
  proxy_intercept_errors on;
  error_page 400 403 404 /40x.html;

  location = /40x.html {
    root share/examples/nginx/html;
  }
}
```

Если в директиве `error_page` присутствует код ошибки 401, то аутентификация окажется незавершенной. Это можно делать, например в случае, когда сервер аутентификации выведен из эксплуатации, но в общем случае не рекомендуется.

Определение истинного IP-адреса клиента

При использовании прокси-сервера у клиента нет прямого соединения с проксируемым сервером. Поэтому проксируемый сервер не может получать информацию непосредственно от клиента. Любую информацию, в том числе истинный IP-адрес клиента, следует передавать в заголовках. Для этой цели NGINX предоставляет директиву `proxy_set_header`:

```
proxy_set_header X-Real-IP $remote_addr;
proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
```

В результате IP-адрес клиента передается в каждом из заголовков `X-Real-IP` и `X-Forwarded-For`. Во втором случае принимается во внимание заголовок из запроса клиента. Если в запросе присутствует заголовок `X-Forwarded-For`, то IP-адрес клиента будет добавлен в конец этого заголовка, через запятую. В зависимости от конфигурации проксируемого сервера необходима одна из этих двух директив. Например, чтобы настроить Apache, так чтобы в его журналах в качестве IP-адреса клиента использовался заголовок `X-Forwarded-For`, нужно воспользоваться спецификатором форматирования `%{<имя-заголовка>}i`.

Ниже показано, как изменить подразумеваемый по умолчанию формат «объединенного» (combined) журнала Apache:

```
LogFormat "%{X-Forwarded-For}i %l %u %t \"%r\" %>s %b \"%{Referer}
i\" \"%{User-Agent}i\"" combined
```

С другой стороны, если проксируемый сервер требует нестандартного заголовка, например `Client-IP`, то и эту задачу легко решить следующим образом:

```
proxy_set_header Client-IP $remote_addr;
```

Точно так же проксируемым серверам можно передавать и другую информацию, например заголовок `Host`:

```
proxy_set_header Host $host;
```

Резюме

Мы рассмотрели вопрос об использовании XGINX в качестве обратного прокси-сервера. Ее эффективная модель обработки соединений идеальна для прямого взаимодействия с клиентами. Получив запрос, NGINX может затем открыть новое соединение с одним из нескольких проксируемых серверов, приняв во внимание сильные и слабые стороны каждого. Использование директивы `if` внутри секции `location` считается допустимым только в определенных ситуациях. Поняв, как NGINX обрабатывает запрос, мы можем подготовить конфигурацию, в большей степени отвечающую решаемой задаче. Если у NGINX по какой-то причине не получается установить соединение с проксируемым сервером, то она может вернуть альтернативную страницу. Поскольку NGINX служит оконечной точкой для

клиентских запросов, проксируемые серверы могут получать информацию о клиенте только с помощью заголовков запроса, проксированного NGINX. Описанные соображения помогут при написании конфигурационного файла NGINX, идеально отвечающего вашим потребностям.

В следующей главе мы рассмотрим более сложные приемы обратного проксирования.

Глава 5. Обратное проксирование, дополнительные вопросы

В предыдущей главе мы видели, что обратный прокси-сервер устанавливает соединения с проксируемыми серверами от имени клиентов. Сами проксируемые серверы не имеют прямой связи с клиентами. Так сделано по разным причинам, в том числе ради безопасности, масштабируемости и повышения производительности.

Обратный прокси-сервер обеспечивает безопасность, потому что если бы злоумышленник захотел попасть напрямую на проксируемый сервер, ему нужно было бы сначала взломать обратный прокси-сервер. Соединения с клиентом можно шифровать по протоколу HTTPS. Оконечной точкой таких SSL-соединений может быть обратный прокси-сервер, если проксируемый сервер не может или не должен предоставлять средства шифрования самостоятельно. NGINX может выступать в роли терминатора SSL-соединений, а также реализовывать дополнительные списки управления доступом и ограничения, зависящие от различных свойств клиента.

Масштабируемость достигается за счет того, что обратный прокси-сервер устанавливает параллельные соединения с несколькими проксируемыми серверами, которые клиентам представляются одним сервером. Если приложению нужна дополнительная вычислительная мощность, то пул серверов, расположенных за единственным обратным прокси-сервером, можно расширить.

Обратный прокси-сервер может повысить производительность приложения несколькими способами. Во-первых, он может кэшировать и сжимать содержимое перед отправкой его клиенту. В роли обратного прокси-сервера NGINX способна обрабатывать больше одновременных клиентских соединений, чем типичный сервер приложений. Иногда NGINX конфигурируют так, что она обслуживает

запросы на статическое содержимое из локального дискового кэша, а проксируемому серверу передает только динамические запросы. Клиенты могут оставлять соединения с NGINX открытыми, тогда как сама NGINX закрывает соединения с проксируемыми серверами немедленно, освобождая тем самым их ресурсы.

Мы обсудим все эти вопросы, а также оставшиеся директивы модуля proxy в следующих разделах.

- Безопасность за счет разделения.
- Обеспечение масштабируемости за счет изоляции компонентов приложения.
- Оптимизация производительности обратного прокси-сервера.

Безопасность за счет разделения

Мы можем повысить безопасность, отделив точку, в которой клиенты подключаются к приложению. Это одна из основных причин использования обратного прокси-сервера в архитектуре системы. Клиент напрямую подключается только к машине, на которой работает прокси-сервер. Эту машину необходимо как следует защитить, чтобы противник не мог найти точку проникновения в систему.

Безопасность - столь обширная тема, что мы лишь вкратце отметим, на что обратить внимание.

- Устанавливайте брандмауэр перед обратным прокси-сервером, открывая доступ только к порту 80 (и 443, если предполагаются также HTTPS-соединения).
- Запускайте NGINX от имени непривилегированного пользователя (в разных операционных системах он называется `www`, `webservd` или `www-data`).
- Шифруйте трафик для предотвращения перехвата.

О последней рекомендации мы подробнее поговорим в следующем разделе.

Шифрование трафика по протоколу SSL

NGINX часто используют как терминатор SSL-соединений - потому что проксируемый сервер не поддерживает SSL или с целью освободить его от накладных расходов, связанных с обработкой SSL-соединиеий. Для этого необходимо откомпилировать двоичный файл `nginx` с поддержкой SSL (параметр `--with_http_ssl_module`) и установить сертификат и ключ SSL.

Подробнее о том, как самостоятельно сгенерировать сертификат SSL, см. врезку «Генерация SSL-сертификата с помощью OpenSSL» в главе 3 «Почтовый модуль».

Ниже показан пример конфигурации для разрешения HTTPS-соединений с сервером `www.example.com`:

```
server {
  listen 443 default ssl;

  server_name www.example.com;

  ssl_prefer_server_ciphers on;
  ssl_protocols TLSv1 SSLv3;
  ssl_ciphers RC4:HIGH:!aNULL:!MD5:@STRENGTH;
  ssl_session_cache shared:WEB:10m;
  ssl_certificate /usr/local/etc/nginx/www.example.com.crt;
  ssl_certificate_key /usr/local/etc/nginx/www.example.com.key;

  location / {
    proxy_set_header X-FORWARDED-PROTO https;
    proxy_pass http://upstream;
  }
 }
```

Здесь мы сначала активируем модуль `ssl`, указав параметр `ssl` в директиве `listen`. Затем мы говорим, что хотим отдать предпочтение серверным, а не клиентским шифрам, чтобы иметь возможность использовать только шифры, признанные наиболее безопасными. Это не позволит клиенту предложить нерекомендуемый шифр. В директиве `ssl_session_cache` мы задаем режим `shared`, чтобы все рабочие процессы могли воспользоваться плодами дорогостоящего начального согласования параметров SSL, уже один раз выполненного для некоторого клиента. Несколько виртуальных серверов могут пользоваться одной и той же директивой `ssl_session_cache`, если все они сконфигурированы с одним и тем же именем или эта директива находится в контексте `http`. Вторая и третья часть значения - имя кэша и его размер соответственно. Далее заданы сертификат и ключ для данного хоста. Отметим, что для файла ключа следует установить права доступа так, чтобы читать его мог только главный процесс. Для заголовка `X-FORWARDED-PROTO` мы задали значение `https`, чтобы приложение, работающее на проксируемом сервере, знало о том, что исходный запрос был отправлен по протоколу HTTPS.

Шифры SSL

Шифры в показанной выше конфигурации выбираются из набора, подразумеваемого в NGINX по умолчанию. В него не входят шифры вообще без аутентификации (`aNULL`) и шифры, в которых используется алгоритм хеширования MD5. Шифр RC4 помещен в начало набора, поэтому предпочтение отдается шифрам, не подверженным атаке BEAST, которая описана в документе CVE-2011-3389. Строка `@STRENGTH` в конце значения говорит о том, что список шифров следует упорядочить по длине ключа алгоритма шифрования.

Выше мы описали шифрование трафика между клиентом и обратным прокси-сервером. Но можно также шифровать трафик между прокси-сервером и проксируемым сервером:

```
server {
  …
proxy_pass https://upstream;
}
```

Обычно это применяется только в системах, где даже внутренняя сеть считается небезопасной.

Аутентификация клиентов по протоколу SSL

В некоторых приложениях используется информация, полученная из предъявленного клиентом сертификата SSL, но в системе с обратным прокси-сервером она непосредственно недоступна. Чтобы передать ее приложению, необходимо, чтобы NGINX добавила заголовок:

```
location /ssl {
  proxy_set_header ssl_client_cert $ssl_client_cert;
  proxy_pass http://upstream;
}
```

В переменной `$ssl_client_cert` хранится сертификат клиента в формате PEM. Мы передаем его проксируемому серверу в одноименном заголовке. Далее приложение может использовать эту информацию, как ему угодно.

Вместо того чтобы передавать проксируемому серверу весь клиентский сертификат, NGINX может самостоятельно проделать часть

работы и убедиться, что сертификат хотя бы корректен, то есть подписан известным удостоверяющим центром (УЦ), не отозван и дата окончания срока действия еще не наступила.

```
server {
  …
  ssl_client_certificate /usr/local/etc/nginx/ClientCertCAs.pem;
  ssl_crl /usr/local/etc/nginx/ClientCertCRLs.crl;
  ssl_verify_client on;

  ssl_verify_depth 3;
  error_page 495 = @noverify;

  error_page 496 = @nocert;
    location @noverify {
  proxy_pass http://insecure?status=notverified;

  }
    location @nocert {
  proxy_pass http://insecure?status=nocert;

  }
    location / {
      if ($ssl_client_verify = FAILED) {
    return 495;
    }
  proxy_pass http://secured;
  }
}
```

Показанный выше фрагмент, в котором предварительно проверяется корректность клиентского сертификата SSL, состоит из следующих частей.

- В аргументе директивы `ssl_client_certificate` указывается путь к представленному в формате PEM списку сертификатов корневых УЦ, которые могут подписывать клиентские сертификаты.
- Аргумент директивы `ssl_crl` определяет путь к списку отозванных сертификатов, который публикуется удостоверяющим центром, подписывающим клиентские сертификаты. Этот список необходимо периодически загружать с помощью отдельной процедуры.
- Директива `ssl_verify_client` говорит, что NGINX должна проверять корректность SSL-сертификатов, предъявленных клиентами.

- Директива `ssl_verify_depth` определяет максимальное количество сторон, подписавших сертификат; если их больше, сертификат признается недействительным. SSL-сертификат может быть подписан одним или несколькими промежуточными УЦ. Сертификат промежуточного УЦ или сертификат подписавшего его корневого УЦ должен присутствовать в пути, указанном в директиве `ssl_client_certificate`, иначе NGINX сочтет его недействительным.
- И случае ошибки при проверке клиентского сертификата NGINX возвращает нестандартный код ошибки 495. Мы определили для этого кода страницу ошибки `error_page`, которая переадресует запрос в именованное местоположение, где он будет обработан отдельным проксируемым сервером. Мы также включили проверку переменной `$ssl_client_venfy` внутри корневого местоположения, чтобы для некорректного сертификата возвращался такой же код.
- Если сертификат некорректен, то NGINX возвращает нестандартный код ошибки 496, который мы также перехватываем с помощью директивы `error_page`, указывающей на именованное местоположение, где запрос передается отдельному обработчику ошибок.

Только в том случае, когда клиент предъявил корректный SSL-сертификат, NGINX передаст запрос проксируемому серверу. При таком подходе мы можем быть уверены, что проксируемый сервер получает запросы только от аутентифицированных пользователей. Это важная часть механизма защиты, реализуемого обратным прокси-сервером.

Начиная с версии 1.3.7, NGINX поддерживает протокол OCSP для проверки сертификатов клиентов. О том, как активировать эту функцию, см. описание директив `ssl_stapling*` и `ssl_trusted_certificate` в приложении А.

Если приложению все-таки нужна какая-то информация из клиентского сертификата, например, чтобы авторизовать клиента, то NGINX может передать ее в заголовке:

```
location / {
  proxy_set_header x-HTTP-AUTH $ssl_client_s_dn;
  proxy_pass http //secured;
}
```

Теперь приложение, работающее па защищенном проксируемом сервере, может проверить значение в заголовке `X-HTTP-AUTH` и предоставить клиенту доступ к тем или иным возможностям. Переменная `$ssl_client_s_dn` содержит отличительное имя (DN) субъекта, хранящееся в клиентском сертификате. Зная его, приложение может справиться с базой данных или поискать пользователя в LDAP-каталоге.

Блокирование трафика на основе IP-адреса отправителя

Поскольку обратный прокси-сервер является оконечной точкой для клиентских соединений, то можно ограничить количество клиентов, исходя из их IP-адресов. Это полезно в случае обнаружения попытки недобросовестного использования, когда очень много недопустимых обращений исходят от клиентов с IP-адресами из некоторого ограниченного множества. Как и в языке Perl, решить задачу можно несколькими способами. Ниже мы обсудим применение для этой цели модуля `GeoIP`.

Двоичный файл `nginx` должен быть откомпилирован с поддержкой модуля `GeoIP` (параметр `-- with-http_geoip_module`), а в системе должна быть установлена библиотека MaxMind GeoIP. Путь к откомпилированной базе данных указывается в директиве `geoip_country` в контексте `http`. Это самый эффективный способ запретить или разрешить IP-адреса по коду страны:

```
geoip_counu y /usr/local/etc/geo/GeoIP.dat;
```

Если IP-адрес клиента присутствует в базе данных, то в переменной `$geoip_country_code` будет находиться двухбуквенный код соответствующей страны по классификатору ISO.

Мы воспользуемся данными, полученными от модуля GeoIP, применив также модуль с похожим названием `geo`. Модуль `geo` предоставляет очень простой интерфейс для установки переменных на основе IP-адреса клиента. Он создает именованный контекст, в котором первый параметр - проверяемый IP-адрес, а второй - значение, присваиваемое переменной при обнаружении соответствия. Сочетание обоих модулей позволит нам блокировать IP-адреса из некоторых стран, разрешив в то же время доступ с некоторых IP-адресов.

Допустим, мы представляем услугу швейцарским банкам. Мы хотим, чтобы Google мог индексировать открытые разделы сайта, по

при этом разрешить доступ к ним только с IP-адресов в Швейцарии. Кроме того, мы хотим, чтобы локальная сторожевая служба могла получить доступ к сайту и удостовериться в его работоспособности. Мы определяем переменную `$exclusions`, по умолчанию равную 0. Если какое-нибудь из сформулированных выше условий выполняется, то переменная получит значение 1, означающее, что доступ к сайту разрешен:

```
http {
  #путь к базе данных GeoIP
  geoip_country /usr/local/etc/geo/GeoIP.dat;

  #определяем переменную $exclusions и перечисляем все IP-адреса,
  #которым разрешен доступ, задавая для них значение 1
  geo $exclusions {

    default 0;
    127.0.0.1 1;
    216.239.32.0/19 1;
    64.233.160.0/19 1;
    66.249.80.0/20 1;
    72.14.192.0/18 1;
    209.85.128.0/17 1;
    66.102.0.0/20 1;
    74.125.0.0/16 1;
    64.18.0.0/20 1;
    207.126.144.0/20 1;
    173.194.0.0/16 1;

  }

    server {
    # страна, которой доступ разрешен. - Швейцария, с кодом "CH"

       if ($geoip_country_code = "CH") {
         set $exclusions 1;
       }

     location / {
     # если IP-адрес не принадлежит Швейцарии и не находится в
     # нашем списке, то переменная $exclusions равна "0", и мы
     # возвращаем HTTP-код "Forbidden"

          if ($exclusions = "0" ) {

             return 403;

          }

     # всем остальным клиентам разрешен доступ к проксируемому серверу
        proxy_pass http://upstream;
     }
  }
}
```

Это лишь один из способов решить задачу блокирования доступа к сайту на основе IP-адреса клиента. Можно также сохранять IP-адрес в каком-нибудь хранилище ключей и значений, увеличивать счетчик при каждом запросе и блокировать доступ, если количество запросов в течение определенного периода превысило пороговое значение.

Обеспечение масштабируемости за счет изоляции компонентов приложения

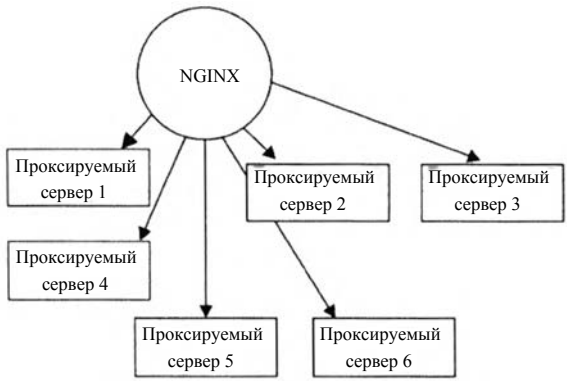
Масштабирование приложение бывает двух видов: горизонтальное и вертикальное. Под вертикальным масштабированием понимается наращивание ресурсов компьютера с целью обслужить больше клиентских запросов. Горизонтальное масштабирование - это расширение пула доступных машин, обслуживающих клиентов, чтобы ни одна из них не была перегружена запросами. Этот подход зачастую является более рентабельным вне зависимости от того, что представляют собой эти машины - визуализированные экземпляры, работающие в облаке, или физические компьютеры, размещенные в центре обработки данных. Именно в такой инфраструктуре NGINX находит свое место в качестве обратного прокси-сервера.

Благодаря чрезвычайно низкому ресурсопотреблению NGINX идеальна в роли брокера клиентских приложений. NGINX обрабатывает соединения с клиентами и при этом способна одновременно обслуживать несколько запросов. В зависимости от конфигурации NGINX либо возвращает файл из своего локального кэша, либо передает запрос проксируемому серверу для дальнейшей обработки. В роли проксируемого может выступать любой сервер, поддерживающий протокол HTTP, При таком подходе удается обслужить больше запросов, чем если бы проксируемый сервер отвечал напрямую:

```
upstream app {
  server 10.0.40.10;
  server 10.0.40.20;
  server 10.0.40.30;
}
```

Со временем начальный набор проксируемых серверов можно расширять. Допустим, что количество обращений к сайту настоль-

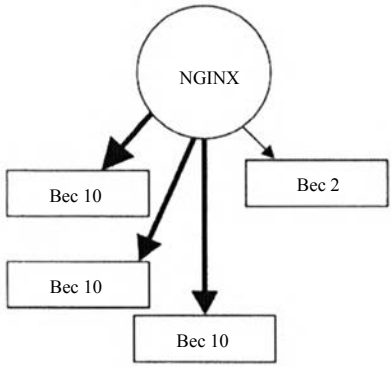
ко увеличилось, что текущий набор перестал с ними справляться. Если использовать NGINX в качестве обратного прокси-сервера, то с этой бедой легко справиться, добавив дополнительные проксируемые серверы.



Добавление проксируемых серверов производится следующим образом:

```
upstream app {
  server 10.0.40.10;
  server 10.0.40.20;
  server 10.0.40.30;
  server 10.0.40.40;
  server 10.0.40.50;
  server 10.0.40.60;
}
```

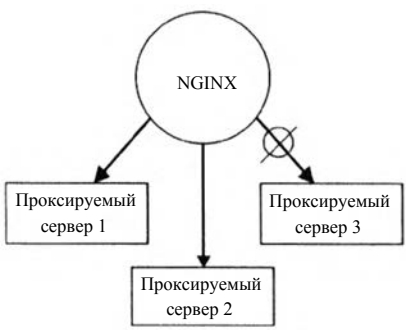
Быть может, настало время переписать приложение или перенести его на сервер с другим прикладным стеком. Прежде чем переносить все приложение, мы можем поместить один сервер в активный пул для тестирования в условиях реальной нагрузки с реальными клиентами. Этому серверу можно передавать меньше запросов, чтобы смягчить негативную реакцию пользователей в случае возникновения ошибок.



Это делается с помощью такой конфигурации:

```
upstream app {
  server 10.0.40.10 weight 10;
  server 10.0.40.20 weight 10;
  server 10.0.40.30 weight 10;
  server 10.0.40.100 weight 2;
}
```

С другой стороны, пусть требуется вывести проксируемый сервер из эксплуатации для проведения планового обслуживания. Если пометить его признаком `down` в конфигурационном файле, то NGINX перестанет передавать ему запросы, и мы сможем запяться обслуживанием:



Ниже показано, как это делается:

```
upstream app {
  server 10.0.40.10;
  server 10.0.40.20;
  server 10.0.40.30 down;
}
```

Система должна быстро реагировать в случае, когда сервер перестает отвечать на запросы. Для некоторых приложений в директивах таймаута можно задать очень малое время:

```
location / {
  proxy_connect_timeout 5;
  proxy_read_timeout 10;
  proxy_send_timeout 10;
}
```

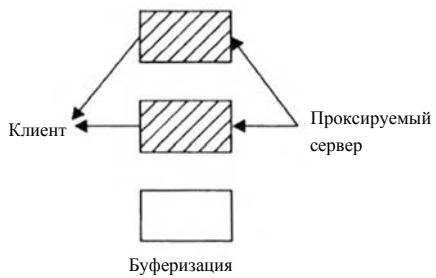
Но будьте осторожны: если ни один проксируемый сервер не сможет ответить в отведенное время, то NGINX вернет ошибку **504 Gateway Timeout Error**.

Оптимизация производительности обратного прокси-сервера

NGINX можно по-разному настраивать, чтобы выжать максимум возможного из приложения, для которого оно служит обратным прокси-сервером. За счет буферизации, кэширования и сжатия NGINX может существенно ускорить время реакции приложения на действия клиента.

Буферизация

Смысл буферизации изображен на следующем рисунке:



С точки зрения производительности проксирования, самым важным фактором является буферизация. По умолчанию NGINX пытается прочитать от проксируемого сервера столько данных, сколько возможно, и так быстро, как возможно, прежде чем вернуть ответ клиенту. Она локально буферизует ответ, чтобы можно было отправить его клиенту одним разом. Если часть запроса от клиента или ответа от проксируемого сервера записывается на диск, то производительность может снизиться. Это компромисс между оперативной и дисковой памятью. Поэтому, настраивая NGINX для работы в качестве обратного прокси-сервера, так важно уделять внимание описанным ниже директивам.

Директивы модуля proxy для управления буферизацией

Директива	Описание
`proxy_buffer_size`	Размер буфера, используемого для первой части ответа от проксируемого сервера, в которой находятся заголовки
`proxy_buffering`	Включает или выключает буферизацию проксированного содержимого. Если буферизация выключена, то ответы отправляются клиенту синхронно сразу после получения при условии, что параметр `proxy_max_temp_file_size` равен 0. Если этот параметр равен 0 и `proxy_buffering` равно `on`, то буферизация включена, но диск при этом не используется
`proxy_buffers`	Количество и размер буферов для хранения ответов от проксируемых серверов
`proxy_busy_buffers_size`	Суммарный размер буферов, которые могут быть заняты для отправки ответа клиенту, пока ответ ещё не прочитан целиком. Обычно устанавливается в два раза больше, чем размер, указанный в директиве `proxy_buffers`

Помимо рассмотренных выше директив, на буферизацию может оказать влияние проксируемый сервер - путем установки заголовка `X-Accel-Buffenng`. По умолчанию значение в этом заголовке равно yes, то есть ответы буферизуются. Задание значения по полезно для приложений, основанных на модели Comet, и в случае потоковой передачи данных по протоколу HTTP, когда буферизовать отпет не нужно.

Измеряя средний размер запросов и ответов, проходящих через обратный прокси-сервер, можно оптимально настроить размеры буферов. Каждая из директив буферизации относится к одному соединению. Принимая во внимание также накладные расходы ОС

на установление соединения, мы можем вычислить, сколько одновременных соединений с клиентами можно поддержать при имеющемся объеме оперативной памяти.

Подразумеваемое по умолчанию значение директивы `proxy_buffers` (`8 4k` или `8 8k` в зависимости от операционной системы) рассчитано на большое число одновременных соединений. Посмотрим, на сколько именно. На типичной машине, где запущена NGINX, объем памяти составляет 1 ГБ, и большая ее часть может быть отдана NGINX. Сколько-то памяти необходимо отвести под кэш файловой системы и другие нужды, поэтому примем осторожную оценку: 768 МБ.

Восемь буферов по 4 КБ займут 32 768 байтов (8 * 4 * 1024) на каждое активное соединение.

Мы отвели под NGINX 768 МБ, то есть 805 306 368 байтов (768 * 1024 * 1024).

Поделив одно на другое, получаем 805 306 368 / 32 768 = = 24 576 активных соединений.

Таким образом, в конфигурации по умолчанию NGINX сможет одновременно обслужить чуть меньше 25 000 соединений в предположении, что буферы постоянно заполнены. Нужно учитывать и другие факторы, например кэширование содержимого и простаивание соединений, но грубую оценку мы получили.

Если теперь принять показанные ниже средние размеры запроса и ответа, то окажется, что восьми буферов по 4КБ недостаточно для обработки типичного запроса. Мы хотим, чтобы NGINX буферизовала столько полученных от проксируемого сервера данных, сколько возможно; конечная цель - передать пользователю сразу весь ответ.

- Средний размер запроса: 800 байтов.
- Средний размер ответа: 900 КБ.

В примерах настройки, приведенных ниже, используется больше памяти ценой уменьшения количества одновременных активных соединений. Это оптимизации, которые не следует рассматривать как рекомендации в общем случае. NGINX уже оптимизирована для обслуживания большого числа медленных клиентов и нескольких быстрых проксируемых серверов. В настоящее время наблюдается смещение в сторону мобильных пользователей, так что клиентское соединение оказывается существенно медленнее, чем при широкополосном доступе. Поэтому прежде чем приступать к оптимизации, нужно хорошо изучить, как именно пользователи подключаются.

Для этого нужно увеличить размеры буферов, чтобы ответ уместился в них целиком:

```
http {
  proxy_buffers 30 32k;
}
```

Это, конечно, означает, что количество одновременно обслуживаемых пользователей резко уменьшится.

Тридцать буферов по 32 КБ займут 983 040 байтов (30 * 32 * 1024) на каждое активное соединение.

Мы отвели под NGINX 768 МБ, то есть 805 306 368 байтов (768 * 1024 * 1024).

Поделив одно на другое, получаем 805 306 368 / 983 040 = = 819,2 активных соединений.

Совсем не много. Уменьшим количество буферов, так чтобы NGINX начала передавать данные клиенту, продолжая читать ответ в оставшиеся буферы.

```
http {
  proxy_buffers 4 32k;
  proxy_busy_buffers_size 64k;
}
```

Четыре буфера по 32 КБ займут 131 072 байтов (4 * 32 * 1024) на каждое активное соединение.

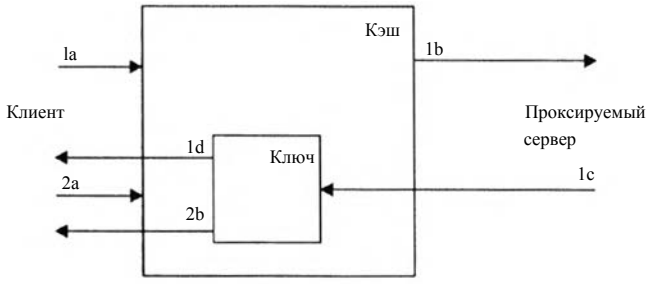
Мы отвели под NGINX 768 МБ, то есть 805 306 368 байтов (768 * 1024 * 1024).

Поделив одно на другое, получаем 805 306 368 / 131 072 = = 6144 активных соединений.

Что касается самой машины, где работает обратный прокси-сервер, то мы можем масштабировать ее либо вертикально, добавив память (при объеме ОЗУ 6 ГБ мы получим приблизительно 37 000 соединений), либо горизонтально, поставив за балансировщиком нагрузки столько дополнительных машин с ОЗУ объемом 1 ГБ, сколько необходимо для обслуживания ожидаемого количества одновременных пользователей.

Кэширование

Идея кэширования представлена на рисунке ниже.



Кэширование

NGINX может кэшировать ответы от проксируемого сервера, так что при повторном поступлении того же запроса не нужно будет снова обращаться к серверу. На рисунке показана следующая последовательность операций.

- **1a**: Клиент отправляет запрос.
- **1b**: Соответствующий запросу ключ не найден в кэше, поэтому NGINX обращается к проксируемому серверу.
- **1c**: Проксируемый сервер отвечает и NGINX помещает ответ в кэш.
- **1d**: Ответ отправляется клиенту.
- **2a**: Другой клиент отправляет запрос с таким же ключом кэширования.
- **2b**: NGINX обслуживает запрос непосредственно из кэша, не обращаясь за ответом к проксируемому серверу.

Директивы модуля proxy для управления кэшированием

Директива	Описание
`proxy_cache`	Определяет размер зоны разделяемой памяти, отведенной под кэш
`proxy_cache_bypass`	Одна или несколько строковых переменных. Если хотя бы одна из них непуста и не содержит нуль, то ответ будет запрошен у проксируемого сервера, а не взят из кэша
`proxy_cache_key`	Строка, которая используется как ключ для поиска значения в кэше. Можно использовать переменные, но следует внимательно следить за тем, чтобы не кэшировалось несколько копий одного и того же содержимого

Директива	Описание
`proxy_cache_lock`	Если эта директива принимает значение `on`, то предотвращается отправка нескольких запросов проксируемому серверу (или серверам) в случае отсутствия в кэше. NGINX дождется результата первого запроса, поместит его в кэш и только потом приступит к обслуживанию остальных. Эта блокировка действует в пределах одного рабочего процесса
`proxy_cache_lock_ timeout`	Сколько времени запрос может ждать появления записи в кэше или освобождения блокировки `proxy cache lock`
`proxy_cache_min_uses`	Сколько запросов с данным ключом должно поступить, прежде чем ответ будет помещен в кэш
`proxy_cache_path`	Каталог, в котором хранятся кэшированные ответы, и зона разделяемой памяти (`keys_zone=name: size`) для хранения активных ключей и метаданных ответа. Необязательные параметры: • levels: список разделенных двоеточиями длин имен подкаталогов на каждом уровне (`1` или `2`); допускается не более трех уровней вложенности; • inactive: максимальное время нахождения неактивного запроса в кэше, по прошествии которого он вытесняется; • `max_size`: максимальный размер кэша; по его достижении процесс-диспетчер удаляет элементы, к которым дольше всего не было обращений; • `loader_files`: максимальное количество кэшированных файлов, метаданные которых загружаются в кэш на одной итерации процесса-загрузчика; • `loader_sleep`: число миллисекунд между итерациями процесса-загрузчика кэша; • `loader_threshold`: максимальное время, отведенное на одну итерацию процесса-загрузчика кэша
`proxy_cache_use_stale`	В каких случаях допустимо использовать устаревшие кэшированные данные, если при доступе к проксируемому серверу произошла ошибка. Параметр `updating` разрешает использовать кэшированный ответ, если как раз в данный момент загружаются более свежие данные
`proxy_cache_valid`	Сколько времени считать действительным кэшированный ответ с кодом 200, 301 или 302. Если перед параметром `time` указан необязательный код ответа, то заданное время относится только к ответу с таким кодом. Специальный параметр `any` означает, что в течение заданного времени следует кэшировать ответ с любым кодом

Показанная ниже конфигурация предназначена для кэширования всех ответов в течение шести часов при общем размере кэша 1 ГБ. Актуальные элементы, то есть такие, к которым были обращения на протяжении шестичасового периода таймаута, остаются действительными сутки. По истечении этого времени у проксируемого сервера будет запрошен новый ответ. Если проксируемый сервер не отвечает из-за ошибки, произошел таймаут, получен недопустимый заголовок или кэшированный элемент обновляется, то разрешается использовать устаревший ответ из кэша. Размер области разделяемой памяти CACHE задан равным 10 МБ, ссылка на него производится из местоположения, для которого поддерживается кэширование.

```
http {
  # размещаемся в той же файловой системе, что и proxy_cache_path
  proxy_temp_path /var/spool/nginx;

  # из соображений безопасности этим каталогом владеет тот же
  # пользователь, который определен в директиве user (от имени
  # которого запущены рабочие процессы)
  proxy_cache_path /var/spool/nginx keys_zone=CACHE:10m levels=1:2
    inactive=6h max_size=1g;

  server {
    location / {
      # используем include, чтобы загрузить файл с общими параметрами
      include proxy.conf;

      # ссылаемся на определенную выше область разделяемой памяти
      proxy_cache CACHE;
      proxy_cache_valid any 1d;
      proxy_cache_use_stale error timeout invalid_header updating
      http_500 http_502 http_503 http_504,
      proxy_pass http://upstream;
    }
   }
  }
```

При такой конфигурации NGINX создаст ряд подкаталогов в каталоге `/var/spool/nginx`, которые распределяют кэшированные данные сначала по последнему символу MD5-свертки URI-адреса, а затем по следующим двум, считая от конца. Например, ответ на запрос к URI "/ this-is-a-typical-url" будет храниться в файле:

```
/var/spool/nginx/3/f1/614c16873c96c9db2090134be91cdf13
```

Помимо директивы `proxy_cache_valid`, кэшированием ответов в NGINX управляют еще некоторые заголовки. Значение заголовка имеет больший приоритет, чем директива.

- Проксируемый сервер может установить заголовок `X-Acce1-Expires`, управляющий поведением кэширования:
  - целочисленное значение в нем указывает, сколько секунд хранить ответ в кэше;
  - если значение равно 0, то кэширование этого ответа запрещено;
  - значение, начинающееся символом `@`, интерпретируется как время в секундах с начала «эпохи». Ответ хранится до наступления этого момента времени.
- У заголовков `Expires` и `Cache-Control` одинаковый приоритет.
- Если значение в заголовке `Expires` определяет момент в будущем, то ответ кэшируется до наступления этого момента.
- Заголовок `Cache-Control` может принимать следующие значения:
  - `no-cache`
  - `no-store`
  - `private`
  - `max-age`
- Ответ кэшируется, только если задано значение `max-age`, которое должно быть положительным числом, то есть `max-age=x`, где $x > 0$.
- Если присутствует заголовок `Set-Cookie`, то ответ не кэшируется. Это правило можно отменить, воспользовавшись директивой `proxy_ignore_headers`:
  `proxy_ignore_headers Set-Cookie`.
- Ко при этом не забудьте сделать значение кука частью ключа `proxy_cache_key`:
  `proxy_cache_key "$host$request_uri $cookie_user";`

Однако, поступая так, следует позаботиться о том, чтобы для одного и того URI-адреса не кэшировалось несколько ответов. Это может случиться, если для открытого содержимого случайно установлен заголовок `Set-Cookie`, который становится частью ключа доступа к данным. Один из способов обеспечить эффективное использование кэша - поместить открытое содержимое в отдельное местоположение. Например, изображения можно хранить в местоположении `/img`, для которого определен другой ключ `proxy_cache_key`:

```
server {
  proxy_ignore_headers Set-Cookie;

  location /img {
    proxy_cache_key "$host$request_uri";
    proxy_pass http://upstream,
  }

  location / {
    proxy_cache_key "$host$request_uri $cookie_user";
    proxy_pass http://upstream;
  }
}
```

Сохранение

С концепцией кэша тесно связано понятие **сохранения**. В случае, когда сервер возвращает большие статические файлы, которые никогда не изменяются, то есть нет никаких причин вытеснять записи из кэша, то NGINX предлагает механизм «сохранения» (store), чтобы ускорить обслуживание таких запросов. NGINX сохраняет локальные копии заданных вами файлов. Эти файлы остаются на диске и более не запрашиваются у проксируемого сервера. Если такой файл изменится на проксируемом сервере, то его сохраненную копию должен будет удалить какой-то внешний процесс, иначе NGINX продолжит возвращать ее. Поэтому для небольших статических файлов использование кэша предпочтительнее.

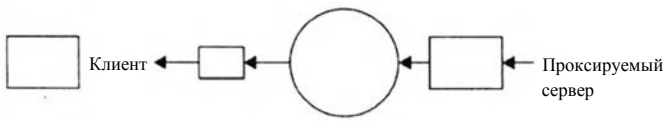
В следующем примере демонстрируются директивы, применяемые для сохранения копий файлов:

```
http {
  proxy_temp_path /var/www/tmp;

  server {
    root /var/www/data;

    location /img {
      error_page 404 = @store;
    }

    location @store {
      internal;
      proxy_store on;
      proxy_store_access group:r all:r;
      proxy_pass http://upstream;
    }
  }
}
```

В этой конфигурации определена секция `server`, в которой корень root размещен в той же файловой системе, что и путь `proxy_temp_path`. Местоположение `/img` наследует `root`, обслуживая файлы, расположенные в подкаталоге `/var/www/data`. Если файл не найден (код ошибки 404), то производится переход к именованному местоположению `@store` для запроса файла у проксируемого сервера. Директива `proxy_store` означает, что мы хотим сохранять файлы в унаследованном от `root` каталоге с правами 0644 (для владельца подразумеваются права `user:rw`, а для группы (`group`) и всех прочих (`all`) задаются в директиве `proxy_store_access`). Это все, что требуется от NGINX для сохранения локальных копий статических файлов, полученных от проксируемого сервера.

Сжатие

Идея сжатия представлена на рисунке ниже.



Сжатие

За счет снижения объема трафика можно уменьшить время передачи ответа. NGINX умеет сжимать ответ, полученный от проксируемого сервера, до передачи его клиенту. Модуль `gzip`, по умолчанию включенный, нередко используется обратным прокси-сервером для сжатия содержимого, когда это имеет смысл. Для некоторых типов файлов сжатие не дает ощутимого эффекта. С другой стороны, некоторые клиенты плохо реагируют на сжатое содержимое. То и другое можно учесть при настройке:

```
http {
  gzip on;
  gzip_http_version 1.0;
  gzip_comp_level 2;
  gzip_types text/plain text/css application/x-javascript text/
    xml application/xml application/xml+rss text/javascript application/
    javascript application/json;
  gzip_disable msie6;
}
```

Здесь мы говорим, что хотим сжимать содержимое с указанными типами MIME алгоритмом gzip с уровнем сжатия 2, если запрос поступил по протоколу HTTP версии не ниже 1.0, но только в том случае, когда пользовательский агент - не Internet Explorer старой версии. Мы поместили этот фрагмент конфигурации в контекст `http`, чтобы он распространялся на все определяемые серверы.

В таблице ниже перечислены директивы, относящиеся к модулю `gzip`.

Директивы модуля gzip

Директива	Описание
`gzip`	Разрешает или запрещает сжатие ответов
`gzip_buffers`	Определяет количество и размеры буферов для сжатия ответа
`gzip_comp_level`	Уровень сжатия gzip (1 -9)
`gzip_disable`	Регулярное выражение, описывающее те пользовательские агенты, которые не должны получать сжатое содержимое. Специальное значение `msie6` является сокращением выражения `MSIE [4-6]\.`, которое исключает `MSIE 6.0... SV1`
`gzip_min_length`	Минимальная длина ответа (определяемая заголовком `Content-length`), до которой вопрос о сжатии вообще не рассматривается
`gzip_proxied`	Разрешает или запрещает сжатие ответа, уже прошедшего через прокси-сервер. Параметр может принимать одно или несколько значений из следующего списка: • `off`: запретить сжатие; • `expired`: разрешить сжатие, если ответ не должен кэшироваться в соответствии с заголовком `Expires`; • `no-cache`: разрешить сжатие, если заголовок `Cache-Control` содержит значение `no-cache`; • `no-store`: разрешить сжатие, если заголовок `Cache-Control` содержит значение `no-store`; • `private`: разрешить сжатие, если заголовок `Cache-Control` содержит значение `private`; • `no-last-modified`: разрешить сжатие, если ответ не содержит заголовка `Last-Modified`; • `no-etag`: разрешить сжатие, если ответ не содержит заголовка `ETag`; • `auth`: разрешить сжатие, если ответ содержит заголовок `Authorization`; • `any`: разрешить сжатие для любого ответа, если запрос содержит заголовок `Via`
`gzip_types`	Типы MIME (в дополнение к `text/html`), которые следует сжимать
`gzip_vary`	Разрешает или запрещает включение в ответ заголовка `Vary: Accept-Encoding`, если директива `gzip` активна

Если при включенном сжатии большие файлы обрезаются, то причина, вероятно, в директиве `gzip_buffers`. По умолчанию она принимает значение `32 4k` или `16 8k` (в зависимости от платформы), то есть суммарный объем буферов равен 128 КБ. Это означает, что NGINX не может сжать файл длиннее 128 КБ. Если на страницу загружается большая несжатая библиотека JavaScript, то вы можете столкнуться с этим пределом. В таком случае просто увеличьте количество буферов, чтобы их общий размер был достаточен для размещения всего файла.

```
http {
  gzip on;
  gzip_min_length 1024;
  gzip_buffers 40 4k;
  gzip_comp_level 5;
  gzip_types text/plain application/x-javascript application/json;
}
```

В примере выше сжимать можно файлы размером до 40 * 4 * * 1024 = 163840 байтов (160 КБ). Кроме того, мы включили директиву `gzip_min_length`, которая разрешает сжимать только файлы, размер которых больше 1 КБ. Уровень сжатия (`gzip_comp_level`) 4 или 5 обычно даст хороший компромисс между скоростью сжатия и размером результирующего файла. Проведение измерений на собственном оборудовании - лучший способ определить оптимальное значение.

Помимо сжатия ответов на лету, NGINX умеет доставлять предварительно сжатые файлы - благодаря модулю `gzip_static`. По умолчанию этот модуль не компилируется, по его можно включить с помощью параметра `--with-http_gzip_static_module` скрипта configure. У этого модуля имеется единственная собственная директива `gzip_static`, но он также понимает следующие директивы модуля `gzip`, определяющие, когда проверять предварительно сжатые файлы:

- `gzip_http_version`
- `gzip_proxied`
- `gzip_disable`
- `gzip_vary`

В следующем примере конфигурации мы разрешаем доставку предварительно сжатых файлов, если запрос содержит заголовок `Authorization` и в ответе присутствует заголовок `Expires` или `Cache-Control`, запрещающий кэширование:

```
http {
  gzip_static on;
  gzip_proxied expired no-cache no-store private auth;
}
```

Резюме

В этой главе мы видели, как эффективно использовать NGINX в качестве обратного прокси-сервера. Она может решать три задачи, вместе или по отдельности: повышение безопасности, масштабируемости и (или) производительности. Безопасность повышается за счет отделения приложения от конечного пользователя. Для обеспечения масштабируемости NGINX можно использовать в сочетании с несколькими проксируемыми серверами. А производительность приложения непосредственно связана с тем, как быстро оно отвечает на запросы пользователей. Мы рассмотрели различные способы повысить быстроту реакции приложения. Чем меньше время ответа, тем лучше пользователям.

В следующей главе мы займемся вопросом об использовании NGINX в качестве HTTP-сервера. До сих пор мы говорили только о работе NGINX в роли обратного прокси-сервера, но она умеет гораздо больше.

Глава 6. NGINX как HTTP-сервер

HTTP-сервер - это программа, основная задача которой состоит в доставке веб-страниц клиентам в ответ на их запросы. Происхождение веб-страницы может быть каким угодно - от простого HTML-файла на диске до многокомпонентного каркаса, генерирующего зависящее от пользователя содержимое, которое динамически обновляется с помощью AJAX или WebSocket. NGINX - модульная программа, способная обслуживать HTTP-запросы любым необходимым способом.

В этой главе мы рассмотрим различные модули, которые в совокупности превращают NGINX в масштабируемый HTTP-сервер. Обсуждаются следующие вопросы.

- Архитектура NGINX.
- Базовый модуль HTTP.
- Установка предельных значений для предотвращения недобросовестного использования.
- Ограничение доступа.
- Потоковая передача мультимедийных файлов.
- Предопределенные переменные.
- Использование NGINX совместно с PHP-FPM.
- Интеграция NGINX и uWSGI.

Архитектура NGINX

NGINX состоит из одного главного и нескольких рабочих процессов. Все они однопоточные и способны одновременно обслуживать тысячи соединений. Большая часть работы производится рабочим процессом, поскольку именно он обрабатывает запросы клиентов. Чтобы как можно быстрее реагировать на запросы, NGINX использует встроенный в операционную систему механизм событий.

Главный процесс NGINX отвечает за чтение конфигурационного файла, работу с сокетами, запуск рабочих процессов, открытие файлов журналов и компиляции встроенных скриптов на языке Perl.

Он же реагирует на административные команды, передаваемые с помощью сигналов.

Рабочий процесс NGINX исполняет цикл событий, в котором обрабатываются входящие соединения. Все модули NGINX исполняются в рабочем процессе, то есть там производится обработка запросов, фильтрация, обработка соединений с прокси-сервером и многое другое. Такая модель позволяет операционной системе отделять рабочие процессы друг от друга и оптимально планировать их выполнение на различных процессорных ядрах. Если какие-то операции, например дискового ввода-вывода, блокируют один рабочий процесс, то нагрузка перемещается на рабочие процессы, исполняемые на других ядрах.

Главный процесс NGINX запускает еще несколько вспомогательных процессов для выполнения специализированных задач. Среди них **загрузчик кэша** и **диспетчер кэша**. Загрузчик кэша отвечает за подготовку метаданных, необходимых рабочим процессам для использования кэша, а диспетчер кэша - за проверку записей кэша и удаление тех, для которых истек срок хранения.

NGINX имеет модульную структуру. Главный процесс предоставляет основу для работы всех модулей. Протоколы и обработчики реализованы в виде отдельных модулей. Из отдельных модулей выстраивается конвейер обработки запросов. Поступивший запрос передается по цепочке фильтров, которые его обрабатывают. Один из таких фильтров предназначен для обработки подзапросов, этот механизм - одна из самых интересных возможностей NGINX.

С помощью подзапросов NGINX может вернуть ответ на запрос, URI которого отличается от указанного клиентом. Подзапросы могут быть вложенными и вызывать другие подзапросы. Фильтры позволяют собрать ответы на несколько подзапросов и составить от них ответ, отправляемый клиенту. По ходу дела в игру вступают различные модули. Подробное описание внутренней структуры NGINX см. на странице http://www.aosabook.org/en/nginx.html.

Далее в этой главе мы рассмотрим модуль http и несколько вспомогательных модулей.

Базовый модуль HTTP

Модуль http является центральным в NGINX, он отвечает за взаимодействие с клиентами но протоколу HTTP. В главе 2 «Руководство по настройке» мы уже обсуждали следующие аспекты этого модуля:

- клиентские директивы;
- директивы файлового ввода-вывода;
- директивы хеширования;
- директивы работы с сокетами;
- директива `listen`;
- сопоставление запроса с директивами `server_name` и `location`.

В этой главе мы рассмотрим оставшиеся директивы, тоже разбив их на категории.

Директива server

Директива server открывает новый контекст. Мы уже видели примеры ее использования на страницах этой книги. Но пока не уделили достаточно внимания понятию сервера по умолчанию.

В NGINX сервером по умолчанию называется первый из множества серверов, прослушивающих один и тот же IP-адрес и порт, указанные в директиве `listen`. Сервер по умолчанию можно также назначить с помощью параметра `default_server` в директиве `listen`.

Сервер по умолчанию полезен, когда нужно определить набор директив, общих для всех серверов, прослушивающих один и тот же IP-адрес и порт:

```
server {
  listen 127.0.0.1:80;
  server_name default.example.com;
  server_name_in_redirect on;
}

server {
  listen 127.0.0 1:80;
  server_name www.example.com;
}
```

В этом примере для сервера `www.example.com` будет действовать та же директива `server_name_in_redirect` с параметром `on`, что и для сервера `default.example.com`. Это было бы так, даже если бы в описания обоих серверов не было директивы `listen`, потому что тогда они бы все равно прослушивали один и тот же IP-адрес и порт (а именно *:80, подразумеваемые в директиве `listen` по умолчанию). Однако наследование не гарантируется. Существует лишь несколько наследуемых директив, и какие именно, меняется от версии к версии.

У сервера по умолчанию есть лучшее применение - обработка запросов, поступающих на данный IP-адрес и порт, но не имеющих

заголовка Host. Если вы не хотите, чтобы запросы без заголовка `Host` обрабатывал именно сервер но умолчанию, то можете задать в какой-нибудь секции `server` пустую директиву `server_name`. Тогда такие запросы будут поступать именно этому серверу.

```
server {
  server_name "";
}
```

В таблице ниже перечислены директивы, употребляемые в секции `server`.

Директивы HTTP-сервера

Директива	Описание
`port_in_redirect`	Определяет, нужно ли указывать данный порт при переадресации средствами NGINX
`server`	Создает новый конфигурационный контекст, определяющий виртуальный хост. Директива `listen` определяет один или несколько IP-адресов и портов, а директива `server_name` - значения заголовка `Host`, которым этот контекст соответствует
`server_name`	Задает имя виртуального хоста
`server_name_in_redirect`	Разрешает использовать первое из указанных в директиве `server_name` значений при любой переадресации, произведенной NGINX в данном контексте
`server_tokens`	Разрешает или запрещает включение номера версии NGINX в отправляемые сообщения об ошибках и заголовок `Server` (по умолчанию принимает значение `on`)

Протоколирование

В NGINX реализована очень гибкая модель протоколирования. На каждом уровне конфигурации может быть свой журнал доступа. Кроме того, на одном уровне можно задавать несколько журналов доступа с разными форматами с помощью директивы `log_format`. Эта директива, которая должна находиться в секции `http`, позволяет точно указать, что протоколировать.

Поскольку путь к файлу журнала может содержать переменные, то есть возможность задавать конфигурацию динамически. В примере ниже показано, как это может выглядеть на практике.

```
http {
  log_format vhost '$host $remote_addr - $remote_user [$time_local]'
                   '"$request" $status $body_bytes_sent '
                   '"$http_referer" "$http_user_agent"';
  log_format downloads '$time_iso8601 $host $remote_addr '
                       '"$request" $status $body_bytes_sent $request_time';
  open_log_file_cache max=1000 inactive=60s;
  access_log logs/access.log;

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

    access_log logs/combined.log vhost;
    access_log logs/$2/access.log;

    location /downloads {

      access_log logs/downloads.log downloads;

    }
  }
}
```

В таблице ниже описаны использованные в этом фрагменте директивы.

Директивы протоколирования из модуля HTTP

Директива	Описание
`access_log`	Определяет, куда и как записывать журналы доступа. Первый параметр - путь к файлу журнала. Путь может содержать переменные. Специальное значение `off` отключает протоколирование. Необязательный второй параметр определяет директиву `log_format`, описывающую формат журнала. Если этот параметр не задан, используется предопределенный формат. Необязательный третий параметр задает размер буфера в случае, если запись в журнал буферизуется. При использовании буферизации этот размер не должен превосходить длину операции атомарной записи на диск для конкретной файловой системы. Если третий параметр равен `gzip`, то журнал буферизуется и на лету сжимается при условии, что двоичный файл `nginx` собирался с библиотекой `zlib`. Последний параметр `flush` определяет, сколько времени данные могут оставаться в буфере в памяти перед сбросом на диск
`log_format`	Определяет состав и формат полей в журнале. Список переменных, относящихся к журналу, приведен в следующей таблице

Директива	Описание
`log_not_found`	Подавляет запись в журнал сообщений об ошибке 404 (по умолчанию принимает значение `on`)
`log_subrequest`	Разрешает или запрещает протоколирование подзапросов в журнале доступа (по умолчанию `off`)
`open_log_file_cache`	Кэширует дескрипторы тех открытых файлов, упоминаемых в директивах `access_log`, в путях к которым встречаются переменные. Допустимы следующие параметры: • `max`: максимальное число дескрипторов в кэше; • `inactive`: сколько времени ждать записи в этот журнал, перед тем как закрыть его; • `min_uses`: сколько раз этот дескриптор должен быть использован в течение времени `inactive`, чтобы оставаться открытым; • `valid`: NGINX будет часто проверять, по-прежнему ли этот дескриптор соответствует файлу с именем, по которому был открыт; • `off`: отключает кэширование.

В примере ниже журнал сжимается согласно алгоритму gzip с уровнем 4. Буфер имеет подразумеваемый по умолчанию размер 64 КБ п сбрасывается на диск не реже одного раза в минуту.

```
access_log /var/log/nginx/access.log.gz combined gzip=4 flush=1m;
```

Отметим, что при задании сжатия (`gzip`) параметр `log_format` также должен быть задан.

Подразумеваемая по умолчанию директива `log_format` с именем `combined` выглядит следующим образом:

```
log_format combined '$remote_addr - $remote_user [$time_local] '
                    '"$request" $status $body_bytes_sent '
                    '"$http_referer" "$http_user_agent"';
```

Как видите, для большей наглядности директиву можно разбить на строки. На самом форматировании это никак не сказывается. В описании формата разрешается использовать любые переменные. В таблице ниже переменные, помеченные звездочкой, можно использовать только в директиве `log_format`, а остальные - также в других местах конфигурационного файла.

Переменные, используемые при описании формата журнала

Переменная	Значение
`$body_bytes_sent`	Количество отправленных клиенту байтов, исключая заголовки
`$bytes_sent`	Общее количество отправленных клиенту байтов
`$connection`	Порядковый номер, уникально идентифицирующий соединение
`$connection_requests`	Количество запросов, поступивших по данному соединению
`$msec`	Время в секундах с точностью до миллисекунды
`$pipe *`	Был ли запрос обработан конвейером (`p`) или нет (`.`)
`$request_length *`	Длина запроса, включая HTTP-метод, URI-адрес, версию протокола HTTP, заголовки и тело
`$request_time`	Время обработки запроса с точностью до миллисекунды с момента получения от клиента первого байта запроса и до момента отправки клиенту последнего байта ответа
`$status`	Код состояния в ответе
`$time_iso8601 *`	Местное время в формате IS08601
`$time_local *`	Местное время в общепринятом формате журнала (`%d/%b/%Y:%H:%M:%S %z`)

В этом разделе мы говорили исключительно о настройке журнале доступа (`access_log`). По NGINX может также протоколировать ошибки. Директива `error_log` описана в главе 8 «Техника устранения неполадок».

Поиск файлов

Чтобы ответить на запрос, NGINX передает его обработчику содержимого, определяемому директивой `location`. Сначала апробируются безусловные обработчики: `perl`, `proxy_pass`, `flv`, `mp4` и т. д. Если ни один не подходит, то запрос передается одному из следующих обработчиков в указанном порядке: `random index`, `index`, `autoindex`, `gzip_static`, `static`. Запросы, в которых URI завершается знаком косой черты, поступают одному из индексных обработчиков. Если модуль gzip не активирован, то запрос обрабатывается модулем static. Как эти модули находят файл или каталог в файловой системе, определяется сочетанием нескольких директив. Директиву `root` лучше всего определять внутри сервера по умолчанию или, по крайней мере, вне директив `location`, чтобы она относилась ко всему серверу:

```
server {
  root /home/customer/html;

  location / {
    index index.html index.htm;
  }

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

В этом примере возвращаемые файлы следует искать в каталоге `/home/customer/html`, который считается корневым. Если клиент указал только имя домена, NGINX попытается найти файл `index.html`. Если такого файла нет, то NGINX будет искать файл `index.htm`. Если пользователь укажет в браузере URI-адрес `/downloads`, то получит список файлов в этом каталоге в формате HTML. Это упрощает доступ к сайтам, на которых размещается интересное пользователям программное обеспечение. NGINX автоматически переписывает URI каталога, добавляя в конец знак косой черты, а затем выполняет переадресацию по протоколу HTTP. NGINX дописывает указанный URI в конец корневого каталога, чтобы получить путь к запрошенному клиентом файлу. Если такого файла не существует, клиенту возвращается сообщение об ошибке **404 Not Found**. Если вы не хотите, чтобы клиент получал такое сообщение, можете попробовать доставить файл из другого места в файловой системе и вернуться к универсальной странице, только если все остальные возможности исчерпаны. Для этой цели можно использовать директиву `try_files`:

```
location / {
  try_files $uri $uri/ backups/$uri /generic-not-found.html;
}
```

Для пущей безопасности NGINX может проверить путь к файлу, который собирается доставить, и, если где-то в нем встречается символическая ссылка, вернуть клиенту сообщение об ошибке:

```
server {
  root /home/customer/html;
  disable_symlinks if_not_owner from=$document_root;
}
```

Здесь NGINX вернет сообщение «Permission Denied», если в части пути после `/home/customer/html` найдена символическая ссылка и файл, на который она указывает, не принадлежит пользователю с таким же идентификатором.

Описанные только что директивы перечислены в следующей таблице.

Директивы HTTP-сервера, относящиеся к путям в файловой системе

Директива	Описание
`disable_symlinks`	Определяет, должна ли NGINX проверять наличие символических ссылок в пути к файлу перед его отправкой клиенту. Распознаются следующие параметры: • `off`: отключает проверку символических ссылок (по умолчанию); • `on`: если какая-нибудь часть пути является символической ссылкой, доступ запрещается; • `if_not_owner`: если какая-нибудь часть пути является символической ссылкой, которая ведет на файл, принадлежащий другому пользователю, то доступ запрещается; • `from=part`: часть пути до part включительно не проверяется на символические ссылки, а остаток проверяется в соответствии с режимом, заданным параметром `on` или `if_not_owner`
`root`	Задает путь к «корню документов». При поиске файлов заданный в запросе URI добавляется в конец этого пути
`try_files`	Проверяет существование файлов, указанных в параметрах. Если ни один файл, кроме последнего, не найден, то последний параметр считается «последней надеждой», поэтому позаботьтесь о том, чтобы такой файл или именованное местоположение существовали, либо задайте возвращаемый код состояния в формате `=<status code>`

Разрешение имен

Если вместо IP-адресов проксируемых серверов в директивах `upstream` или `*_pass` используются доменные имена, то NGINX по умолчанию обращается к механизму разрешения имен, встроенному в операционную систему, за IP-адресом, который необходим для установления соединения с сервером. Это происходит только один раз, при первом обращении к проксируемому серверу, и не происходит вовсе, если в директиве `*_pass` встречается переменная. Однако можно сконфигурировать для NGINX отдельный разреши-

тель. В этом случае можно переопределить время жизни (TTL), возвращаемое DNS-сервером, и использовать в директивах `*_pass` переменные.

```
server {
 resolver 192.168.100.2 valid=300s;
}
```

Директивы разрешения имен

Директива	Описание
`resolver`	Задает имена одного или нескольких серверов, используемых для получения IP-адресов по доменным именам. Необязательный параметр `valid` переопределяет время TTL, заданное в записи о доменном имени

Чтобы заставить NGINX каждый раз получать IP-адрес заново, поместите логическое имя в переменную. Разрешая эту переменную, NGINX неявно производит поиск IP-адреса в DNS. Но чтобы это заработало, необходимо включить директиву `resolver`:

```
server {
  resolver 192.168.100.2;

  location / {
    set $backend upstream.example.com;
    proxy_pass http://$backend;
  }
}
```

Разумеется, используя DNS для поиска проксируемого сервера, мы предполагаем, что разрешитель всегда доступен. Если это не так, произойдет ошибка шлюза. Чтобы сократить время ожидания ответа клиентом, следует присвоить небольшое значение параметру `resolver_timeout`. Затем ошибку шлюза можно перехватить в директиве `error_ page`, предназначенной специально для этой цели.

```
server {
  resolver 192.168.100.2;
  resolver_timeout 3s;
  error_page 504 /gateway-timeout.html;

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

Взаимодействие с клиентами

NGINX может взаимодействовать с клиентами разными способами. Настраиваются как атрибуты самого соединения (IP-адрес, таймауты, свойство `keepalive` и т. д.), так и заголовки для согласования содержимого. В следующей таблице описаны директивы для конфигурирования различных заголовков и кодов ответа, которые служат клиенту указанием либо запросить страницу, либо взять ее из собственного кэша.

Директивы модуля HTTP для взаимодействия с клиентами

Директива	Описание
`default_type`	Определяет подразумеваемый по умолчанию MIME-тип ответа. Используется в случае, когда MIME-тип файла не удается сопоставить ни с одним из определенных в директиве `types`
`error_page`	Определяет URL-адрес страницы, которую нужно вернуть, если код ответа попадает в диапазон ошибок. Необязательный параметр, следующий за знаком =, позволяет изменить код ответа. Если после знака равенства не указан код ответа, то он берется из URI-адреса, при этом соответствующая страница должна возвращаться каким-то проксируемым сервером
`etag`	Отключает автоматическую генерацию заголовка ответа `ETag` для статических ресурсов (по умолчанию `on`)
`if_modified_since`	Управляет порядком сравнения времени модификации ответа со значением в заголовке запроса `If-Modified-Since`. • `off`: заголовок `If-Modified-Since` игнорируется. • `exact`: точное соответствие (по умолчанию). • `before`: время модификации ответа меньше или равно значению в заголовке `If-Modified-Since`
`ignore_invalid_headers`	Разрешает или запрещает игнорировать заголовки с недопустимыми именами (по умолчанию `on`). Допустимыми считаются имена, содержащие буквы в кодировке ASCII, цифры, знак минус и, возможно, знак подчеркивания (определяется директивой `underscores_in_headers`)
`merge_slashes`	Разрешает или запрещает удаление идущих подряд знаков косой черты. Подразумеваемое по умолчанию значение `on` означает, что NGINX будет заменять несколько соседних знаков / одним.
`recursive_error_pages`	Разрешает производить несколько переадресаций с помощью директивы `error_page` (по умолчанию `off`)

Директива	Описание
types	Определяет соответствие между MIME-типами и расширениями имен файлов. В дистрибутив NGINX входит файл conf/mime. types, содержащий большинство соответствий. Как правило, достаточно просто включить этот файл директивой include
underscores_in_headers	Разрешает или запрещает использование символа подчеркивания в заголовках запросов от клиентов. Если оставлено подразумеваемое по умолчанию значение off, то анализ таких заголовков производится согласно режиму, заданному в директиве ignore_invalid_headers

Директива error_page - одна из самых гибких в NGINX. С ее помощью можно вернуть любую страницу в случае возникновения ошибки. Эта страница может находиться на локальной машине, а может динамически генерироваться сервером приложений или даже размещаться совсем на другом сайте.

```
http {
  # обобщенная страница, возвращаемая при любой ошибке самого сервера
  error_page 500 501 502 503 504 share/examples/nginx/50x.html.

  server {
    server_name www.example.com;
    root /home/customer/html;

    # если файл не найден, то возвращается содержимое файла
    # /home/customer/html/404.html
    error_page 404 /404.html;

    location / {
      # ошибки сервера для этого виртуального хоста переадресуются
      # специальному обработчику в приложении
      error_page 500 501 502 503 504 = @error_handler;
    }

    location /microsite {
      # если не существует файл в области /microsite,
      # то клиенту демонстрируется страница с другого сервера
      error_page 404 http://microsite.example.com/404.html;
    }

    # именованное местоположение, содержащее специальный
    # обработчик ошибок
    location @error_handler {
    # мы задаем здесь тип по умолчанию, чтобы браузер
```

```
        # корректно отобразил страницу ошибки
        default_type text/html;
        proxy_pass http://127.0.0.1:8080;
    }
  }
}
```

Установка предельных значений для предотвращения недобросовестного использования

Мы создаем сайты, чтобы пользователи на них заходили. Мы хотим, чтобы наши сайты всегда были открыты для добросовестного доступа. Но это означает, что необходимо принимать меры для ограничения доступа недобросовестным пользователям. Под «недобросовестностью» можно понимать разные вещи, например, отправку более одного запроса в секунду одним пользователем или чрезмерно большое количество соединений, открытых с одного и того же IP-адреса. Недобросовестность может также принимать форму **DDOS-атаки** (распределенная атака типа «отказ от обслуживания»), когда роботы с различных компьютеров, разбросанных по всему миру, одновременно пытаются обращаться к сайту с максимальной частотой. В этом разделе мы рассмотрим, как противостоять таким недобросовестным действиям и обеспечить доступность сайта.

Сначала приведем конфигурационные директивы, предназначенные для этой цели.

Директивы модуля HTTP для задания предельных значений

Директива	Описание
`limit_conn`	Определяет зону разделяемой памяти (настраиваемую с помощью директивы `limit_conn_zone`) и максимальное количество соединений с одинаковым значением ключа
`limit_conn_log_ level`	Если NGINX ограничивает соединения согласно директиве `limit_conn`, то эта директива определяет уровень протоколирования для сообщения о достижении пороговой величины
`limit_conn_zone`	В первом параметре задается ключ, к которому относятся ограничения, указанные в директиве `limit_conn`. Второй параметр задает имя зоны разделяемой памяти, в которой хранится не более указанного числа соединений для каждого ключа, а также размер этой зоны (`name:size`)

Директива	Описание
limit_rate	Ограничивает скорость (в байтах/с) отдачи содержимого клиентам. Ограничение действует на уровне соединения, то есть один клиент может повысить свою пропускную способность, открыв несколько соединений
limit_rate_after	Начинает применять ограничение limit_rate после того, как передано указанное количество байтов
limit_req	Задает ограничение при резком увеличении (всплеске) количества запросов для указанного ключа в зоне разделяемой памяти (заданной в директиве limit_req_zone). Всплеск описывается вторым параметром. Если до возникновения всплеска задерживать запросы не нужно, следует включить третий параметр nodelay
limit_req_log_level	Если NGINX ограничивает количество запросов согласно директиве limit_req, то эта директива определяет уровень протоколирования для сообщения о достижении пороговой величины. Сообщение о применении задержки имеет уровень, на единицу меньший указанного в этой директиве
limit_req_zone	В первом параметре задается ключ, к которому относятся ограничения, указанные в директиве limit_req. Второй параметр задает имя зоны разделяемой памяти, в которой хранится не более указанного числа запросов для каждого ключа, а также размер этой зоны (name: size). Третий параметр определяет количество запросов в секунду (r/s) или в минуту (r/m), при превышении которого начинает применяться ограничение
max_ranges	Задает максимальное количество диапазонов, допустимых в запросе с указанием диапазонов байтов. Если задано значение 0, то поддержка диапазонов байтов отключается

В примере ниже мы разрешаем не более 10 соединений с одного IP-адреса. Для нормального доступа к сайту этого должно быть достаточно, потому что современные браузеры открывают только два или три соединения с одним сервером. Однако не забывайте, что для всех пользователей, находящихся за прокси-сервером, сервер видит один и тот же IP-адрес. Поэтому следите за появлением в журналах сообщений об ошибке 503 (Service Unavailable), означающей применение этого ограничения:

```
http {
  limit_conn_zone $binary_remote_addr zone=connections:10m;
  limit_conn_log_level notice;

  server {
```

```
    limit_conn connections 10;
   }
 }
```

Ограничение доступа на основе частоты запросов выглядит почти так же, но работает несколько иначе. Когда мы ограничиваем количество страниц, запрашиваемых пользователем в единицу времени, NGINX вставляет задержку после запроса первой страницы и так тех пор, пока число избыточных запросов не превысит размер всплеска. Возможно, вы этого не хотите, поэтому NGINX предоставляет возможность подавить задержку с помощью параметра nodelay:

```
http {
  limit_req_zone $binary_remote_addr zone=requests:10m rate=1r/s;
  limit_req_log_level warn;

  server {
    limit_req zone=requests burst=10 nodelay;
  }
}
```

Использование переменной $binary_remote_addr
В примере выше мы воспользовались переменной `$binary_remote_addr`, чтобы узнать, сколько места в памяти отводится для хранения одного IP-адреса. На 32-разрядных платформах это 32 байта, на 64-разрядных - 64 байта. Поэтому сконфигурированная зона размером `10m` на 32-разрядных платформах способна хранить 320 000 состояний, а на 64-разрядных - 160 000.

Можно также ограничить потребление полосы пропускания одним клиентом, чтобы несколько недобросовестных пользователей не заняли всю доступную полосу. Но имейте в виду одну тонкость: директива `limit_rate` работает на уровне соединений. Если одному клиенту разрешено открывать несколько соединений, то он легко сможет обойти это ограничение:

```
location /downloads {
  limit_rate 500k;
}
```

Можно вместо этого разрешить частые запросы при загрузке небольших файлов, но ввести ограничение для файлов большего размера:

```
location /downloads {
  limit_rate_after 1m;
  limit_rate 500k.
}
```

Комбинируя различные ограничения на частоту, можно создать конфигурацию с очень гибкими ограничениями на действия клиентов:

```
http {
  limit_conn_zone $binary_remote_addr zone=ips:10m;
  limit_conn_zone $server_name zone=servers:10m;
  limit_req_zone $binary_remote_addr zone=requests:10m rate=1r/s;
  limit_conn_log_level notice;
  limit_req_log_level warn;
  reset_timedout_connection on;

  server {
    # эти ограничения применяются ко всему виртуальному серверу
    limit_conn ips 10;

    # не более 1000 одновременных соединений с одним server_name
    limit_conn servers 1000;

    location /search {
      # здесь мы ограничиваем частоту запросов только для URL /search
      limit_req zone=requests burst=3 nodelay.
    }

    location /downloads {
      # используем limit_conn, чтобы ограничить потребление полосы
      # пропускания одним клиентом, не позволяя обойти ограничение
      limit_conn connections 1;
      limit_rate_after 1m;
      limit_rate 500k;
    }
  }
 }
```

Ограничение доступа

В предыдущем разделе мы рассмотрели различные способы поставить преграды на пути недобросовестного посетителя сайта, работающего под управлением NGINX. Теперь посмотрим, как ограничить доступ ко всему сайту или его частям. Есть два способа ограничения доступа: с некоторых IP-адресов или некоторым пользователям.

Комбинируя эти способы, мы можем разрешить пользователям доступ к сайту либо с определенных IP-адресов, либо при условии ввода корректного имена и пароля.

В этом нам помогут следующие директивы.

Директивы модуля HTTP для управления доступом

Директива	Описание
`allow`	Разрешает доступ с указанного IP-адреса, из указанной сети или отовсюду (`all`)
`auth_basic`	Разрешает аутентификацию по схеме HTTP Basic Authentication. Параметр задает имя области (realm). Специальное значение `off` означает, что значение `auth basic`, заданное на родительском уровне, отменяется
`auth_basic_user_file`	Определяет, где находится файл, содержащий тройки `username:password:comment`, который используется для аутентификации клиентов. Поле password должно быть зашифровано алгоритмом crypt. Поле `comment` необязательно
`deny`	Запрещает доступ с указанного IP-адреса, из указанной сети или отовсюду (`all`)
`satisfy`	Разрешает доступ, если все (`all`) или хотя бы одна (`any`) из предыдущих директив разрешает доступ. Подразумеваемое по умолчанию значение `all` говорит, что пользователь должен находиться в определенной сети и ввести правильный пароль

Чтобы разрешить доступ только клиентам, которые заходят с определенных IP-адресов, используйте директивы `allow` и `deny` следующим образом:

```
location /stats {
  allow 127.0.0.1;
  deny all;
}
```

Здесь мы разрешаем доступ к URI `/stats` только с локального компьютера `localhost`.

Чтобы разрешить доступ только аутентифицированным пользователям, воспользуйтесь директивами `auth_basic` и `auth_basic_user_file`:

```
server {
  server_name restricted.example.com;
  auth_basic "restricted";
  auth_basic_user_tile conf/htpasswd;
}
```

Пользователь, желающий зайти па сайт `restricted.example.com`, должен будет ввести имя и пароль, присутствующие в файле `htpasswd` в подкаталоге `conf` корневого каталога NGINX. Записи в `htpasswd` можно сгенерировать любым инструментом, поддерживающим стандартную для UNIX функцию `crypt()`. Так, приведенный ниже скрипт на Ruby генерирует файл нужного формата.

```
#!/usr/bin/env ruby
# разбираем флаги в командной строке
require 'optparse'
OptionParser.new do |o|
  o.on('-f FILE') { |file| $file = file }
  o.on('-u', "--username USER") { |u| $user = u }
  o.on('-p', "--password PASS") { |p| $pass = p }
  o.on('-c', "--comment COMM (optional)") { |c| $comm = c }
  o.on('-h') { puts o; exit }
  o.parse!
  if $user.nil? or $pass.nil?

  puts o; exit
 end
end
# инициализируем массив ASCII-символов, используемый в качестве затравки
ascii = ('a'..'z').to_a + ('A'..'Z').to_a + ('0'..'9').to_a + [ ".", "/" ]
$lines = []
  begin
  # читаем аутентификационные данные из файла
      File.open($file) do |f|
        f.lines.each { |l| $lines << l }
      end
      rescue Errno::ENOENT
  # если файл не найден (при первом использовании), инициализируем массив
       $lines = ["#{$user}:#{$pass}\n"]
      end
 # удаляем редактируемую запись о пользователе из текущего
 списка$lines.map! do |line|
 # если есть комментарий, вставляем его
   unless line =~ /#{$user}:/
     line
   end
 end
# шифруем пароль с помощью crypt()
pass = $pass.crypt(ascii[rand(64)] + ascii[rand(64)])
```

```
if $comm
  $lines << "#{$user}:#{pass}:#{$comm}\n"
else
  $lines << "#{$user}:#{pass}\n"
end
# записываем новый файл, при необходимости создав его
File.open($file, File::RDWR|File::CREAT) do |f|
  $lines.each { |l| f << l}
end
```

Сохраните этот код в файле `http_auth_basic.rb` и выполните его, указав имя файла (`-f`), имя пользователя (`-u`) и пароль (`-p`). Будет сгенерирована запись в формате, который понимает директива NGINX `auth_basic_user_file`:

```
$ ./http_auth_basic.rb -f htpasswd -u testuser -p 123456
```

Для случая, когда имя и пароль нужно вводить, только если вход осуществляется по с одного из предопределенных IP-адресов, в NGINX предусмотрена директива `satisfy`. В примере ниже задан параметр `any`, подходящий для сценария или-или:

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

  location / {
    auth_basic "intranet: please login";
    auth_basic_user_file conf/htpasswd-intranet;
    allow 192.168.40.0/24;
    allow 192.168.50.0/24;
    deny all;
    satisfy any;
}
```

Если, напротив, требуется, чтобы пользователь мог заходить только с определенных IP-адресов и при этом аутентифицировался, то следует воспользоваться параметром `all`, который и так подразумевается по умолчанию. Поэтому директиву `satisfy` можно опустить, оставив только директивы `allow`, `deny`, `auth_basic` и `auth_basic_user_file`:

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

  location / {
    auth_basic "staging server";
```

```
    auth_basic_user_file conf/htpasswd-stage;
    allow 192.168.40.0/24;
    allow 192.168.50.0/24;
    deny all;
}
```

Потоковая передача мультимедийных файлов

NGINX умеет обслуживать запросы на некоторые типы видеофайлов. Модули `flv` и `mp4`, включенные в базовый дистрибутив, выполняют так называемую **псевдопотоковую передачу**. Это означает, что NGINX начнет передачу с того места видеофайла, которое указано в параметре запроса `start`.

Для поддержки псевдопотоковой передачи на этапе конфигурирования сборки необходимо указать флаги `--with-http_flv_module` (для Flash Video - FLV) и (или) `--with-http_mp4_module` (для H.264/AAC). Тогда в конфигурационном файле можно будет использовать следующие директивы.

Директивы модуля HTTP для управления потоковой передачей

Директива	Описание
`flv`	Активирует модуль `flv` для данного местоположения
`mp4`	Активирует модуль `mp4` для данного местоположения
`mp4_buffer_size`	Задает начальный размер буфера для доставки MP4-файлов
`mp4_max_buffer_size`	Задает максимальный размер буфера для обработки метаданных MP4

Активация псевдопотоковой передачи FLV для местоположения сводится к добавлению директивы `flv`:

```
location /videos {
  flv;
  }
```

У псевдопотоковой передачи MP4 параметров больше, так как в формате H.264 имеются метаданные, которые необходимо разбирать. Поиск в файле доступен, если проигрыватель обнаружил атом "moov atom". Поэтому для повышения производительности метаданные должны находиться в начале файла. Если в журнале появляется сообщение

```
mp4 moov atom is too large
```

то увеличьте значение параметра `mp4_max_buffer_size`. Это можно сделать следующим образом:

```
location /videos {
  mp4;
  mp4_buffer_size 1m;
  mp4_max_buffet size 20m;
}
```

Предопределенные переменные

В конфигурационном файле NGINX могут встречаться переменные, вместо которых на этапе выполнения подставляются их значения. Помимо пользовательских переменных, создаваемых директивами `set` и `map`, имеются также предопределенные переменные, устанавливаемые самой NGINX. Их вычисление оптимизировано, а значения кэшируются на время выполнения запроса. Любую из таких переменных можно использовать в предложении `if` или в директиве `proxy_pass`. Некоторые полезны при определении нестандартного формата журнала. Попытка создать свою переменную с таким же именем, как у предопределенной, приведет к ошибке:

```
<timestamp> [emerg] <master pid>#0: the duplicate "<variable_name>"
variable in <path-to-configuration-file>:<line-number>
```

Кроме того, они не предназначены для макрорасширения конфигурационного файла, а используются главным образом во время выполнения.

В модуле `http` определены следующие переменные.

Переменные модуля HTTP

Имя переменной	Значение
`$arg_name`	Аргумент с именем `name`, присутствующий в параметрах запроса
`$args`	Все параметры запроса
`$binary_remote_addr`	IP-адрес клиента в двоичном виде (длина всегда равна 4 байтам)
`$content_length`	Значение заголовка запроса `Content-Length`
`$content_type`	Значение заголовка запроса `Content-Type`

Имя переменной	Значение
$cookie_name	Значение кука с именем name
$document_root	Значение директивы root или alias для текущего запроса
$document_uri	Псевдоним $uri
$host	Значение заголовка запроса Host, если таковой присутствует. В противном случае это значение, заданное в директиве server_name, отвечающей запросу
$hostname	Имя хоста, на котором работает NGINX
$http_name	Значение заголовка запроса с именем name. Если имя заголовка содержит дефисы, они преобразуются в знаки подчеркивания. Прописные буквы преобразуются в строчные
$https	Для соединений, шифруемых по протоколу SSL, принимает значение on. В противном случае пустая строка
$is_args	Если в запросе есть аргументы, принимает значение ?. В противном случае пустая строка
$limit_rate	Значение, заданное в директиве limit_rate. Если значение не установлено, то с помощью этой переменной можно регулировать скорость отдачи содержимого клиентам
$nginx_version	Номер версии работающей программы nginx
$pid	Идентификатор рабочего процесса
$query_string	Псевдоним $args
$realpath_root	Значение директивы root или alias для текущего запроса после разрешения всех символических ссылок
$remote_addr	IP-адрес клиента
$remote_port	Номер порта клиента
$remote_user	При использовании простой схемы аутентификации по HTTP эта переменная содержит имя пользователя
$request	Полный запрос, полученный от клиента, включая метод HTTP, URI-адрес, версию протокола HTTP, заголовки и тело
$request_body	Тело запроса для использования в местоположениях, обрабатываемых директивой *_pass
$request_body_file	Путь к временному файлу, в котором сохранено тело запроса. Для гарантированного создания этого файла директива client_body_in_file_only должна принимать значение on
$request_completion	Если запрос получен полностью, принимает значение OK, иначе пустая строка
$request_filename	Путь к файлу для текущего запроса, формируемый на основе значения директивы root или alias и URI-адреса
$request_method	HTTP-метод текущего запроса
$request_uri	Полный URI, указанный в запросе, включая аргументы
$scheme	Схема текущего запроса: HTTP или HTTPS
$sent_http_name	Значение заголовка ответа с именем name. Если имя заголовка содержит дефисы, они преобразуются в знаки подчеркивания. Прописные буквы преобразуются в строчные

Имя переменной	Значение
$server_addr	Адрес сервера, принявшего запрос
$server_name	Значение директивы server_name виртуального сервера, принявшего запрос
$server_port	Номер порта сервера, принявшего запрос
$server_protocol	Версия протокола HTTP для текущего запроса
$status	Код состояния ответа
$tcpinfo_rtt $tcpinfo_rttvar $tcpinfo_snd_cwnd $tcpinfo_rcv_space	Если система поддерживает опцию сокета TCP_INFO, то в эти переменные заносится соответствующая информация
$uri	Нормализованный URI текущего запроса

Использование NGINX совместно с PHP-FPM

Долгое время Apache считался единственным веб-сервером для сайтов, написанных на PHP, поскольку его модуль mod_php позволяет без труда интегрировать PHP непосредственно с веб-сервером. Но после включения **PHP-FPM** в ядро PHP появилась альтернатива. PHP-FPM - это технология исполнения PHP-скриптов под управлением FastCGI-сервера. Главный процесс PHP-FPM берет на себя заботу о запуске рабочих процессов с учетом нагрузки на сайт; при необходимости дочерние процессы перезапускаются. Главный процесс взаимодействует с другими службами по протоколу FastCGI. Подробнее о PHP-FPM можно прочитать на странице http://php.net/manual/en/install.fpm.php.

В NGINX имеется модуль fastcgi, который может взаимодействовать не только с PHP-FPM, но и с любым сервером, совместимым с протоколом FastCGI. Он включен по умолчанию, поэтому для использования NGINX совместно с FastCGI-серверами ничего специально делать не нужно.

Директивы модуля FastCGI

Директива	Описание
fastcgi_buffer_size	Размер буфера для первой части ответа от FastCGI-сервера, в которой находятся заголовки
fastcgi_buffers	Количество и размер буферов для получения ответа от FastCGI-сервера в расчете на одно соединение

Директива	Описание
fastcgi_busy_buffers_size	Суммарный размер буферов, которые могут быть заняты для отправки ответа клиенту, пока ответ от FastCGI-сервера ещё не прочитан целиком. Обычно устанавливается вдвое большим, чем размер, указанный в директиве fastcgi _buffers
fastcgi_cache	Определяет зону разделяемой памяти, используемую для кэширования
fastcgi_cache_bypass	Одна или несколько строковых переменных. Если хотя бы одна из них непуста и не содержит нуль, то ответ будет запрошен у FastCGI-сервера, а не взят из кэша
fastcgi_cache_key	Строка, которая используется как ключ для поиска значения в кэше
fastcgi_cache_lock	Если включено, то предотвращается отправка нескольких запросов FastCGI-серверу в случае отсутствия в кэше
fastegi_cache_lock_timeout	Сколько времени запрос может ждать появления записи в кэше или освобождения блокировки fastcgi_cache_lock
fastcgi_cache_min_uses	Сколько запросов с данным ключом должно поступить, прежде чем ответ будет помещен в кэш
fastcgi_cache_path	Каталог, в котором хранятся кэшированные ответы, и зона разделяемой памяти (keys_zone=name:size) для хранения активных ключей и метаданных ответа. Необязательные параметры: • levels: список разделенных двоеточиями длин имен подкаталогов на каждом уровне (1 или 2); допускается не более трех уровней вложенности; • inactive: максимальное время нахождения неактивного запроса в кэше, по прошествии которого он вытесняется; • max_size: максимальный размер кэша; по его достижении процесс-диспетчер удаляет элементы, к которым дольше всего не было обращений; • loader_files: максимальное количество кэшированных файлов, метаданные которых загружаются в кэш на одной итерации процесса-загрузчика; • loader_sleep: число миллисекунд между итерациями процесса-загрузчика кэша; • loader_threshold: максимальное время, отведенное на одну итерацию процесса-загрузчика кэша
fastcgi_cache_use_stale	В каких случаях допустимо использовать устаревшие кэшированные данные, если при доступе к FastCGI-серверу произошла ошибка. Параметр updating разрешает использовать кэшированный ответ, если как раз в данный момент загружаются более свежие данные

Директива	Описание
`fastcgi_cache_valid`	Сколько времени считать действительным кэшированный ответ с кодом 200, 301 или 302. Если перед параметром `time` указан необязательный код ответа, то заданное время относится только к ответу с таким кодом. Специальный параметр `any` означает, что в течение заданного времени следует кэшировать ответ с любым кодом
`fastcgi_connect_timeout`	Максимальное время, в течение которого NGINX ожидает установления соединения при отправке запроса FastCGI-серверу
`fastcgi_hide_header`	Список заголовков, которые не следует передавать клиенту
`fastcgi_ignore_client_abort`	Если значение равно `on`, то NGINX не станет разрывать соединение с FastCGI-сервером в случае, когда клиент разрывает свое соединение
`fastcgi_ignore_beaders`	Какие заголовки можно игнорировать при обработке ответа от FastCGI-сервера
`fastcgi_index`	Задает имя файла, дописываемое в конец строки `$fastcgi_script_name` после знака косой черты
`fastcgi_intercept_errors`	Если значение равно `on`, то NGINX будет отображать страницу, заданную директивой `error_page`, вместо ответа, полученного от FastCGI-сервера
`fastcgi_keep_conn`	Разрешает соединения типа `keepalive` с FastCGI-сервером, инструктируя его не закрывать соединение немедленно
`fastcgi_max_temp_file_size`	Максимальный размер временного файла, в который записывается часть ответа в случае, когда он не умещается целиком в буферах памяти
`fastcgi_next_upstream`	Определяет условия, при которых для ответа будет выбран следующий FastCGI-сервер. Это не происходит, если клиент уже что-то отправил. Условия задаются с помощью следующих параметров: • `error`: произошла ошибка при взаимодействии с FastCGI-сервером; • `timeout`: произошел таймаут при взаимодействии с FastCGI-сервером; • `invalid_header`: FastCGI-сервер вернул пустой или недопустимый ответ; • `http_500`: FastCGI-сервер вернул ответ с кодом ошибки 500; • `http_503`: FastCGI-сервер вернул ответ с кодом ошибки 503; • `http_404`: FastCGI-сервер вернул ответ с кодом ошибки 404; • `off`: запретить передачу запроса следующему FastCGI-серверу в случае ошибки

Директива	Описание
fastcgi_no_cache	Одна или несколько строковых переменных. Если хотя бы одна из них непуста и не содержит нуль, то NGINX не будет помещать в кэш ответ, полученный от FastCGI-сервера
fastcgi_param	Задает имя и значение параметра, передаваемого FastCGI-серверу. Если следует передавать только параметры с непустыми значениями, то необходимо задать дополнительный параметр if_not_empty
fastcgi_pass	Определяет FastCGI-сервер, которому передается запрос, в виде пары address:port или unix:path для сокета в домене UNIX
fastcgi_pass_header	Отменяет сокрытие заголовков, определенных в директиве fastcgi_hide_header, разрешая передавать их клиенту
fastcgi_read_timeout	Сколько времени может пройти между двумя последовательными операциями чтения данных от FastCGI-сервера, прежде чем соединение будет закрыто
fastcgi_send_timeout	Сколько времени может пройти между двумя последовательными операциями записи данных на FastCGI-сервер, прежде чем соединение будет закрыто
fastcgi_split_path_info	Задает регулярное выражение с двумя запоминаемыми подвыражениями. Первая запомненная строка становится значением переменной $fastcgi_script_name, вторая - значением переменной $fastcgi_path_info. Необходимо только для приложений, в которых используется переменная PATH_INFO
fastcgi_store	Разрешает сохранение полученных от FastCGI-сервера ответов в файлах на диске. Если параметр равен on, то в качестве пути к каталогу с сохраняемыми файлами используется значение, заданное в директиве alias или root. Можно вместо этого задать строку, определяющую другой каталог для хранения файлов
fastcgi_store_access	Какие права доступа следует задать для новых файлов, создаваемых в соответствии с директивой fastcgi_store
fastcgi_temp_file_write_size	Ограничивает размер данных в одной операции записи во временный файл, чтобы NGINX не слишком долго блокировала исполнение программы при обработке одного запроса
fastcgi_temp_path	Каталог для хранения временных файлов, получаемых от FastCGI-сервера. Может быть многоуровневым

Пример конфигурации для Drupal

Drupal (http://drupal.org) - это популярная платформа с открытым исходным кодом для управления содержимым. Она установлена на многих известных сайтах. Как и большинство веб-каркасов на основе PHP, Drupal обычно работает под Apache с модулем `mod_php`. Мы покажем, как настроить NGINX для запуска Drupal.

На сайте https://github.com/perusio/drupal-with-nginx имеется очень подробная инструкция по настройке Drupal для работы с NGINX. Она куда детальнее того, что мы можем позволить себе в этой книге, но некоторые вещи мы все же отметим и остановимся на различиях между Drupal 6 и Drupal 7:

```
## Определяем переменную $no_slash_uri variable для Drupal 6.
map $uri $no_slash_uri {
  ~^/(?<no_slash>.*)$ $no_slash;
}

server {
  server_name www.example.com;
  root /home/customer/html;
  index index.php;

  # Не закрывать соединения с FastCGI-сервером (используется в
  # сочетании с директивой "Keepalive" в секции upstream)
  fastcgi_keep_conn on;

  # Местоположение по умолчанию.
  location / (
    ## (Drupal 6) Использовать index.html. если нет index.php.
    location = / {
      error_page 404 =200 /index.html;
    }
    # Обслуживание обычных частных файлов (тех, что обрабатываются Drupal).
    location ^~ /system/files/ {

      include fastcgi_private_files.conf;
      fastcgi_pass 127.0.0.1:9000;
      # Добавить следующую строку, чтобы в журнал ошибок не попадало
      # сообщение об ошибке 404 при доступе к каталогу system/files.
      # Ошибка 404 в данном случае - ожидаемое поведение.
  log_not_found off;
  }
  # Попытка прямого доступа к частным файлам возвращает ошибку 404.
    location ^~ /sites/default/files/private/ {
```

```
    internal;
  }
  ## (Drupal 6) При доступе к изображению, сгенерированному imagecache,
  ## пытаемся вернуть его, если оно имеется, иначе переправляем запрос
  ## Drupal для генерации или (перегенерации).
    location ~* /imagecache/ {
      access_log off;
      expires 30d;
      try_files $uri /index.php?q=$no_slash_uri&$args;
    }
    # Обработка изображений в Drupal 7, когда imagecache - часть ядра
    location ~* /files/styles/ {
      access_log off;
      expires 30d;
      try_files $uri @drupal;
    }
```

Следующие далее настройки модуля Advanced Aggregation отличаются только значением `location`. Вот как выглядит конфигурация этого модуля для CSS:

```
# Поддержка CSS модулем Advanced Aggregation.
location ^~ /sites/default/files/advagg_css/ {
location ~* /sites/default/files/advagg_css/css_[[:alnum:]]+\.css$ {
```

А так для JavaScript:

```
# Поддержка JS модулем Advanced Aggregation.
location ^~ /sites/default/files/advagg_js/ {
location ~* /sites/default/files/advagg_js/js_ [[:alnum:]]+\.js$ {
```

Далее приведены строки, общие для обеих секций:

```
      access_log off;
      add_header Pragma '';
      add_header Cache-Control 'public, max-age=946080000';
      add_header Accept-Ranges '';

      # Это для Drupal 7
      try_files $uri @drupal;

      ## Это для Drupal 6 (оставить что-то одно)

    try_files $uri /index.php?q=$no_slash_uri&$args;
  }
}
  # Все статические файлы обслуживаются непосредственно.
```

```
  location ~* ^.+\.(?:css|cur|js|jpe?g|gif|htc|ico|png|html|xml)$ {
    access_log off;
    expires 30d;

    # Посылать все сразу.
    tcp_nodelay off;

    # Настроить файловый кэш операционной системы
    open_file_cache max=3000 inactive=120s;
    open_file_cache_valid 45s;
    open_file_cache_min_uses 2;
    open_file_cache_errors off;
  }
  # Обработка файлов PDF и powerpoint.
  location ~* ^.+\.(?:pdf|pptx?)$ {
    expires 30d;
    # Посылать все сразу.
    tcp_nodelay off;
  }
```

На примере звуковых файлов демонстрируется использование асинхронного ввода-вывода. Местоположение MP3 выглядит так:

```
# MP3-файлы обслуживаются с использованием AIO, если ОС поддерживает.
  location ^~ /sites/default/files/audio/mp3 {
    location ~* ^/sites/default/files/audio/mp3/.*\.mp3$ {
```

А местоположение для файлов Ogg/Vorbis - так:

```
  # Ogg/Vorbis-файлы обслуживаются с использованием AIO, если
  # ОС поддерживает.
  location ^~ /sites/default/files/audio/ogg {
    location ~* ^/sites/default/files/audio/ogg/.*\.ogg$ {
```

Следующие строки являются общими для обоих местоположений:

```
      directio 4k; # for XFS
      tcp_nopush off;
      aio on;
      output_buffers 1 2M;
    }
  }
  # Псевдопотоковая передача FLV-файлов

    location ^~ /sites/default/files/video/flv {
      location ~* ^/sites/default/files/video/flv/.*\.flv$ {
    flv;
    }
  }
```

Следующие две секции также похожи. Псевдопотоковая передача для H264-файлов настраивается так:

```
# Псевдопотоковая передача H264-файлов.
 location ^~ /sites/default/files/video/mp4 {
 location ~* ^/sites/default/files/video/mp4/.*\.(?:mp4|mov)$ {
```

А псевдопотоковая передача для AAC-файлов - так:

```
 # Псевдопотоковая передача AAC-файлов.
   location ^~ /sites/default/files/video/m4a {
   location ~* ^/sites/default/files/video/m4a/.*\.m4a$ {
```

Следующие строки являются общими для обеих секций:

```
    mp4;
    mp4_buffer_size 1M;
    mp4_max_buffer_size 5M;
   }
 }
 # Модуль Advanced Help делает доступным файлы README,
 # предоставленные всеми модулями.
   location ^~ /help/ {
    location ~* ^/help/[^/]*/README\.txt$ {
     include fastcgi_private_files.conf;
     fastcgi_pass 127.0.0.1:9000;
    }
   }
 # Копируем директиву Apache <FilesMatch> для стандартного для
 # Drupal файла .htaccess.
 # Запрещаем доступ ко всем файлам, содержащим код.
 # Возвращаем ошибку 404, чтобы не раскрывать информацию.
 # Также скрываем текстовые файлы.
location ~* ^(?:.+\.(?:htaccess|make|txt|engine|inc|info|inst
all|module|profile|po|sh|.*sql|test|theme|tpl(?:\.
php)?|xtmpl)|code-style\.pl|/Entries.*|/Repository|/Root|/Tag|
/Template)$ {
return 404;
}
 # Сначала пробуем URI и перебрасываем на /index.php?q=$uri&$args,
 # если не найдено.
 try_files $uri @drupal;
 ## (Drupal 6) Сначала пробуем URI и перебрасываем на
 # /index.php?q=$no_slash_uri&$args, если не найдено
 # (оставить что-то одно).
 try_files $uri /index.php?q=$no_slash_uri&$args;
```

```
} # конец местоположения по умолчанию

# Ограничиваем доступ к обязательным файлам PHP. Сужаем
# область воздействия эксплойтов. Обработка PHP-кода и
# цикла событий Drupal.
location @drupal {
  # Включаем конфигурацию FastCGI.
  include fastcgi_drupal.conf;
  fastcgi_pass 127.0.0.1:9000;
}

location @drupal-no-args {
  include fastcgi_private_files.conf;
  fastcgi_pass 127.0.0.1:9000.
}

## (Drupal 6)
## Ограничиваем доступ к обязательным файлам PHP. Сужаем
## область воздействия эксплойтов. Обработка PHP-кода и
## цикла событий Drupal.
## (оставить что-то одно)
location = /index.php {


  # Помечен internal в качестве профилактической меры защиты.
  # Прямой доступ к файлу index.php запрещен; доступ возможен
  # только со стороны NGINX из других местоположений или в
  # результате внутренней переадресации.
  internal;
  fastcgi_pass 127.0.0.1:9000;
}
```

Все последующие местоположения возвращают 404, чтобы запретить доступ:

```
#Запретить доступ к каталогу .git: возвращаем 404, чтобы не
#раскрывать информацию.
location ^~ /.git { return 404; }

#Запретить доступ к каталогу patches.
location ^~ /patches { return 404; }

#Запретить доступ к каталогу backup.
location ^~ /backup { return 404; }

# Запретить протоколирование обращений к файлу robots.txt в
# журналах доступа.
location = /robots.txt {
  access_log off;
}
```

```
# Поддержка RSS-каналов.
location = /rss.xml {
  try_files $uri @drupal-no-args;
  ## (Drupal 6: оставить что-то одно)
try_files $uri /index.php?q=$uri;
}
# Поддержка карты сайта в формате XML.
  location = /sitemap.xml {
  try_files $uri @drupal-no-args;
## (Drupal 6: оставить что-то одно)
try_files $uri /index.php?q=$uri;
}
  # Поддержка favicon. Вернуть прозрачный GIF размером 1x1.
  #  если  не  существует.
location = /favicon.ico {
   expires 30d;
   try_files /favicon.ico @empty;
   }
# Возвращает хранящийся в памяти прозрачный GIF размером 1x1
location @empty {
   expires 30d;
   empty_gif;
}
# Все прочие попытки обратиться к PHP-файлам возвращают 404.
location ~* ^.+\.php$ {
   return 404;
   }
} # конец серверного контекста
```

Включаемые директивой `include` файлы здесь для краткости не приводятся. Их можно найти в репозитории perusio на сайте GitHub, упомянутом в начале этого раздела.

Интеграция NGINX и uWSGI

Python **WSGI (Web Server Gateway Interface)** - спецификация интерфейса, формализованная в документе PEP-3333 (http://www.python.org/dev/peps/pep-3333/). Ее цель - описать «стандартный интерфейс между веб-серверами и веб-приложениями или каркасами на языке Python с тем, чтобы повысить переносимость приложения с одного веб-сервера на другой». Вследствие популярности в сообществе пользователей Python разработаны реализации спецификации WSGI на другие языки. Сервер uWSGI, хотя и не был написан спе-

циально для Python, позволяет запускать приложения, отвечающие этой спецификации. Протокол взаимодействия с сервером uWSGI называется uwsgi. Дополнительные сведения о сервере uWSGl, в том числе инструкции по установке, примеры конфигурационных файлов и список поддерживаемых языков можно найти на страницах http://projects.unbit.it/uwsgi/ и https://github.com/unbit/uwsgi-docs.

Модуль NGINX `uwsgi` можно настроить для взаимодействия с этим сервером с помощью директив, аналогичных рассмотренным выше директивам `fastcgi_*`. Большая их часть обладает той же семантикой, что в случае FastCGI, с тем очевидным различием, что их имена начинаются префиксом `uwsgi_` вместо `fastcgi_`. Однако есть и несколько исключений - директивы `uwsgi_modifier1`, `uwsgi_modifier2` и `uwsgi_string`. Первые две устанавливают соответственно первый или второй модификатор в заголовке пакета uwsgi. А директива `uwsgi_string` позволяет NGINX передать произвольную строку серверу uWSGI или любому другому серверу, совместимому с протокол uwsgi и поддерживающему модификатор `eval`. Модификаторы - это специфическая особенность протокола uwsgi. Таблица допустимых модификаторов с описанием их назначения приведена на странице `http://uwsgi-docs. rcadthedocs.org/en/latest/Protocol. html`.

Пример конфигурации для Django

Django (https://www.djangoproject.com/) - это написанный на Python каркас для быстрой разработки высокопроизводительных веб-приложений. Он получил широкое распространение, и теперь для него существуют веб-приложения самых разных видов.

Ниже приведен пример конфигурационного файла, демонстрирующего подключение NGINX к нескольким приложениям для Django, работающим под управлением сервера uWSGI в режиме Emperor с включенным модулем `FastRouter`. Дополнительные сведения о такой эксплуатации uWSGI см. ссылки в комментариях.

```
http {
  # запустить сервер uWSGI, к которому мы будем подключаться
  # uwsgi --master --emperor /etc/djangoapps --fastrouter 127.0.0.1:3017
  # --fastrouter-subscription-server 127.0.0.1:3032
  # см. http://uwsgi-docs.readthedocs.org/en/latest/Emperor. html
  # и http://projects.unbit.it/uwsgi/wiki/Example
  upstream emperor {
    server 127.0.0.1:3017;
  }
```

```
server {
  # при задании корня документов используется переменная, чтобы
  # можно было обслуживать несколько сайтов - отметим, что все
  # статические файлы должны находиться в подкаталоге "static",
  # а все закачанные пользователями файлы - в подкаталоге "media"
  # см. https://docs.djangoproject.com/en/dev/howto/static-files/

  root /home/www/sites/$host;
    location / {
    # CSS files are found under the "styles" subdirectory
      location ~* ^.+\.$ {
         root /home/www/sites/$host/static/styles;
         expires 30d;
       }
      # если путь не найден в корне документов, то запрос
      # передается Django, работающему под управлением
      try_files $uri @django;
    }
    location @django {
    # переменная $document_root должна указывать на код приложения
    root /home/www/apps/$host;
    # файл uwsgi_params входит в состав дистрибутива nginx
    include uwsgi_params;
    # ссылаемся на определенную выше секцию upstream
    # сервер uWSGI работает в режиме Emperor с модулем FastRouter
    uwsgi_param UWSGI_FASTROUTER_KEY $host;

   uwsgi_pass emperor.

  }
  # файл robots.txt находится в подкаталоге "static"
  # задание точного соответствия ускоряет поиск
   location = /robots.txt {
      root /home/www/sites/$host/static;
      access_log off;
      }
  # еще одно точное соответствие
  location = /favicon.ico {

      error_page 404 = @empty;
      root /home/www/sites/$host/static;
      access_log off;
      expires 30d;
  }
  # генерируем пустое изображение, на которое есть ссылка выше
  location @empty {
```

```
    empty_gif;
  }

  # попытка напрямую обратиться к файлу с расширением '.py'
  # приводит к ошибке 404 (Not Found)

   location ~* ^.+\.py$ {
      return 404;
    }
  }
}
```

При такой конфигурации можно динамически обслуживать несколько сайтов, ничего не меняя в настройках.

Резюме

В этой главе мы рассмотрели директивы NGINX, описывающие порядок обслуживания запросов но протоколу HTTP. Эту функциональность предоставляет не только сам модуль `http`, но и ряд вспомогательных модулей, необходимых для нормальной работы NGINX. По умолчанию эти модули включены. Комбинирование директив, принадлежащих разным модулям, позволяет построить конфигурацию, отвечающую нашим требованиям. Мы показали, как NGINX ищет файлы, соответствующие запрошенному URI. Мы обсудили различные директивы, управляющие взаимодействием HTTP-сервера с клиентами, и узнали о различных способах применения директивы `error_page`. Мы убедились, что NGINX позволяет ограничивать доступ, основываясь на потреблении полосы пропускания, скорости отдачи содержимого и количестве соединений.

Мы также видели, как разрешить доступ только с определенных IP-адресов или пользователям, прошедшим аутентификацию. Мы рассмотрели средства протоколирования NGINX и разобрались, как включать в журналы лишь интересующую нас информацию. Мы также бегло остановились на псевдопотоковой передаче. NGINX предоставляет ряд переменных, которые можно использовать в конфигурационном файле. Была описана возможность использования модуля `fastcgi` для подключения к PHP-FPM-приложениям и модуля `uwsgi` для взаимодействия с сервером uWSGI. В примерах было продемонстрировано применение директив, обсуждавшихся как в этой, так и в других главах.

В следующей главе мы познакомимся с модулями, которые помогают интегрировать NGINX с собственным приложением.

Глава 7. NGINX для разработчика

До сих пор мы рассматривали настройку NGINX для решения различных задач. Но пока ни слова не было сказано о том, что NGINX предлагает разработчикам приложений. Существует несколько способов интегрировать NGINX с приложением. Мы обсудим их в следующих разделах.

- Интеграция с механизмом кэширования.
- Динамическое изменение содержимого.
- Включение на стороне сервера.
- Принятие решений в NGINX.
- Создание безопасной ссылки.
- Генерация изображений.
- Отслеживание посетителей сайта.
- Предотвращение случайного выполнения кода.

Интеграция с механизмом кэширования

NGINX великолепно справляется с обслуживанием статического содержимого. Она спроектирована для поддержки свыше 100 000 одновременных соединений при минимальном потреблении системных ресурсов. Интеграция динамического веб-приложения с таким замечательным сервером может резко повысить его производительность. Возможно, мы не сумеем поддержать так много одновременных соединений, но уменьшить время реакции системы на действия пользователя мы все-таки сможем.

О кэшировании мы рассказывали в главе 5 «Обратное проксирование, дополнительные вопросы». В этом разделе мы более подробно остановимся на интеграции механизма кэширования NGINX с веб-приложением. Возможно, ваше приложение уже умеет что-то кэшировать. Быть может, оно записывает заранее сформированные

страницы в базу данных, чтобы не выполнять накладную процедуру построения страницы снова и снова. Или, что еще лучше, сохраняет сформированные страницы в файловой системе, позволяя NG1NX проявить свои поразительные способности в деле обслуживания статических файлов. Какой бы механизм кэширования ни использовался в вашем приложении (даже если его нет вовсе), NGINX позволит интегрировать его с сервером.

Приложения без кэширования

Даже если приложение вообще ничего не кэширует, NGINX все равно может ускорить время реакции. Поскольку модули `proxy` и `fastcgi` пользуются механизмом кэширования, то вам остается настроить кэширование для своего приложения с помощью директив `proxy_cache_*` или `fastcgi_cache_*`. Директивы `proxy_cache_*` описаны в разделе «Кэширование» главы 5, а директивы `fastcgi_cache_*` - в главе 6.

Здесь мы расскажем о том, как ваше приложение может попросить NGINX кэшировать отдельные страницы. Для этого нужно отправить NGINX определенные заголовки. Можно использовать как стандартные заголовки `Expires` или `Cache-Control`, так и специальный заголовок `x-Accel-Expires`, который NGINX интерпретирует самостоятельно и не передает клиенту. Этот заголовок позволяет приложению точно управлять временем кэширования файла и, в частности, прекратить кэширование объектов, которые обычно находятся в кэше очень долго.

Предположим, что имеется новостное приложение, страницы которого загружаются слишком медленно. Причин может быть много, но анализ показал, что каждая страница всякий раз строится заново, что сопровождается чтением содержимого из базы данных. Поэтому, когда заходит пользователь, приложение должно подключиться к базе данных, выполнить несколько SQL-запросов, разобрать результаты и только потом отправить сформированную страницу. Из-за многочисленных связей с системой, стоящей за этим веб-приложением, изменить архитектуру и реализовать более разумную стратегию построения страниц не так-то просто.

С учетом имеющихся ограничений вы решаете остановиться на следующей стратегии кэширования.

- Лицевая страница должна кэшироваться одну минуту, поскольку содержит часто изменяющийся список ссылок на статьи.
- Каждая статья кэшируется один день, поскольку, будучи опубликована, она уже не изменяется, но вместе с тем мы не хотим

забивать кэш старыми данными, которые придется удалять ввиду нехватки места.

- Изображения кэшируются столько времени, сколько возможно, поскольку они хранятся в базе данных и извлекать их оттуда накладно.

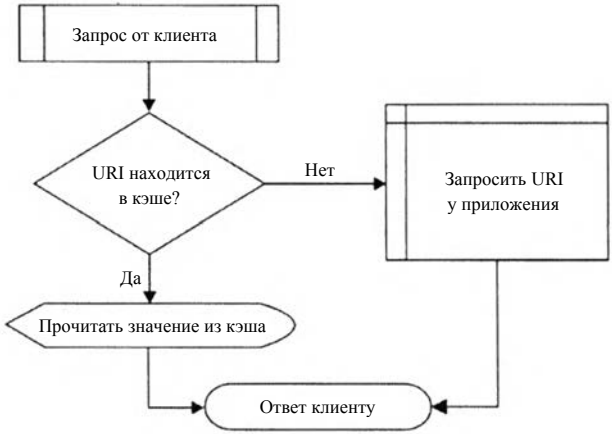
Для поддержки этой стратегии мы сконфигурируем NGINX следующим образом:

```
http {
  # настраиваем две зоны памяти для ключей и метаданных и пути к
  # каталогам, в которых будут храниться сами кэшированные объекты

proxy_cache_path /var/spool/nginx/articles keys_zone=ARTICLES:16m
  levels=1:2 inactive=1d;
proxy_cache_path /var/spool/nginx/images keys_zone=IMAGES:128m
  levels=1:2 inactive=30d;
  # но оба каталога находятся в той же файловой системе,
  #  что  и  proxy_temp_path

  proxy_temp_path /var/spool/nginx;

  server {
    location / {
    # здесь находится список статей
       proxy_cache_valid 1m;
    }

    location /articles {
      # URL каждой статьи начинается с "/articles"
       proxy_cache_valid 1d;
    }
      location /img {
      # URL любого изображения начинается с "/img"
       proxy_cache_valid 10y;
    }
  }
```

Тем самым наши требования удовлетворены. Мы реализовали кэширование для унаследованной системы, которая кэширование не поддерживала.

Кэширование в базе данных

Если приложение кэширует сформированные страницы в базе данных, то, скорее всего, будет нетрудно хранить их в кэше memcached. NGINX может обслуживать запросы непосредственно из этого кэша. Такая логика показана на рисунке ниже.



Интерфейс очень прост, но может быть настолько гибким, насколько необходимо. NGINX ищет ключ в хранилище. Если находит, то возвращает значение клиенту. Построение правильного ключа - задача настройки, которую мы обсудим ниже. Сохранение ассоциированного с ключом значения - задача не NGINX, а приложения.

Решить, какой ключ использовать, не слишком сложно. Для неперсонализированных ресурсов лучше всего использовать сам URI. Ключ записывается в переменную `$memcached_key`:

```
location / {
  set $memcached_key $uri;
  memcached_pass 127.0.0.1:11211;
}
```

Если для построения страницы приложение читает аргументы запроса, то их нужно включать и в ключ `$memcached_key`:

```
location / {
  set $memcached_key "$uri?$args";
  memcached_pass 127.0.0.1:11211;
}
```

Если ключ не найден, то NGINX нужны какие-то средства для запроса страницы у приложения. Хочется надеяться, что приложение затем поместит пару ключ-значение в кэш memcached, чтобы следующий запрос можно было обслужить из памяти. NGINX вернет ошибку «Not Found», если ключ отсутствует в memcached, поэтому самый правильный способ переадресовать запрос приложению -

воспользоваться директивой error_page и завести специальное местоположение для обработки запроса. Мы должны включить сюда же коды ошибок «Bad Gateway» и «Gateway Timeout» па случай, если memcached не отвечает на запрос о наличии ключа:

```
server {
  location / {
    set $memcached_key "$uri?$args";
    memcached_pass 127.0.0.1:11211;
    error_page 404 502 504 = @app;
  }

  location @app {
    proxy_pass 127.0.0.1:8080;
  }
}
```

Напомним, что увидев знак равенства (=) в аргументах директивы error_page, NGINX подставит код, возвращенный страницей, которая указана в последнем аргументе. Это позволяет преобразовать ошибку в нормальный ответ.

В таблице ниже описаны директивы, относящиеся к модулю memcached, который включается в nginx по умолчанию:

Директивы модуля memcached

Директива	Описание
memcached_buffer_size	Размер буфера для ответа от memcached. Этот ответ синхронно отправляется клиенту
memcached_connect _timeout	Максимальное время, в течение которого NGINX ожидает установления соединения при отправке запроса серверу memcached
memcached_next_upstream	Определяет условия, при которых для ответа будет выбран следующий сервер memcached. Условия задаются с помощью следующих параметров: • error: произошла ошибка при взаимодействии с сервером memcached; • timeout: произошел таймаут при взаимодействии с сервером memcached; • invalid_response: сервер memcached вернул пустой или недопустимый ответ; • not_found: ключ отсутствует в кэше memcached; • off: запретить передачу запроса следующему серверу memcached
memcached_pass	Определяет имя или адрес сервера memcached и его порт. Можно также указывать группу серверов, объявленную в контексте upstream

Директива	Описание
memcached_read_timeout	Сколько времени может пройти между двумя последовательными операциями чтения данных от сервера memcached, прежде чем соединение будет закрыто
memcached_send_timeout	Сколько времени может пройти между двумя последовательными операциями записи данных на сервер memcached, прежде чем соединение будет закрыто

Кэширование в файловой системе

Предположим, что приложение записывает сформированные страницы в файлы. Поскольку вы знаете, сколько времени файл остается актуальным, то можете настроить NGINX так, чтобы она отправляла клиенту вместе с файлом заголовки, содержащие инструкции самому клиенту и прокси-серверам на пути к нему о том, как долго хранить файл в кэше. Там самым вы разрешите использовать локальный кэш клиента, не изменив ни единой строки кода в приложении.

Для этого следует настроить заголовки `Expires` и `Cache-Control`. Это стандартные HTTP-заголовки, которые понимают клиенты и прокси-серверы. В приложение никаких изменений вносить не нужно, достаточно прописать эти заголовки в конфигурационном блоке NGINX для соответствующего местоположения. NGINX упрощает эту задачу, предоставляя директивы `expires` и `add_header`.

Директивы для модификации заголовков

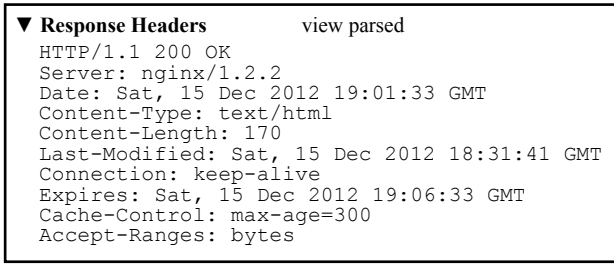
Директива	Описание
add_header	Добавляет заголовок в ответ с кодом состояния 200, 204, 206, 301, 302, 303, 304 или 307
expires	Добавляет или изменяет заголовки `Expires` и `Cache-Control`. Может быть задан необязательный параметр `modified`, за которым следует время (параметр `time`) или одно из ключевых слов `epoch`, `max` либо `off`. Если присутствует только `time`, то значением в заголовке `Expires` будет текущее время плюс время, указанное в параметре `time`. Значение в заголовке `Cache-Control` будет равно `max-age=t`, где `t` время в секундах, указанное в качестве аргумента. Если параметру `time` предшествует параметр `modified`, то в заголовок `Expires` записывается время модификации файла плюс время, указанное в параметре `time`. Если параметр `time` содержит знак `@`, то указанное время интерпретируется как время суток; например, `@12h` - это полдень. Параметр epoch соответствует абсолютному времени `Thu`, 01 Jan 1970 00:00:01 GMT. Если задан параметр `max`, то в заголовок Expires записывается значение `Thu, 31 Dec 2037 23:55:55 GMT`, а `Cache-Control` устанавливается на 10 лет. Отрицательное значение времени приводит к записи в `Cache-Control` значения `no-cache`

Зная, как поступать с файлами, которые генерирует ваше приложение, вы можете правильно настроить эти заголовки. Возьмем в качестве примера приложение, для которого главная страница должна кэшироваться 5 минут, все JavaScript и CSS-файлы - 24 часа, все HTML-страницы - 3 дня, а изображения - так долго, как возможно.

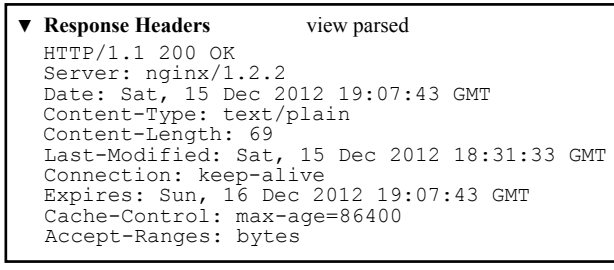
```
server {
  root /home/www;

  location / {
    # index.html сопоставляется явно, так что регулярное выражение
    # *.html ниже не будет соответствовать главной странице
    location = /index.html {
       expires 5m;
    }
    # для файлов с расширениями .js или .css (Javascript и CSS)
    location ~* /.*\.(js|css)$ {
       expires 24h;
    }
    # все страницы с расширением .html
    location ~* /.*\.html$ {
       expires 3d;
    }
  }
    # все изображения находятся в отдельном местоположении (/img)
    location /img {
       expires max;
    }
  }
```

Чтобы понять, как при такой конфигурации устанавливаются заголовки, посмотрим, как выглядит каждое местоположение в браузере. В любом современном браузере имеется встроенное или подключаемое дополнительно средство для просмотра заголовков запроса и ответа. На снимках экрана ниже показано, как заголовки ответов для разных местоположений отображаются в Chrome.

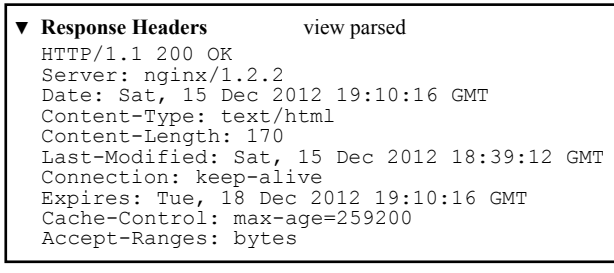
- **Главная страница** (index.html): в заголовке `Expires` установлено время, на 5 минут позже того, что указано в заголовке `Date`. Параметр `max-age` в заголовке `Cache-Control` равен 300 секунд.



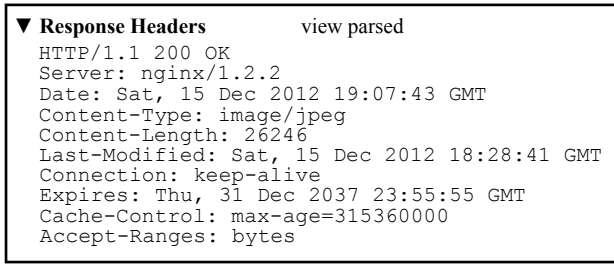
- **CSS-файл**: в заголовке `Expires` установлено время, на 24 часа позже того, что указано в заголовке `Date`. Параметр `max-age` в заголовке `Cache-Control` равен 86400 секунд.



- **HTML-файл**: в заголовке `Expires` установлено время, на 3 дня позже того, что указано в заголовке `Date`. Параметр `max-age` в заголовке `Cache-Control` равен 259 200 секунд.



- **Изображение**: в заголовке `Expires` установлено время `Thu, 31 Dec 2037 23:55:55 GMT`. Параметр `max-age` в заголовке `Cache-Control` равен 315 360 000 секунд.



Поместив всего лишь одну директиву `expires` в нужное местоположение, мы можем настроить кэширование файлов в течение необходимого времени.

Динамическое изменение содержимого

Иногда бывает полезно подвергнуть дополнительной обработке данные, поступившие от приложения. Быть может, мы хотели бы добавить на страницу строку, показывающую, какой фронтальный сервер доставил эту страницу клиенту. Или каким-либо образом преобразовать уже сформированную страницу. NGINX предлагает три модуля, которые могут быть в этом смысле полезны: `addition`, `sub` И `xslt`.

Модуль addition

Модуль `addition` работает как фильтр, который добавляет текст до и (или) после ответа. По умолчанию он не компилируется, поэтому если вам нужна эта функция, то ее надо включить на этапе конфигурирования с помощью параметра `--with-http_addition_module`.

Этот фильтр ссылается на подзапрос, результат которого добавляется в начало или в конец ответа:

```
server {
  root /home/www;

  location / {
    add_before_body /header;
```

```
  add_after_body /footer;
}

location /header {
  proxy_pass http://127.0.0.1:8080/header;
}

location /footer {
  proxy_pass http://127.0.0.1:8080/footer;
}
}
```

Директивы модуля `addition` перечислены в таблице ниже.

Директивы модуля addition

Директива	Описание
`add_before_body`	Поместить результат обработки подзапроса перед телом ответа
`add_after_body`	Поместить результат обработки подзапроса после тела ответа
`addition_types`	Указывается список MIME-типов ответа (в дополнение к `text/html`), для которых производится добавление. Звездочка (*) означает все MIME-типы

Модуль sub

Модуль `sub` работает как фильтр, который заменяет один текст другим. По умолчанию он не компилируется, поэтому если вам нужна эта функция, то ее надо включить на этапе конфигурирования с помощью параметра `--with-http_sub_module`.

Работать с ним несложно. В директиве `sub_filter` задается заменяемая и заменяющая строка, а фильтр ищет первую строку без учета регистра и подставляет вместо нее вторую:

```
location / {
  sub_filter </head> '<meta name="frontend" content="web3"></head>';
}
```

Здесь мы «на проходе» через NGINX добавили новый тег meta в заголовок страницы.

Можно также искать все совпадения. Для этого добавьте директиву `sub_filter_once`, указав в ней параметр `off`. Это может быть, например, полезно для замены всех относительных ссылок на странице абсолютными:

```
location / {
  sub_filter_once off;
  sub_filter '<img src="img/' '<img src="/img/';
}
```

Если искомая строка содержит пробелы или кавычки, то ее необходимо заключить в кавычки, чтобы NGINX распознала ее всю как первый параметр.

NGINX автоматически применяет директиву `sub_filter` к любому HTML-файлу. Если же требуется выполнить подстановку в файлах других типов, например JavaScript или CSS, то следует указать соответствующий MIME-тип с помощью директивы `sub_filter_types`.

```
location / {
  sub_filter_types text/css;
  sub_filter url(img/ 'url(/img/';
}
```

Поскольку тип `text/html` подразумевается по умолчанию, добавлять его не нужно - он не пропадет, если указать дополнительные MIME-типы. Это справедливо для всех директив задания MIME-типов, встречающихся в NGINX.

Все вышеупомянутые директивы сведены в следующую таблицу.

Директивы модуля sub

Директива	Описание
`sub_filter`	Задает заменяемую строку (без учета регистра) и заменяющую строку. Заменяющая строка может содержать переменные
`sub_filter_once`	Если параметр равен `off`, то директива `sub_filter` применяется ко всем вхождениям заменяемой строки
`sub_filter_types`	Указывается список MIME-типов ответа (в дополнение к `text/html`), для которых производится замена. Звездочка (*) означает все MIME-типы

Модуль xslt

Модуль `xslt` работает как фильтр, который преобразует XML-документ с помощью таблиц стилей XSLT. По умолчанию он не компилируется, поэтому если вам нужна эта функция, то необходимо установить библиотеки `libxml2` и `libxslt` и включить модуль на этапе конфигурирования с помощью параметра `--with-http_xslt_module`.

Для использования модуля `xslt` следует определить DTD-схему, в которой объявлены знаковые сущности. После этого задается одна или несколько таблиц стилей XSLT для обработки XML-документа вместе с параметрами:

```
location / {
  xml_entities /usr/local/share/dtd/entities.dtd;
  xsl_stylesheet /usr/local/share/xslt/style1.xslt;
  xsl_stylesheet /usr/local/share/xslt/style2.xslt theme=blue;
}
```

Директивы модуля `xslt` перечислены в следующей таблице.

Директивы модуля xslt

Директива	Описание
`xml_entities`	Путь к файлу DTD-схемы, где объявлены знаковые сущности, на которые есть ссылки в обрабатываемом XML-файле
`xslt_param`	Параметры, передаваемые в таблицы стилей; значениями являются выражения XPath
`xslt_string_param`	Параметры, передаваемые в таблицы стилей; значениями являются строки
`xslt_stylesheet`	Путь к таблице стилей XSLT, применяемой для преобразования XML-ответа. Можно передать параметры в виде списка пар ключ-значение
`xslt_types`	Указывается список MIME-типов ответа (в дополнение к text/xml), для которых производится замена. Звездочка (*) означает все MIME-типы. Если результатом преобразования является HTML-документ, то его MIME-тип будет заменен на text/html

Включение на стороне сервера

Модуль `ssi` также является фильтром, причем одним из самых гибких. Он разрешает использовать технологию Server Side Includes для встраивания логики обработки в веб-страницу. Поведение модуля управляется следующими директивами.

Директивы модуля ssi

Директива	Описание
`ssi`	Разрешает обработку SSI-файлов
`ssi_silent_errors`	Подавляет сообщение, которое обычно выводится в случае ошибки во время обработки SSI
`ssi_types`	Указывается список MIME-типов ответа (в дополнение к `text/html`), для которых обрабатываются команды SSI. Звездочка (*) означает все MIME-типы

Команды SSI, поддерживаемые NGINX, приведены в таблице ниже. Все они устроены по одному образцу:

```
<!--# command parameter1=value1 parameter2=value2 … -->
```

Команды SSI

Команда	Аргумент	Пояснение
block		Определяет секцию, на которую можно сослаться в команде include. Заканчивается командой <!--# endblock -->
	name	Имя блока
config		Устанавливает глобальные параметры, используемые во время обработки SSI
	errmsg	Задает строку, которая используется как сообщение об ошибке обработки SSI. По умолчанию [an error occurred while processing the directive]
	timefmt	Строка, передаваемая функции strftime() для форматирования временных меток в других командах. По умолчанию %A, %d-%b-%Y %H:%M:%S %Z
echo		Выводит значение переменной
	var	Имя выводимой переменной
	encoding	Метод кодирования переменной. Допустимы значения none, url и entity. По умолчанию entity
	default	Значение, которое выводится, если переменная не определена. Если не задано, по умолчанию выводится none
if		Вычисляет условие. Если true, то следующий далее блок включается. Глубина вложенности цепочки if, elsif, else и endif не более единицы
	Expr	Вычисляемое логическое выражение: • существование переменной (expr="$var"); • сравнение (expr="$var = text" или expr="$var !=text"); • сопоставление с регулярным выражением (expr="$var = /regexp/" или expr="$var != / regexp/")
include		Выводит результат подзапроса
	file	Имя включаемого файла
	virtual	URI подзапроса, результат которого включается
	stub	Блок, включаемый вместо пустого тела или в случае ошибки при обработке
	wait	Если этот параметр присутствует, то при наличии на одной странице нескольких команд include они будут обрабатываться последовательно
	set	Если подзапрос, указанный в параметре virtual, ссылается на местоположение с директивой proxy_pass или memcached_pass, то его результат можно сохранить в переменной, имя которой указано в параметре set

Команда	Аргумент	Пояснение
set		Создает переменную и присваивает ей значение
	var	Имя переменной
	value	Значение переменной

SSI-файл - это не что иное, как обычный HTML-файл с командами, помещенными внутрь комментариев. Таким образом, если для какого-то местоположения, содержащего такой файл, модуль ssi не активирован, то HTML-часть тем не менее будет отрисована, хотя страница окажется неполной.

Ниже приведен пример SSI-файла, в котором подзапросы используются для формирования верхнего и нижнего колонтитула, а также меню:

```
<html>
  <head>
    <title>*** SSI test page ***</title>
    <link rel="stylesheet" href="/css/layout.css" type="text/css"/>
      <!--# block name="boilerplate" -->
      <p>...</p>
      <!--# endblock -->
  </head>
  <body>
    <div id="header">
      <! # include virtual="/render/header?page=$uri" stub="boilerplate" -->
      </div>
    <div id="menu">
     <!--# include virtual="/render/menu?page=$uri" stub="boilerplate" -->
    </div>
    <div id="content">
      <p> Здесь находится содержимое страницы </p>
    </div>
    <div id="footer">
     <!--# include virtual="/render/footer?page=$uri" stub="boilerplate" -->
    </div>
  </body>
</html>
```

Команда stub применяется для отрисовки содержимого по умолчанию в случае, когда при обработке подзапроса произошла ошибка.

Если этих команд недостаточно для реализации задуманной логики, то можно воспользоваться модулем perl, который предоставляет встроенный интерпретатор языка Perl, - его должно хватить для решения практически любой задачи обработки или конфигурирования.

Принятие решений в NGINX

Иногда мы ловим себя на попытке использовать конфигурационные директивы NG1NX совершенно непредусмотренными способами. С этим часто приходится сталкиваться в конфигурациях, где производится много проверок в попытке эмулировать некую логическую цепочку. В этом случае лучше воспользоваться встроенным в NGINX модулем `perl`. Он предоставляет в ваше распоряжение всю гибкость языка Perl для создания нужной конфигурации.

Модуль `perl` по умолчанию не компилируется, поэтому для его включения необходимо указать на этапе конфигурирования параметр `--with-http_perl_module`. Кроме того, сам интерпретатор Perl должен быть откомпилирован с флагами `-Dusemultiplicity=yes` (или `-Dusethreads=yes`) и `-Dusemymalloc=no`. Когда NGINX перезагружает конфигурационный файл, происходит некоторая утечка памяти в модуле `perl`, и последний параметр в какой-то мере сглаживает эту проблему.

После включения в `nginx` встроенного интерпретатора Perl становятся доступны следующие директивы.

Директивы модуля perl

Директива	Описание
`perl`	Активирует обработчик Perl в данном местоположении. В аргументе указывается имя обработчика или строка, содержащая полный код подпрограммы
`perl_modules`	Определяет дополнительные пути поиска Perl-модулей
`perl_require`	Задает имя Perl-модуля, который должен загружаться при каждом изменении конфигурационного файла NGINX. Может встречаться несколько раз, если требуется указать несколько модулей
`perl_set`	Устанавливает Perl-обработчик, который присваивает значение переменной. В аргументе указывается имя обработчика или строка, содержащая полный код подпрограммы

В Perl-скриптах, используемых в конфигурационном файле NGINX, доступен объект `$r`, представляющий запрос. У этого объекта имеются следующие методы:

- `$r->args`: аргументы запроса;
- `$r->filename`: имя файла, па который ссылается URI;
- `$r->has_request_body(handler)`: этот обработчик вызывается, если у запроса имеется тело;
- `$r->allow_ranges`: разрешает использование в ответе диапазонов байтов;

- `$r->discard_request_body`: отбрасывает тело запроса;
- `$r->header_in(header)`: значение в указанном заголовке запроса;
- `$r->header_only`: говорит NGINX, что клиенту нужно вернуть только заголовки;
- `$r->header_out(header, value)`: записывает в указанный заголовок ответа указанное значение;
- `$r->internal_redirect(uri)`: производит внутреннюю переадресацию на указанный URI, после того как Perl-обработчик закончит выполнение;
- `$r->print(text)`: помещает в ответ клиенту указанный текст;
- `$r->request_body`: тело запроса, если оно умещается в памяти;
- `$r->request_body_file`: тело запроса, если оно сохранено во временном файле;
- `$r->request_method`: HTTP-метод запроса;
- `$r->remote_addr`: IP-адрес клиента;
- `$r->flush`: немедленно отправляет данные клиенту;
- `$r->sendfile(name[, offset[, length]])`: отправляет клиенту указанный файл, после того как Perl-обработчик закончит выполнение (можно задать необязательное смещение от начала и длину);
- `$r->send_http_header([type])`: отправляет клиенту заголовки ответа (можно задать необязательный тип содержимого);
- `$r->status(code)`: устанавливает в ответе HTTP-код состояния;
- `$r->sleep(milliseconds, handler)`: взводит таймер на указанное количество миллисекунд. После срабатывания таймера NGINX выполнит обработчик, а до этого времени продолжает обработку других запросов;
- `$r->unescape(text)`: декодирует URI-кодированный текст;
- `$r->uri`: URI, указанный в запросе;
- `$r->variable(name[, value])`: либо возвращает именованную, локальную для запроса переменную, либо присваивает ей указанное значение.

Модуль `perl` можно использовать также в сочетании с SSI. В команде SSI можно следующим образом указать код, написанный па Perl:

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

Рассмотрим пример работы с модулем `perl`. Наша цель - передать запрос проксируемому серверу, определяемому первой буквой

указанного в запросе URI-адрсса. Это можно было бы сделать, определив несколько местоположений в NGINX, но при использовании Perl-обработчика конфигурация получается короче.

Первый шаг - определить действия в Perl-обработчике:

```
# upstreammapper.pm

# имя пакета
package upstreammapper;

# включаем определения методов объекта запроса в nginx
use nginx;

# эта подпрограмма вызывается из nginx

  sub handler {
  my $r = shift;
  my @alpha = ("a".."z");

  my %upstreams = ();
  # не мудрствуя лукаво, описываем соответствие между буквой и
  # IP-адресом, последний октет которого изменяется от 10 до 35

  foreach my $idx (0..$#alpha) {
     $upstreams{ $alpha[$idx] } = $idx + 10;

  }
  # создаем массив, содержащий все символы URI-адреса
  my @uri = split(//,$r->uri);
  # чтобы можно было использовать первую букву в качестве клоча

  my $ip = "10.100.0." . $upstreams{ $uri[1] };
     return $ip;
  }
  1;
  __END__
```

Теперь настраиваем NGINX, так чтобы этот модуль использовался для определения подходящего сервера:

```
http {
  # путь относительно главного конфигурационного файла
  perl_modules perl/lib;
  perl_require upstreammapper.pm;

  # результат возвращенный обработчиком, сохраняется в переменной
  # $upstream
  perl_set $upstream upstreammapper::handler;
```

Передаем запрос нужному проксируемому серверу:

```
 location / {
   include proxy.conf;
   proxy_pass http://$upstream;
 }
}
```

Это очень простой пример реализации логики конфигурирования в Perl-обработчике. Но точно так же можно решить и другие подобные задачи.

Perl-обработчик должен быть максимально сфокусирован на решении конкретной задачи. Пока NGINX ожидает его завершения, весь рабочий процесс, в котором обрабатывается данный запрос, блокирован. Поэтому ввод-вывод и обращения к DNS следует выполнять вне Perl-обработчика.

Создание безопасной ссылки

Иногда требуется защитить часть содержимого сайта, но подключать для этого механизм полноценной аутентификации не хочется. Один из способов решить задачу - воспользоваться модулем `secure_link`. Для его включения в NGINX необходимо на этапе конфигурирования задать параметр `--with-http_secure_link`. В результате становится доступна директива `secure_link_secret` и переменная `$secure_link`.

Принцип работы модуля `secure_link` заключается в вычислении MD5-свертки ссылки, конкатенированной с секретным словом. Если свертка совпадает с указанной в URI, то в переменную `$secure_link` записывается часть URI после свертки. В противном случае переменная `$secure_link` будет содержать пустую строку.

Одно из возможных применений - сгенерировать страницу со ссылками на скачиваемые файлы, используя секретное слово. Это слово записывается в конфигурационный файл NGINX и позволяет проверить, разрешен ли доступ к ссылкам. И слово, и сама страница время от времени меняются, чтобы воспрепятствовать доступу по сохраненной ссылке в будущем. Этот сценарий иллюстрируется в примере ниже.

Сначала выбираем секретное слово - `supersecret`. Затем генерируем MD5-свертки интересующих нас ссылок:

```
$ echo -n "alphabet_soup.pdfsupersecret" |md5sum
8082202b04066a49a1ae8da9ec4feba1 -
$ echo -n "time_again.pdfsupersecret" |md5sum
5b77faadb4f5886c2ffb81900a6b3a43 -
```

Теперь можно поместить ссылки в HTML-разметку:

```
<a href="/downloads/8082202b04066a49a1ae8da9ec4feba1/alphabet_soup.
pdf">alphabet soup</a>
<a href="/downloads/5b77faadb4f5886c2ffb81900a6b3a43/time_again.
pdf">time again</a>
```

Эти ссылки действительны, только если в директиве `secure_link_secret` конфигурационном файле указано то же секретное слово, которое использовалось для генерации сверток:

```
# доступ к ресурсам, URI-адреса которых начинаются с /downloads/, защищен
location /downloads/ {
  # это строка, с помощью которой генерировались свертки
  secure_link_secret supersecret;

  # запретить доступ, если свертки не совпадают

    if ($secure_link = "") {
  return 403;
  }
  try_files /downloads/$secure_link =404;
  }
```

Чтобы убедиться, что ссылка без свертки не работает, добавим на HTML-страницу еще одну ссылку:

```
<a href="/downloads/bare_link.pdf">bare link</a>
```

Попытка перейти по ней приводит к ошибке «403 Forbidden», как и ожидалось.

Описанная выше техника с применением модуля `secure_link` - лишь один из способов решения поставленной задачи. Даже в самой NGINX имеется альтернативный подход, описанный на странице http://wiki.nginx.org/HttpSecureLinkModule

Генерация изображений

Вместо того чтобы самостоятельно писать модуль манипулирования изображениями, вы можете поручить NGINX некоторые простые преобразования: поворот, изменение размера или обрезание.

Чтобы воспользоваться этой функциональностью, необходимо установить библиотеку `libgd` и откомпилировать модуль `image_filter`, задав на этапе конфигурирования параметр `--with-http_image_filter_module`. В результате становятся доступны директивы, перечисленные в таблице ниже.

Библиотека GD (`libgd`) предназначена для манипулирования изображениями и написана на C. Она нередко используется в сочетании с такими языками программирования, как PHP или Perl с целью генерации изображений для вебсайтов. В NGINX модуль `image_fliter` обращается к `libgd` для создания простого механизма изменения размера изображений, который будет рассмотрен в примере ниже.

Директивы модуля image_filter

Директива	Описание
`empty_gif`	Порождает прозрачный GIF размером 1x1 для данного местоположения
`image_filter`	Преобразует изображение в соответствии с одним из следующих параметров: • `off`: отключить преобразование изображений; • `test`: проверяет, что ответ представлен в формате GIF, JPEG или PNG. В противном случае возвращается ошибка 415 (Unsupported Media Type); • `size`: выводит информацию об изображении в формате JSON; • `rotate`: поворачивает изображение против часовой стрелки на угол 90, 180 или 270 градусов; • `resize`: пропорционально уменьшает изображение до указанных размеров. Если требуется уменьшить только по одному измерению, то в качестве второго можно указать «-». При использовании в сочетании с `rotate` поворот производится сначала. В случае ошибки возвращается код 415 (Unsupported Media Type); • `crop`: уменьшает изображение до размера большей из двух заданных сторон и обрезает лишние края по другой стороне. Если требуется уменьшить только по одному измерению, то в качестве второго можно указать При использовании в сочетании с rotate поворот производится сначала. В случае ошибки возвращается код 415 (Unsupported Media Type)
`image_filter_buffer`	Размер буфера для обработки изображений. Если потребуется больше памяти, сервер вернет ошибку 415 (Unsupported Media Type)

Директива	Описание
image_filter_jpeg_quality	Качество результирующего изображения в формате JPEG. Не рекомендуется указывать значение больше 95
image_filter_sharpen	Повышает резкость результирующего изображения на указанный процент
image_filter_transparency	Определяет, сохранять ли прозрачность при обработке изображений в форматах PNG и GIF. Подразумеваемое по умолчанию значение on означает сохранение прозрачности

Отметим, что директива empty_gif не является частью модуля image_fiIter, а включается в NGINX по умолчанию.

С помощью этих директив мы можем следующим образом построить механизм изменения размера изображений:

```
location /img {
  try_files $uri /resize/$uri;
}

location ~* /resize/(?.<name>.*)_(?<width>[[:digit:]]*)
x(?<height>[[:digit:]]*)\.(?<extension>gif|jpe?g|png)$ {
  error_page 404 = /resizer/$name.$extension?width=$width&height=$
} height;

location /resizer {
  image_filter resize $arg_width $arg_height;
}
```

Здесь мы сначала пытаемся вернуть изображение, запрошенное в URI. Если файла с таким именем нет, то мы переходим в местоположение /resize, где определено регулярное выражение, выделяющее из имени файла требуемый размер изображения. Обратите внимание на именованные запоминаемые группы, позволяющие создавать переменные с осмысленными именами. Затем выделенные размеры передаются в местоположение /resizer, так что мы имеем исходное имя файла в URI, а ширину и высоту - в виде именованных аргументов.

Теперь все это можно объединить с директивой proxy_store или proxy_cache, чтобы сохранить изображение с новыми размерами. Тогда при следующем обращении к тому же URI «дергать» модуль image_filter не понадобится:

```
server {
  root /home/www;

  location /img {
```

```
    try_files $uri /resize/$uri;
  }

  location /resize {
    error_page 404 = @resizer;
  }

  location @resizer {
    internal;
    proxy_pass http://localhost:8080$uri;
    proxy_store /home/www/img$request_uri;
    proxy_temp_path /home/www/tmp/proxy_temp;
  }

}

  server {
  listen 8080;
  root /home/www/img;
  location ~* /resize/(?.<name>.*)_(?<width>[[:digit:]]*)
  x(?<height>[[:digit:]]*)\.(?<extension>gif|jpe?g|png)$ {
    error_page 404 = /resizer/$name.$extension?width=$width&heigh
    t=$height;
  }
  location /resizer {
     image_filter resize $arg_width $arg_height;
  }
}
```

Из таблицы директив модуля `image_filter` видно, что любая ошибка приводит к возврату кода 415. Мы можем перехватить этот код и вернуть в этом случае пустой GIF, так что пользователь все-таки получит изображение, а не сообщение об ошибке:

```
location /thumbnail {
  image_filter resize 90 90;
  error_page 415 = @empty;
}

location = @empty {
  access_log off;
  empty_gif;
}
```

На параметре `size` модуля `image_filter` стоит остановиться особо. Если в некотором местоположении указан этот параметр, то возвращается не само изображение, а сведения о нем. Это может быть полезно, когда приложению нужно получить метаданные изображения, перед тем как производить изменение размера или обрезание:

```
location /img {
  image_filter size;
}
```

Результатом является JSON-объект такого вида:

```
{ "img" : { "width": 150, "height": 200, "type": "png" } }
```

Отслеживание посетителей сайта

Модуль `userid` предлагает ненавязчивый способ отслеживать посетителей сайта. Он устанавливает куки, идентифицирующие уникальных клиентов. Значение кука сохраняется в переменной `$uid_set`. Когда пользователь снова зайдет на сайт со все еще действительным куком, значение кука можно будет получить из переменной `$uid_got`. Ниже приведен пример использования.

```
http {
  log_format useridcomb '$remote_addr - $uid_got [$time_local] '
                        '"$request" $status $body_bytes_sent '
                        '"$http_referer" "$http_user_agent"';
  server {
    server_name .example.com;
    access_log logs/example.com-access.log useridcomb;
    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"';
  }
}
```

Все эти директивы перечислены в таблице ниже.

Директивы модуля userid

Директива	Описание
`userid`	Активирует модуль в соответствии со следующими параметрами: • `on`: устанавливает куки версии 2 и записывает в журнал полученные куки; • `v1`: устанавливает куки версии 1 и записывает в журнал полученные куки; • `log`: отключает установку куков, но продолжает записывать их в журнал; • `off`: отключает как установку, так и протоколирование куков

Директива	Описание
`userid_domain`	Настраивает записываемый в куки домен
`userid_expires`	Задает срок хранения кука. Если указано значение `max`, то устанавливается дата `31 Dec 2037 23:55:55 GMT`
`userid_name`	Задает имя кука (по умолчанию `uid`).
`userid_p3p`	Настраивает заголовок P3P для сайтов, которые объявляют свою политику конфиденциальности, следуя протоколу **Platform for Privacy Preferences Project**
`userid_path`	Задает путь, указываемый в куке.
`userid_service`	Идентификатор службы, которая устанавливает кук. Например, для куков версии 2 по умолчанию подразумевается IP-адрес сервера, установившего кук

Предотвращение случайного выполнения кода

Иногда при разработке конфигурационного файла можно случайно включить не то, что предполагалось. Рассмотрим, к примеру, такой конфигурационный блок:

```
location ~* \.php {
  include fastcgi_params;
  fastcgi_pass 127.0.0.1:9000;
}
```

На первый взгляд, мы здесь просто передаем все запросы к PHP-файлам FastCGI-серверу, который отвечает за их разработку. И все было бы хорошо, если бы PHP обрабатывала только файлы, являющиеся частью приложения. Однако из-за различий в способе компиляции и настройки PHP так бывает не всегда. Это может оказаться проблемой, если загружаемые пользователями файлы попадают в ту же часть файловой системы, где находятся PHP-файлы.

Можно запретить пользователям загружать файлы с расширением php, разрешив только расширения `.jpg`, `.png` и `.gif`. Но злонамеренный пользователь мог бы загрузить графический файл с внедренным PHP-кодом и заставить FastCGI-сервер выполнить этот код, передав URI, в котором указано имя загруженного файла.

Чтобы избежать этого, нужно либо присвоить параметру PHP `cgi.fix_pathinfo` значение 0, либо модифицировать конфигурацию следующим образом:

```
location ~* \.php {
  try_files $uri =404;
  include fastcgi_params;
  fastcgi_pass 127.0.0.1:9000;
}
```

Здесь мы с помощью директивы `try_files` проверяем существование файла, перед тем как передавать запрос FastCGI-серверу для обработки PHP-приложением.

Не забывайте анализировать конфигурацию на предмет соответствия поставленным целям. Если имеется всего несколько файлов, то, быть может, лучше явно перечислить, какие PHP-файлы разрешено исполнять, чем описывать секцию `location` регулярным выражением и добавлять директивы `try_files`.

Резюме

NGINX предлагает различные способы интеграции приложения с высокопроизводительным веб-сервером. Мы рассмотрели несколько таких возможностей как для унаследованных, так и для вновь разрабатываемых приложений. Кэширование играет важную роль в любом современном веб-приложении. В NGINX реализованы пассивные и активные механизмы кэширования с целью более быстрого возврата веб-страниц.

Мы также показали, как с помощью NGINX изменить ответ, добавив или заменив текст. NGINX поддерживает также технологию включения на стороне сервера - SSI. Мы видели, как команды SSI вставляются в обычную HTML-разметку. Затем мы рассмотрели мощные возможности встроенного в NGINX интерпретатора Perl. С помощью стандартных средств NG1NX возможно также выполнять преобразование изображений. Мы обсудили установку уникальных куков для отслеживания посетителей веб-сайта. И завершили главу предостережением - как предотвратить непреднамеренное выполнение кода. В общем и целом, используя NGINX в качестве веб-сервера, разработчик получает немало полезных инструментов.

В следующей главе мы поговорим о технике поиска неполадок и объясним, как поступать, когда что-то идет вразрез с ожиданиями.

Глава 8. Техника устранения неполадок

Наш мир несовершенен. Вопреки самым лучшим намерениям и тщательному планированию иногда что-то идет не так, как мы рассчитывали. Тогда нужно на шаг отступить и выяснить, что случилось. Если с первого взгляда причина неясна, то придется обратиться к методикам исследования проблемы. Процедура, позволяющая понять, в чем состоит ошибка и как ее исправить, и называется устранением неполадок.

В этой главе мы рассмотрим различные приемы устранения неполадок при работе с NGINX:

- Анализ журналов.
- Настройка расширенного протоколирования.
- Типичные ошибки конфигурирования.
- Ограничения операционной системы.
- Проблемы с производительностью.
- Использование модуля Stub Status.

Анализ журналов

Прежде чем приступать к длительному сеансу отладки, пытаясь понять, в чем причина проблемы, полезно заглянуть в журналы. Часто в них можно найти ключ, который поможет проследить истоки ошибки и исправить ее. Но сообщения в файле `error_log` иногда выглядят загадочно, поэтому обсудим формат журнальных записей и на примерах покажем, как их интерпретировать.

Форматы записей в журнале ошибок

Для записи в журнал `error_log` в NGINX применяются две разные функции. Форматы в них устроены следующим образом:

<временная метка> [уровень-протоколирования] <pid главного/рабочего процесса>#0: сообщение

Например:

```
2012/10/14 18:56:41 [notice] 2761#0: using inherited sockets from "6;"
```

Это пример информационного сообщения (уровень протоколирования `notice`). В нем говорится, что процесс nginx заменил исполнявшийся ранее и успешно унаследовал сокеты от старого процесса.

В случае ошибки порождается сообщение такого вида:

```
2012/10/14 18:50:34 [error] 2632#0: *1 open() "/opt/nginx/html/blog"
failed (2: No such file or directory), client: 127.0.0.1, server:www.
example.com, request: "GET /blog HTTP/1.0", host: "www.example.com"
```

В зависимости от ошибки может быть выведено либо сообщение операционной системы (как в данном случае), так и самой NGINX. В этом сообщении можно выделить следующие компоненты:

- временная метка (`2012/10/14 18:50:34`);
- уровень протоколирования (`error`);
- идентификатор рабочего процесса (`2632`);
- номер соединения (`1`);
- системный вызов (`open`);
- аргумент системного вызова (`/opt/nginx/htmi/blog`);
- сообщение об ошибке, возвращенное системным вызовом (`2: No such file or directory`);
- IP-адрес клиента, который отправил запрос, приведший к ошибке (`127.0.0.1`);
- сервер, обслуживавший данный запрос (`www.example.com`);
- сам запрос (`GET /blog HTTP/1.0`);
- заголовок Host запроса (`www.example.com`).

Ниже приведен пример критической ошибки:

```
2012/10/14 19:11:50 [crit] 3142#0: the changing binary signal is ignored:
you should shutdown or terminate before either old or new binary's process
```

Критическая ошибка означает, что NGINX не смогла выполнить запрошенное действие. Если в этот момент NGINX еще не работала, значит, она и не запустилась.

Ниже приведен пример сообщения о чрезвычайной ошибке:

```
2012/10/14 19:12:05 [emerg] 3195#0: bind() to 0.0.0.0:80 failed
(98: Address already in use)
```

Чрезвычайная ошибка также означает, что NGINX не смогла сделать то, что ее просили. Если это произошло при попытке запуска, NGINX не запустилась, а если по время работы в отпет на требование перечитать конфигурационный файл, значит, запрошенное изменение не выполнено.

Если вас интересует, почему после модификации конфигурационного файла ничего не изменилось, загляните в журнал ошибок. Скорее всего, NGINX обнаружила ошибку в файле и не применила изменение.

Примеры записей в журнале ошибок

Ниже приведены примеры сообщений об ошибках, взятые из реальных журналов. После каждого примера дается краткое пояснение. Отметим, что в ваших журналах текст сообщения может отличаться в связи с улучшениями в новых версиях NGINX.

```
2012/11/29 21:31:34 [error] 6338#0: *1 upstream prematurely
closed connection while reading response header from upstream,
client: 127.0.0.1, server: , request: "GET / HTTP/1.1", upstream:
"fastcgi://127.0.0.1:8080", host: "www.example.com"
```

Это сообщение можно интерпретировать двояко. Возможно, в сервере, с которым мы взаимодействуем, некорректно реализован протокол FastCGI. Но, быть может, мы по ошибке направили трафик HTTP-серверу вместо FastCGI-сервера. В последнем случае проблему можно решить простым изменением конфигурации (включить директиву `fastcgi_pass` вместо `proxy_pass` или правильно задать адрес FastCGI-сервера).

Это сообщение может также означать, что проксируемый сервер просто слишком долго генерировал ответ. Причин может быть масса, но на стороне NGINX решение очевидно: увеличить таймаут. В зависимости от того, какой модуль устанавливал соединение, нужно будет изменить подразумеваемое по умолчанию значение 60s в директиве `proxy_read_timeout` или `fastcgi_read_timeout` (или еще в какой-то директиве вида `*_read_timeout`).

```
2012/11/29 06:31:42 [error] 2589#0: *6437 client intended to send
too large body: 13106010 bytes, client: 127.0.0.1, server: ,
request: "POST /upload_file.php HTTP/1.1", host: "www.example.com",
referrer: "http://www.example.com/file_upload.html"
```

Здесь все просто, NGINX сообщает, что не смогла закачать слишком большой файл. Чтобы решить проблему, нужно увеличить значение в директиве `client_body_size`. Имейте в виду, что из-за кодирования объем закачиваемых данных может процентов на 30 превышать размер самого файла (поэтому, если вы собираетесь разрешить пользователям закачивать файлы размером до 12 МБ, задавайте значение `16m`).

```
2012/10/14 19:51:22 [emerg] 3969#0: "proxy_pass" cannot have URI part in
location given by regular expression, or inside named location, or inside "if"
statement, or inside "limit_except" block in /opt/nginx/conf/nginx.conf:16
```

Здесь мы видим, что NGINX не запустилась из-за ошибки в конфигурационном файле. В чем дело, описано очень подробно. Проблема в том, что в директиве `proxy_pass` присутствует URI, которого в этом месте быть не должно. NGINX даже сообщает, в какой строке (`16`) какого файла (`/opt/nginx/conf/nginx.conf`) произошла ошибка.

```
2012/10/14 18:46:26 [emerg] 2584#0: mkdir() "/home/www/tmp/proxy_temp"
failed (2: No such file or directory)
```

В этом примере NGINX не запустилась, потому что не смогла сделать то, о чем ее просили. В директиве `proxy_temp_path` указано, где сохранять временные файлы при проксировании. Если NGINX не сможет создать такой каталог, то не запустится, поэтому лучше создайте требуемый каталог заранее.

```
2012/10/14 18:46:54 [emerg] 2593#0: unknown directive
"client_body_temp_ path" in /opt/nginx/conf/nginx.conf:6
```

А вот это сообщение озадачивает. Мы знаем, что директива `client_body_temp_path` существует, но NGINX почему-то не распознает ее. Но поразмыслив, как NGINX обрабатывает конфигурационный файл, мы понимаем, что сообщение имеет смысл. NGINX построена но модульному принципу. Каждый модуль отвечает за обработку своего конфигурационного контекста. Отсюда можно сделать вывод, что эта директива оказалась вне контекста модуля, который ее понимает.

```
2012/10/16 20:56:31 [emerg] 3039#0: "try_files" directive is not
allowed here in /opt/nginx/conf/nginx.conf:16
```

Иногда NGINX подсказывает, что не так. В примере выше NGINX распознала директиву `try_files`, по говорит, что она находится не там, где надо. И при этом любезно сообщает, в каком месте конфигурационного файла произошла ошибка, так что найти ее не составит труда.

```
2012/10/16 20:56:42 [emerg] 3043#0: host not found in upstream
"tickets. example.com" in /opt/nginx/conf/nginx.conf:22
```

Это чрезвычайное сообщение показывает, насколько сильно NGINX зависит от DNS, если в конфигурационном файле употребляются доменные имена. Если NGINX не может разрешить имена, встречающиеся в директивах `upstream`, `proxy_pass`, `fastcgi_pass` или иных директивах вида `*_pass`, то она не запускается. Поэтому так важен порядок инициализации различных подсистем при загрузке операционной системы. Позаботьтесь о том, чтобы к моменту запуска NGINX подсистема разрешения доменных имен уже была запущена.

```
2012/10/29 18:59:26 [emerg] 2287#0: unexpected "}" in
/opt/nginx/conf/nginx.conf:40
```

Это сообщение означает, что NGINX не смогла закрыть конфигурационный контекст. Что-то предшествующее указанной строке помешало NGINX сформировать полный контекст, заключенный в фигурные скобки. Обычно так бывает, когда предыдущая строка не заканчивается точкой с запятой, поэтому NGINX считает прочитанный символ } частью незавершенной строки.

```
2012/10/28 21:38:34 [emerg] 2318#0: unexpected end of file,
expecting "}" in /opt/nginx/conf/nginx.conf:21
```

Это сообщение тесно связано с предыдущим и означает, что NGINX дошла до конца конфигурационного файла, но не нашла парную закрывающую скобку. Такая ошибка случается, когда скобки { и } не сбалансированы. Текстовый редактор, отслеживающий соответственные скобки, поможет понять, какая отсутствует. Внимательно следите, куда вставляете отсутствующую скобку; если допустить ошибку, то можно получить совершенно не то, что ожидалось.

```
2012/10/29 18:50:11 [emerg] 2116#0: unknown "exclusion" variable
```

Это пример использования необъявленной переменной; сообщение означает, что переменная `$exclusion` встретилась в конфигурационном файле до директивы `set`, `map` или `geo`, в которой определено ее значение. Возможно, все объясняется простой опечаткой, например, была определена переменная `$exclusions`, но по ошибке в ссылке фигурирует имя `$exclusion`.

```
2012/11/29 21:26:51 [error] 3446#0: *2849
SSL3_GET_FINISHED:digest check failed
```

Это означает, что нужно запретить повторное использование сеансов SSL. Сделать это можно, задав в директиве `proxy_ssl_session_reuse` значение `off`.

Настройка расширенного протоколирования

При нормальных обстоятельствах мы хотим, чтобы журналы были как можно меньше. Обычно нас интересует, к каким URI-адресам обращались клиенты, когда это происходило и - в случае ошибки - сообщение о ней. Если же требуется больше информации, то придется включить отладочное протоколирование.

Отладочное протоколирование

Для активации отладочного протоколирования на этапе конфигурирования следует задать флаг `--with-debug--with-debug`. Поскольку делать это в производственной системе, которая должна работать максимально быстро, не рекомендуется, то нужно подготовить два варианта двоичного файла `nginx`: один для производственной системы и другой - откомпилированный точно с такими же параметрами и флагом `--with-debug` - который можно будет подставить вместо первого, когда возникает необходимость в отладке.

Переключение двоичного файла во время выполнения

NGINX может переключать двоичный файл во время выполнения. Заменив один файл `nginx` другим - то ли для перехода на новую версию, то ли вследствие желания изменить состав включенных модулей - мы можем приступить к процедуре переключения.

1. Послать работающему главному процессу NGINX сигнал `USR2`, по которому тот запускает новый главный процесс. В результате PID-файл процесса будет переименован с добавлением суффикса `.oldbin` (например, `/var/run/nginx. pid.oldbin`):
```
# kill -USR2 `cat /var/run/nginx.pid`
```
   В этот момент у нас есть два главных процесса NGINX, каждый со своим набором рабочих процессов для обработки поступающих запросов:
```
root 1149 0.0 0.2 20900 11768 ?? Is Fri03PM 0:00.13 nginx:
master process /usr/local/sbin/nginx
www 36660 0.0 0.2 20900 11992 ?? S 12:52PM 0:00.19 nginx:
worker process (nginx)
www 36661 0.0 0.2 20900 11992 ?? S 12:52PM 0:00.19 nginx:
worker process (nginx)
www 36662 0.0 0.2 20900 12032 ?? I 12:52PM 0:00.01 nginx:
worker process (nginx)
www 36663 0.0 0.2 20900 11992 ?? S 12:52PM 0:00.18 nginx:
worker process (nginx)
root 50725 0.0 0.1 18844 8408 ?? I 3:49PM 0:00.05 nginx:
master process /usr/local/sbin/nginx
www 50726 0.0 0.1 18844 9240 ?? I 3:49PM 0:00.00 nginx:
worker process (nginx)
www 50727 0.0 0.1 18844 9240 ?? S 3:49PM 0:00.01 nginx:
worker process (nginx)
www 50728 0.0 0.1 18844 9240 ?? S 3:49PM 0:00.01 nginx:
worker process (nginx)
www 50729 0.0 0.1 18844 9240 ?? S 3:49PM 0:00.01 nginx:
worker process (nginx)
```
2. Послать старому главному процессу NGINX сигнал `WINCH`, по которому тот прекращает прием новых запросов и уничтожает свои рабочие процессы по мере того, как они заканчивают обработку начатых запросов.
```
# kill -WINCH `cat /var/run/nginx.pid.oldbin`
```
   В ответ мы получаем такие сообщения:
```
root 1149 0.0 0.2 20900 11768 ?? Ss Fri03PM 0:00.14 nginx:
master process /usr/local/sbin/nginx
root 50725 0.0 0.1 18844 8408 ?? I 3:49PM 0:00.05 nginx:
master process /usr/local/sbin/nginx
www 50726 0.0 0.1 18844 9240 ?? I 3:49PM 0:00.00 nginx:
worker process (nginx)
www 50727 0.0 0.1 18844 9240 ?? S 3:49PM 0:00.01 nginx:
worker process (nginx)
```

```
www 50728 0.0 0.1 18844 9240 ?? S 3:49PM 0:00.01 nginx:
worker process (nginx)
www 50729 0.0 0.1 18844 9240 ?? S 3:49PM 0:00.01 nginx:
worker process (nginx)
```

3. Послать старому главному процессу NGINX сигнал `QUIT`, после того как все его рабочие процессы завершились. После этого остается только новый `nginx`, который обрабатывает все запросы:
   ```
   # kill -QUIT `cat /var/run/nginx.pid.oldbin`
   ```

Если в новом двоичном файле обнаружится какая-то ошибка, то мы можем вернуться к старому, если еще не успели послать ему сигнал `QUIT`:

```
# kill -HUP `cat /var/run/nginx.pid.oldbin`
# kill -QUIT `cat /var/run/nginx.pid`
```

Если новый главный процесс все еще работает, ему можно послать сигнал тени для принудительного завершения:

```
# kill -TERM `cat /var/run/nginx.pid`
```

Аналогично новые рабочие процессы, продолжающие работать, могут быть остановлены сигналом `KILL`.

Некоторые операционные системы автоматически выполняют процедуру замены исполняемого файла после модернизации пакета `nginx`.

Запустив `nginx` с поддержкой отладки, мы можем настроить отладочное протоколирование:

```
user www;
events {
  worker_connections 1024;
}
error_log logs/debug.log debug;
http {
…
}
```

Директиву `error_log` мы поместили в главный контекст конфигурации NGINX, чтобы ее действие распространялось на все дочерние

контексты, где эта директива не переопределена. Можно включить несколько директив `error_log` с разными уровнями протоколирования, указывающие на разные файлы. Помимо `debug`, в директиве `error_log` могут быть заданы следующие параметры:

- `debug_core`
- `debug_alloc`
- `debug_mutex`
- `debug_event`
- `debug_http`
- `debug_imap`

Каждый из них предназначен для отладки одного модуля NGINX.

Имеет также смысл сконфигурировать отдельный журнал ошибок для каждого виртуального сервера, тогда в него будут попадать ошибки, относящиеся только к этому серверу. Эту идею можно обобщить на модули `core` и `http`:

```
error_log logs/core_error.log;

events {
  worker_connections 1024;
}
http {

  error_log logs/http_error.log;
    server {
    server_name www.example.com;
    error_log logs/www.example.com_error.log;
  }
  server {
    server_name www.example.org;
    error_log logs/www.example.org_error.log;
   }
}
```

Следуя этому образцу, мы можем отлаживать конкретный виртуальный сервер, если задача состоит именно в этом.

```
server {
  server_name www.example.org;
  error_log logs/www.example.org_debug.log debug_http;
}
```

Ниже приведен пример отладочных сообщений для одного запроса, когда задан уровень `debug_http`. Распечатка перемежается комментариями.

```
<timestamp> [debug] <worker pid>#0: *<connection number>
http cl:-1 max:1048576
```

На ранней стадии обработки запроса активируется модуль `rewrite`:

```
<timestamp> [debug] <worker pid>#0: *<connection number> rewrite phase: 3
<timestamp> [debug] <worker pid>#0: *<connection number> post rewrite phase: 4
<timestamp> [debug] <worker pid>#0: *<connection number> generic phase: 5
<timestamp> [debug] <worker pid>#0: *<connection number> generic phase: 6
<timestamp> [debug] <worker pid>#0: *<connection number> generic phase: 7
```

Проверяются ограничения доступа:

```
<timestamp> [debug] <worker pid>#0: *<connection number> access phase: 8
<timestamp> [debug] <worker pid>#0: *<connection number> access:
0100007F FFFFFFFF 0100007F
```

Далее разбирается директива `try_files`. Путем конкатенации строки (`http script copy`) и переменной (`http script var`), указанных в параметрах этой директивы, конструируется относительный путь к файлу:

```
<timestamp> [debug] <worker pid>#0: *<connection number> try files phase: 11
<timestamp> [debug] <worker pid>#0: *<connection number> http script copy: "/"
<timestamp> [debug] <worker pid>#0: *<connection number> http script var:
"ImageFile.jpg"
```

Результат вычисления конкатенируется с путем, заданным в директиве `alias` или `root` для данного местоположения, в результате чего получается полный путь:

```
<timestamp> [debug] <worker pid>#0: *<connection number> trying
to use file: "/ImageFile.jpg" "/data/images/ImageFile.jpg"
<timestamp> [debug] <worker pid>#0: *<connection number> try
file uri: "/ImageFile.jpg"
```

Обрабатывается содержимое найденного файла:

```
<timestamp> [debug] <worker pid>#0: *<connection number> content phase: 12
<timestamp> [debug] <worker pid>#0: *<connection number> content phase: 13
<timestamp> [debug] <worker pid>#0: *<connection number> content phase: 14
<timestamp> [debug] <worker pid>#0: *<connection number> content phase: 15
<timestamp> [debug] <worker pid>#0: *<connection number> content phase: 16
```

`http filename` ниже - это полный путь к отправляемому файлу:

```
<timestamp> [debug] <worker pid>#0: *<connection number> http filename:
"/data/images/ImageFile.jpg"
```

Модуль `static` принимает дескриптор этого файла:

```
<timestamp> [debug] <worker pid>#0: *<connection number> http static fd: 15
```

Временно сохраненное тело ответа больше не нужно:

```
<timestamp> [debug] <worker pid>#0: *<connection number> http set discard body
```

Имея всю информацию о файле, NGINX может сконструировать заголовки ответа:

```
<timestamp> [debug] <worker pid>#0: *<connection number> HTTP/1.1 200 OK
Server: nginx/<version>
Date: <Date header>
Content-Type: <MIME type>
Content-Length: <filesize>
Last-Modified: <Last-Modified header>
Connection: keep-alive
Accept-Ranges: bytes
```

На следующей фазе применяются активные фильтры, которые могут производить различные преобразования:

```
<timestamp> [debug] <worker pid>#0: *<connection number>
http write filter: l:0 f:0 s:219
<timestamp> [debug] <worker pid>#0: *<connection number>
http output filter "/ImageFile.jpg?file=ImageFile.jpg"
<timestamp> [debug] <worker pid>#0: *<connection number>
http copy filter: "/ImageFile.jpg?file=ImageFile.jpg"
<timestamp> [debug] <worker pid>#0: *<connection number>
http postpone filter "/ImageFile.jpg?file=ImageFile.jpg" 0007FFF30383040
<timestamp> [debug] <worker pid>#0: *<connection number>
http write filter: l:1 f:0 s:480317
<timestamp> [debug] <worker pid>#0: *<connection number>
http write filter limit 0
<timestamp> [debug] <worker pid>#0: *<connection number>
http write filter 0000000001911050
<timestamp> [debug] <worker pid>#0: *<connection number>
http copy filter: -2 "/ImageFile.jpg?file=ImageFile.jpg"
<timestamp> [debug] <worker pid>#0: *<connection number>
http finalize request: -2, "/ImageFile.jpg?file=ImageFile.jpg" a:1, c:1
<timestamp> [debug] <worker pid>#0: *<connection number>
http run request: "/ImageFile.jpg?file=ImageFile.jpg"
```

```
<timestamp> [debug] <worker pid>#0: *<connection number>
http writer handler: "/ImageFile.jpg?file=ImageFile.jpg"
<timestamp> [debug] <worker pid>#0: *<connection number>
http output filter "/ImageFile.jpg?file=ImageFile.jpg"
<timestamp> [debug] <worker pid>#0: *<connection number>
http copy filter: "/ImageFile.jpg?file=ImageFile.jpg"
<timestamp> [debug] <worker pid>#0: *<connection number>
http postpone filter "/ImageFile.jpg?file=ImageFile.jpg" 0000000000000000
<timestamp> [debug] <worker pid>#0: *<connection number>
http write filter: l:1 f:0 s:234338
<timestamp> [debug] <worker pid>#0: *<connection number>
http write filter limit 0
<timestamp> [debug] <worker pid>#0: *<connection number>
http write filter 0000000000000000
<timestamp> [debug] <worker pid>#0: *<connection number>
http copy filter: 0 "/ImageFile.jpg?file=ImageFile.jpg"
<timestamp> [debug] <worker pid>#0: *<connection number>
http writer output filter: 0, "/ImageFile.jpg?file=ImageFile.jpg"
<timestamp> [debug] <worker pid>#0: *<connection number>
http writer done: "/ImageFile.jpg?file=ImageFile.jpg"
```

После того как все фильтры отработали, обработка запроса завершается:

```
<timestamp> [debug] <worker pid>#0: *<connection number> http
finalize request: 0, "/ImageFile.jpg?file=ImageFile.jpg" a:1, c:1
```

Обработчик `keepalive` смотрит, оставить ли соединение открытым:

```
<timestamp> [debug] <worker pid>#0: *<connection number> set
http keepalive handler
<timestamp> [debug] <worker pid>#0: *<connection number> http close request
```

Запрос обработан, можно поместить запись о нем в журнал:

```
<timestamp> [debug] <worker pid>#0: *<connection number> http log handler
<timestamp> [debug] <worker pid>#0: *<connection number> hc free:
0000000000000000 0
<timestamp> [debug] <worker pid>#0: *<connection number> hc busy:
0000000000000000 0
<timestamp> [debug] <worker pid>#0: *<connection number> tcp_nodelay
```

Клиент закрыл соединение, и NGINX - тоже:

```
<timestamp> [debug] <worker pid>#0: *<connection number>
http keepalive handler
<timestamp> [info] <worker pid>#0: *<connection number>
client <IP address> closed keepalive connection
<timestamp> [debug] <worker pid>#0: *<connection number>
close http connection: 3
```

Как видите, информации немало. Если вы никак не можете понять, почему некоторая конфигурация не работает, изучение отладочного журнала может оказаться очень кстати. Из него видно, в каком порядке работают фильтры и какие обработчики вызывались для обслуживания запроса.

Использование журналов доступа для отладки

Когда я только учился программировать и не мог найти источник ошибки, мой друг советовал «понаставить printf'ы». Именно так ему удавалось быстрее всего разобраться в проблеме. А смысл совета заключался в том, чтобы печатать сообщения в каждой точке ветвления программы, тогда было бы видно, по какому пути шло выполнение и где логика дала сбой.

Ту же идею можно применить и к настройке NGINX. Только вместо `printf()` мы используем директивы `log_format` и `access_log` для визуализации и анализа порядка обработки запроса. Директива `log_format` позволяет увидеть значения переменных в различных точках конфигурации:

```
http {
  log_format sentlog '[$time_local] "$request" $status $body_bytes_sent ';
  log_format imagelog '[$time_local] $image_file $image_type '
    '$body_bytes_sent $status';
  log_format authlog '[$time_local] $remote_addr $remote_user '
    '"$request" $status';
}
```

Используя несколько директив `access_log`, мы можем посмотреть, какие местоположения вызываются в различные моменты времени. Настроив свой журнал доступа для каждого местоположения, легко понять, какие не используются вовсе. Любое изменение в таком местоположении не повлияет на обработку запроса. Первыми следует изучить местоположения, расположенные выше в иерархии.

```
http {
  log_format sentlog '[$time_local] "$request" $status $body_bytes_sent ';
  log_format imagelog '[$time_local] $image_file $image_type '
    '$body_bytes_sent $status';
  log_format authlog '[$time_local] $remote_addr $remote_user '
    '"$request" $status';

  server {
```

```
    server_name .example.com;
    root /home/www;

    location / {
      access_log logs/example.com-access.log combined;
      access_log logs/example.com-root_access.log sentlog;

      rewrite ^/(.*)\.(png|jpg|gif)$ /images/$1.$2;
      set $image_file $1;
      set $image_type $2;
    }

    location /images {
      access_log logs/example.com-images_access.log imagelog;
    }

    location /auth {
      auth_basic "authorized area";
      auth_basic_user_file conf/htpasswd;
      deny all;

      access_log logs/example.com-auth_access.log authlog;
    }
  }
}
```

В этом примере в каждое местоположение включена директива `access_log`, и для каждого журнала доступа задан свой формат в директиве `log_format`. Заглянув в журнал доступа, мы сможем определить, какие запросы в какие местоположения попадали. Если в файле `example.com-images_access. log` не оказалось ни одной записи, значит, ни один запрос не достиг местоположения `/images`. Сравнив содержимое различных файлов, нетрудно понять, правильные ли значения присвоены переменным. Например, если переменные `$image_file` и `$image_ type` пусты, то соответствующие им в формате `imagelog` места окажутся незаполненными в журнале доступа.

Типичные ошибки конфигурирования

Следующей шаг при поиске неполадок - разобраться, достигает ли конфигурация поставленных перед ней целей. Интернет вот уже много лет пестрит конфигурациями для NGINX. Зачастую они были написаны для прежних версий и решали какую-то конкретную задачу. Увы, эти конфигурации копируют бездумно, не понимая,

для чего они предназначены. Иногда существует более правильный способ решить ту же задачу, воспользовавшись более современными средствами.

Использование if вместо try_files

Рассмотрим, к примеру, случай, когда требуется доставить пользователю статический файл, если он есть в файловой системе, а в противном случае передать запрос FastCGI-серверу:

```
server {
  root /var/www/html;

  location / {
    if (!-f $request_filename) {
      include fastcgi_params;
      fastcgi_pass 127.0.0.1:9000;
      break;
    }
  }
}
```

Так обычно решалась эта задача до появления директивы `try_files` в версии NGINX 0.7.27. Сейчас такая конфигурация считается ошибкой, потому что `if` используется внутри `location`. В разделе «Преобразование конфигурации с 'if' в более современную форму» из главы А было объяснено, почему это может иметь неожиданные последствия и даже стать причиной аварийного завершения программы. Правильное решение выглядит так:

```
server {
  root /var/www/html;

  location / {
    try_files $uri $uri/ @fastcgi;
  }
  location @fastcgi {
    include fastcgi_params;
    fastcgi_pass 127.0.0.1:9000;
  }
}
```

Директива `try_files` смотрит, существует ли файл, и, если нет, передает запрос FastCGI-серверу. Никакие `if` при этом не нужны.

Использование if для ветвления по имени хоста

Не счесть примеров конфигураций, в которых `if` используется для переадресации запросов на основе HTTP-заголовка `Host`. В этом случае конфигурация работает как селектор, вычисляемый для каждого запроса:

```
server {
  server_name .example.com;
  root /var/www/html;
  if ($host ~* ^example\.com) {
    rewrite ^/(.*)$ http://www.example.com/$1 redirect;
  }
}
```

Вместо того чтобы тратить ресурсы на вычисление `if` для каждого запроса, не проще ли воспользоваться обычной процедурой NGINX маршрутизации запросов к подходящему виртуальному серверу? Тогда переадресацию можно поместить туда, где ей место и притом без всякого `rewrite`:

```
server {
  server_name example.com;
  return 301 $scheme://www.example.com;
}

server {
  server_name www.example.com;
  root /var/www/html;
  location / {
  …
  }
}
```

Неоптимальное использование контекста server

Еще одно место, где некритичное копирование фрагментов часто приводит к некорректной конфигурации, - это контекст `server`. В нем описывается виртуальный сервер (все, что адресуется с помощью имени, указанного в директиве `server_name`). Но эта идея в копируемых фрагментах не находит должного отражения.

Часто приходится видеть директивы `root` и `index` в каждой секции `location`:

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

  location / {
    root /var/www/html;
    index index.php index.html index.htm;
  }
  location /ftp{
     root /var/www/html;
     index index.php index.html index.htm;
   }
 }
```

Это может стать причиной ошибок при добавлении нового местоположения, если забыть скопировать туда эти директивы или скопировать их неправильно. Смысл директив `root` и `index` заключается соответственно в том, чтобы указать корень документов для виртуального сервера и файлы, которые следует поискать, когда в URI-адресе указан только каталог. Любая секция `location` наследует эти значения от контекста `server`.

```
server {
  server_name www.example.com;
  root /var/www/html;
  index index.php index.html index.htm;

  location / {

  ...

  }

  location /ftp{
  ...
  }
}
```

Здесь мы указали, что все файлы находятся в каталоге `/var/www/html` и его подкаталогах, и в любом местоположении следует искать файлы `index.php`, `index.html` и `index.htm` именно в таком порядке.

Ограничения операционной системы

Обычно операционная система - последнее место, которое мы подозреваем, сталкиваясь с ошибкой. Мы предполагаем, что человек, который устанавливал систему, оптимизировал ее под нашу рабочую нагрузку и протестировал в похожих условиях. Часто это не так. Бывает, что приходится копаться в настройках самой операционной системы, чтобы найти узкое место.

Что касается NGINX, то обращать внимание нужно, прежде всего, на два потенциальных источника проблем: **ограничение на количество файловых дескрипторов** и **сетевые лимиты**.

Ограничение на количество файловых дескрипторов

В NGINX файловые дескрипторы применяются несколькими способами. Прежде всего, дескриптор необходим для каждого соединения с клиентом. Для каждого исходящего соединения (а они обязательно присутствуют, когда NGINX играет роль прокси-сервера) требуется уникальная пара (IP-адрес, номер TCP-порта), которая также представляется файловым дескриптором. Дескриптор нужен и тогда, когда NGINX возвращает статический файл или ответ из своего кэша. Как видите, количество потребных дескрипторов может быстро возрастать с увеличением количества одновременных пользователей. Однако общее число дескрипторов, доступных NGINX, ограничено операционной системой.

В типичной UNIX-системе к суперпользователю (с именем `root`) и к обычному пользователю применяются различные лимиты, поэтому приведенную ниже команду следует запускать от имени того непривилегированного пользователя, под которым работает NGINX (заданного либо с помощью параметра `--user` на этапе конфигурирования, либо в директиве `user` в конфигурационном файле).

```
ulimit -n
```

Эта команда показывает, сколько файловых дескрипторов доступно данному пользователю. Обычно их 1024 или того меньше. Поскольку мы знаем, что NGINX будет основным потребителем дескрипторов на данной машине, то можем существенно увеличить этот предел. Насколько именно, зависит от операционной системы. Делается это следующим образом:

- Linux

```
vi /etc/security/limits.conf

www-run hard nofile 65535
$ ulimit -n 65535
```

- FreeBSD

```
vi /etc/sysctl.conf

kern.maxfiles=65535
kern.maxfilesperproc=65535
kern.maxvnodes=65535
# /etc/rc.d/sysctl reload
```

- Solaris

```
# projadd -c "increased file descriptors" -K "process.max-
filedescriptor=(basic,65535,deny)" resource.file

# usermod -K project=resource.file www
```

Показанные команды увеличивают максимальное число файловых дескрипторов, доступных новому процессу, запущенному от имени пользователя `www`. Указанное значение будет действовать и после перезагрузки ОС.

Следующие две команды увеличивают число файловых дескрипторов, доступных работающему процессу NGINX:

```
# prctl -r -t privileged -n process.max-file-descriptor -v
65535 -i process `pgrep nginx`
# prctl -x -t basic -n process.max-file-descriptor -i
process `pgrep nginx`
```

Все вышеописанные методы изменяют лимиты операционной системы, но NGINX об этом ничего не узнает, пока мы не зададим количество файловых дескрипторов в директиве `worker_rlimit_nofile`:

```
worker_rlimit_nofile 65535;
worker_processes 8;

events {
  worker_connections 8192;
}
```

После этого нужно послать главному процессу `nginx` сигнал `HUP`:

```
# kill -HUP `cat /var/run/nginx.pid`
```

Теперь NGINX сможет обслуживать свыше 65 000 одновременных клиентов, соединений с проксируемыми серверами и локальных статических или кэшированных файлов. Столько рабочих процессов (директива `worker_processes`) имеет смысл запускать, только если машина действительно оснащена восемью процессорными ядрами или рабочая нагрузка подразумевает большой объем ввода-вывода. В противном случае уменьшите значение `worker_processes`, сделав его равным количеству ядер, и увеличьте `worker_connections`, так чтобы произведение обоих чисел было приблизительно равно 65 000.

Разумеется, можно довести общее количество файловых дескрипторов и `worker_connections` до величины, имеющей смысл для конкретного состава оборудования и решаемых задач. NGINX в состоянии обслужить миллионы одновременных соединений при условии правильного задания лимитов операционной системы и конфигурационных параметров.

Сетевые лимиты

Очутившись в ситуации нехватки сетевых буферов, вы, скорее всего, сможете только зайти с консоли - и то, если повезет. Это бывает, когда к NGINX поступает так много клиентских соединений, что все сетевые буферы оказываются заняты. Процедура увеличения количества сетевых буферов также зависит от операционной системы:

- FreeBSD

```
vi /boot/loader.conf

kern.ipc.nmbclusters=262144
```

- Solaris

```
# ndd -set /dev/tcp tcp_max_buf 16777216
```

Если NGINX работает как прокси-сервер - почтовый или HTTP, - то ей необходимо открывать много соединений с проксируемыми серверами. Чтобы увеличить число соединений, необходимо до максимума расширить диапазон эфемерных TCP-портов.

- Linux

```
vi /etc/sysctl.conf

net.ipv4.ip_local_port_range = 1024 65535
# sysctl -p /etc/sysctl.conf
```

- FreeBSD

```
vi /etc/sysctl.conf
net.inet.ip.portrange.first=1024
net.inet.ip.portrange.last=65535
# /etc/rc.d/sysctl reload
```

- Solaris

```
# ndd -set /dev/tcp tcp_smallest_anon_port 1024
# ndd -set /dev/tcp tcp_largest_anon_port 65535
```

Разобравшись с основными параметрами, можно переходить в настройке более специфичных параметров, относящихся к производительности. Они описаны в следующем разделе.

Проблемы с производительностью

Проектируя приложение и настраивая для него NGINX, мы естественно рассчитываем на высокую производительность. Если же в этом плане возникают какие-то проблемы, то необходимо выяснить, в чем причина. Возможно, дело в самом приложении. А, возможно, в конфигурации NGINX. Мы должны научиться определять источник проблемы.

Работая в качестве прокси-сервера, NGINX задействует главным образом сеть. Если на сетевом уровне имеются какие-то ограничения, то NGINX не сможет работать оптимально. Процедура настройки сети зависит от операционной системы и от самой сети, в которой эксплуатируется NGINX, поэтому следует проводить ее с учетом конкретных условий.

Один из важных параметров, относящихся к производительности сети, - размер очереди TCP-соединений системного вызова `listen`. Чем больше это значение, тем больше можно обслужить клиентов. Каким должно быть оптимальное значение и как его задать, зависит от операционной системы.

- Linux

```
vi /etc/sysctl.conf

net.core.somaxconn = 3240000
# sysctl -p /etc/sysctl.conf
```

- FreeBSD

```
vi /etc/sysctl.conf

kern.ipc.somaxconn=4096
# /etc/rc.d/sysctl reload
```

- Solaris

```
# ndd -set /dev/tcp tcp_conn_req_max_q 1024
# ndd -set /dev/tcp tcp_conn_req_max_q0 4096
```

Следующий интересный параметр - размеры буферов приема и отправки. Приведенные ниже значения показаны только для примера - они могут стать причиной повышенного потребления памяти, так что применяйтесь к своей ситуации и тестируйте.

- Linux

```
vi /etc/sysctl.conf

net.ipv4.tcp_wmem = 8192 87380 1048576
net.ipv4.tcp_rmem = 8192 87380 1048576
# sysctl -p /etc/sysctl.conf
```

- FreeBSD

```
vi /etc/sysctl.conf

net.inet.tcp.sendspace=1048576
net.inet.tcp.recvspace=1048576
# /etc/rc.d/sysctl reload
```

- Solaris

```
# ndd -set /dev/tcp tcp_xmit_hiwat 1048576
# ndd -set /dev/tcp tcp_recv_hiwat 1048576
```

Можно также изменить размеры буферов непосредственно в конфигурационном файле NGINX, тогда они будут действовать только NGINX. Это желательно, когда на машине работает несколько служб, но требуется, чтобы именно NGINX могла получить максимум отдачи от имеющихся сетевых возможностей:

```
server {
  listen 80 sndbuf=1m rcvbuf=1m;
}
```

Настройка сети может оказать существенное влияние на производительность. Однако рекомендуется изменять по одному параметру за раз и замерять результаты после изменения. Оптимизировать производительность можно на стольких разных уровнях, что краткое введение в этой главе не дает полного представления о предмете. Для интересующихся этой тематикой есть немало книг и сетевых ресурсов.

Как сделать измененные сетевые параметры постоянными в Solaris

Выше мы изменяли некоторые параметры TCP в командной строке. В случае Linux и FreeBSD эти изменения сохраняются и после перезагрузки ОС, потому что записываются в конфигурационный файл (например, `/etc/sysctl.conf`). Но в Solaris ситуация иная. Эти изменения никуда не записываются и потому не сохраняются при перезагрузке.

Начиная с версии Solaris 10, предоставляется уникальный каркас управления службами (**Service Management Framework - SMF**), позволяющий к тому же задавать порядок их запуска при загрузке ОС. (Это, конечно, чрезмерно упрощенное описание, каркас SMF умеет гораздо больше.) Чтобы настройки TCP сохранялись, необходимо написать манифест SMF и соответствующий скрипт.

Подробности см. в приложении D «Сохранение сетевых настроек в Solaris».

Использование модуля Stub Status

И состав NGINX входит модуль самодиагностики, который выводит статистические данные о работе программы. Этот модуль называется **Stub Status** и активируется заданием параметра `--with-http_stub_status_module` на этапе конфигурирования.

Чтобы увидеть статистику, порождаемую этим модулем, необходимо включить директиву `stub_status`, задав в ней значение `on`. Для этого модуля следует создать отдельную секцию `location`, чтобы можно было применить список ACL:

```
location /nginx_status {
  stub_status on;
  access_log off;
  allow 127.0.0.1;
  deny all;
}
```

Обращение к этому URI с компьютера `localhost` (например, `curl http://localhost/nginx_status`) возвращает ответ такого вида:

```
Active connections: 2532
server accepts handled requests
  1476737983 1476737983 3553635810
Reading: 93 Writing: 13 Waiting: 2426
```

Мы видим, что открыто 2532 соединения, и в настоящий момент для 93 соединений NGINX читает заголовки запроса, а 13 находятся в состоянии, когда NGINX либо читает тело запроса, либо обрабатывает запрос, либо отправляет ответ клиенту. Остальные 2426 соединений имеют тип `keepalive` и находятся в состоянии ожидания. С момента запуска процесс nginx принял и обработал 1 476 737 983 соединения, то есть ни одно не было закрыто сразу после приема. Всего было обработано 3 553 635 810 запросов, то есть в среднем по 2,4 запроса в одном соединении.

Эти данные можно передать в какую-нибудь программу обработки метрик и представить в графическом виде. Существуют модули, подключаемые к системам Munin, Nagios, collectd и другим, которые для сбора статистики обращаются к модулю `stub_status`. Со временем, имея достаточно данных, вы сможете обнаружить какие-то тенденции и коррелировать их с теми или иными факторами. На графиках должны быть видны всплески количества запросов, а также изменения в настройках операционной системы.

Резюме

Когда вводится в эксплуатацию новая программа, проблемы возникают на самых разных уровнях. Некоторые ошибки можно «отловить» и устранить в тестовой среде, другие проявляются только в условиях реальной нагрузки. Для поиска причин ошибок NGINX предлагает очень подробные журналы с различными уровнями протоколирования. Некоторые сообщения допускают несколько интерпретаций, но общий принцип понятен. Экспериментируя с различными конфигурациями и наблюдая за появляющимися сообщениями, можно научиться интерпретировать записи в журнале. На работу NGINX оказывает влияние операционная система, которая накладывает определенные ограничения, когда параметры заданы по умолчанию в расчете на наличие многих пользователей. Для оптимизации параметров с учетом реальных условий полезно понимать, что происходит на уровне TCP. И в завершение краткого обзора поиска п устранения неполадок мы показали, какую информацию возвращает модуль `stub_status`. Эти данные полезны, когда требуется составить общее представление о том, как работает NGINX.

Далее идут приложения. В первом из них перечислены все конфигурационные директивы NGINX с указанием значений по умолчанию и контекста, в котором можно употреблять директиву.

Приложение А. Справочник директив

В этом приложении перечислены все конфигурационные директивы, встречавшиеся на страницах этой книги. Для полноты приведены также некоторые директивы, которые мы не рассматривали. Дополнительно для каждой директивы указывается контекст, в котором она может употребляться. Если у директивы есть значение по умолчанию, оно также указывается. Директивы описаны по состоянию в версии NGINX 1.3.9. Актуальный список можно найти по адресу http://nginx.org/en/docs/dirindex.html.

Справочник директив

Директива	Описание	Контекст/ Умолчание
`accept_mutex`	Сериализует обращения к методу `accept ()` для новых соединений со стороны рабочих процессов	Допустимый контекст: `events` По умолчанию: `on`
`accept_mutex_delay`	Сколько времени рабочий процесс может ждать возможности принимать новые соединения, если в данный момент этим занимается другой рабочий процесс	Допустимый контекст: `events` По умолчанию: `500ms`

Директива	Описание	Контекст/ Умолчание
access_log	Определяет, куда и как записывать журналы доступа. Первый параметр - путь к файлу журнала. Путь может содержать переменные. Специальное значение off отключает протоколирование. Необязательный второй параметр определяет директиву log_format, описывающую формат журнала. Если этот параметр не задан, используется предопределенный формат. Необязательный третий параметр задает размер буфера в случае, если запись в журнал буферизуется. При использовании буферизации этот размер не должен превосходить длину операции атомарной записи на диск для конкретной файловой системы. Если третий параметр равен gzip, то журнал буферизуется и на лету сжимается при условии, что двоичный файл nginx собирался с библиотекой zlib. Последний параметр flush определяет, сколько времени данные могут оставаться в буфере в памяти перед сбросом на диск	Допустимые контексты: http, server, location, if в location, limit_ except По умолчанию: logs/ access.log combined
add_after_body	Поместить результат обработки подзапроса после тела ответа	Допустимый контекст: location По умолчанию: -
add_before_body	Поместить результат обработки подзапроса перед телом ответа	Допустимый контекст: location По умолчанию: -
add_header	Добавляет заголовок в ответ с кодом состояния 200, 204, 206, 301, 302, 303, 304 или 307	Допустимые контексты: http, server, location По умолчанию: -
addition_types	Указывается список MIME-типов ответа (в дополнение к text/html), для которых производится добавление. Звездочка (*) означает все MIME-типы	Допустимые контексты: http, server, location По умолчанию: text/ html

Директива	Описание	Контекст/ Умолчание
aio	Разрешает использование асинхронного файлового ввода-вывода. Это возможно во всех современных версиях FreeBSD и дистрибутивах Linux. Во FreeBSD директиву aio можно использовать для предварительной загрузки данных для sendfile. В Linux требуется директива directio, которая автоматически отключает sendfile	Допустимые контексты: http, server, location По умолчанию: off
alias	Определяет путь в файловой системе, соответствующий имени местоположения. Если местоположение задано с помощью регулярного выражения, то значение alias должно ссылаться на запомненные подвыражения	Допустимый контекст: location По умолчанию: -
allow	Разрешает доступ с указанного IP-адреса, из указанной сети или отовсюду (all)	Допустимые контексты: http, server, location, limit _except По умолчанию: logs/access.log combined
ancient_ browser	Подстроки, при обнаружении которых в заголовке User-Agent браузер считается устаревшим. В результате в переменную $ancient_browser записывается значение, заданное в директиве ancient_browser_value	Допустимые контексты: http, server, location По умолчанию: -
ancient_ browser_ value	Значение, присваемое переменной $ancient_ browser	Допустимые контексты: http, server, location По умолчанию: 1
auth_basic	Разрешает аутентификацию по схеме HTTP Basic Authentication. Параметр задает имя области (realm). Специальное значение off означает, что значение auth_basic, заданное на родительском уровне, отменяется	Допустимые контексты: http, server, location, limit_except По умолчанию: off
auth_basic_ user_file	Определяет, где находится файл, содержащий тройки username:password:comment, который используется для аутентификации клиентов. Поле password должно быть зашифровано алгоритмом crypt. Поле comment необязательно	Допустимые контексты: http, server, location, limit_except По умолчанию: -

Директивы	Описание	Контекст/ Умолчание
auth_http	Задает сервер, который служит для аутентификации пользователя в протоколах POP3 и IMAP	Допустимые контексты: mail, server По умолчанию: -
auth_http_header	Включает дополнительный заголовок (первый параметр) с указанным значением (второй параметр)	Допустимые контексты: mail, server По умолчанию: -
auth_http_timeout	Максимальное время ожидания ответа от сервера аутентификации	Допустимые контексты: mail, server По умолчанию: 60s
autoindex	Разрешает автоматическую генерацию страницы с содержимым каталога	Допустимые контексты: http, server, location По умолчанию: off
autoindex_exact_size	Определяет, следует ли указывать размеры файлов в каталоге в байтах или округлять до килобайт, мегабайт или гигабайт	Допустимые контексты: http, server, location По умолчанию: on
autoindex_localtime	Какое время последней модификации файла указывать на странице с содержимым каталога: местное (on) или UTC (off)	Допустимые контексты: http, server, location По умолчанию: off
break	Завершает обработку директив модуля rewrite, находящихся в том же контексте	Допустимые контексты: server, location, if По умолчанию: -
charset	Добавляет указанную кодировку в заголовок ответа Content-Type. Если кодировка отличается от указанной в директиве source_charset, то производится перекодирование	Допустимые контексты: http, server, location, if в location По умолчанию: off
charset_map	Задает таблицу перекодирования из одной кодировки в другую. Коды символов записываются в шестнадцатеричном виде. В файлах conf/koi-win, conf/koi-utf и conf/win-utf хранятся соответственно таблицы перекодирования из koi8-r в windows-1251, из koi8-r в utf-8 и из windows-1251 в utf-8	Допустимый контекст: http По умолчанию: -

Директивы	Описание	Контекст/ Умолчание
charset_types	Список MIME-типов ответа (помимо text/html), для которых будет производиться перекодирование	Допустимые контексты: http, server, location По умолчанию: text/html, text/xml, text/plain, text/vnd. wap. wml, application/xjavascript, application/rss+xml
chunked_transfer_encoding	Позволяет отключить специфицированный в стандарте HTTP/1.1 механизм поблочной передачи данных (chunked transfer encoding) в ответе клиенту	Допустимые контексты: http, server, location По умолчанию: on
client_body_buffer_size	Задает размер буфера для чтения тела запроса клиента. По умолчанию для буфера выделяется две страницы памяти. Увеличение размера позволяет предотвратить запись во временный файл на диске	Допустимые контексты: http, server, location По умолчанию: 8k| 16k (в зависимости от платформы)
client_body_in_file_only	Используется для отладки или последующей обработки тела запроса клиента. Если значение равно on, то тело запроса принудительно записывается в файл. Значение clean приводит к удалению файлов после завершения обработки запроса	Допустимые контексты: http, server, location По умолчанию: off
client_body_in_single_buffer	Заставляет NGINX сохранить все тело запроса клиента в одном буфере, чтобы уменьшить количество операций копирования	Допустимые контексты: http, server, location По умолчанию: off
client_body_temp_path	Определяет путь к каталогу для сохранения файлов с телами запросов клиентов. Второй, третий и четвертый параметры (если заданы) определяют иерархию подкаталогов в виде количества знаков в имени подкаталога	Допустимые контексты: http, server, location По умолчанию: client_body_temp

Директивы	Описание	Контекст/ Умолчание
client_body_ timeout	Задает время между последовательными операциями чтения тела запроса клиента. В случае превышения клиент получает сообщение об ошибке 408 (Request Timeout)	Допустимые контексты: http, server, location По умолчанию: 60s
client_header_ buffer_size	Задает размер буфера для чтения заголовка запроса клиента, если он превышает подразумеваемую по умолчанию величину 1 КБ	Допустимые контексты: http, server По умолчанию: 1к
client_header_ timeout	Время, отведенное на чтение всего заголовка запроса. В случае превышения клиент получает сообщение об ошибке 408 (Request Timeout)	Допустимые контексты: http, server По умолчанию: 60s
client_max_body_ size	Максимальный размер тела запроса клиента. В случае превышения отправляется ответ 413 (Request Entity Too Large)	Допустимые контексты: http, server, location По умолчанию: 1m
connection_pool_ size	Позволяет производить точную настройку выделения памяти под конкретные соединения	Допустимые контексты: http, server По умолчанию: 256
create_full_put_ path	Разрешает создание промежуточных каталогов при работе с WebDAV	Допустимые контексты: http, server, location По умолчанию: off
daemon	Определяет, будет ли nginx запускаться в режиме демона	Допустимый контекст: main По умолчанию: on
dav_access	Устанавливает права доступа к создаваемым файлам и каталогам. Если задан параметр group или all, то user можно опустить	Допустимые контексты: http, server, location По умолчанию: user rw
dav_methods	Разрешает указанные HTTP- и WebDAV-методы. При использовании метода PUT сначала создается временный файл, а затем он переименовывается. Поэтому рекомендуется указывать в директиве client_body_temp_path ту файловую систему, в которой должен находиться загруженный файл. Время модификации таких файлов можно задать в заголовке Date	Допустимые контексты: http, server, location По умолчанию: off

Директивы	Описание	Контекст/ Умолчание
debug_connection	Включает отладочное протоколирование для клиентов, соответствующих значению этой директивы. Может встречаться несколько раз. Для отладки сокетов в домене UNIX следует указать unix	Допустимый контекст: events По умолчанию: -
debug_points	В случае обнаружения внутренней ошибки процесс либо создает core-файл (abort), либо останавливается (stop) с целью последующего подключения системного отладчика	Допустимые контексты: main По умолчанию: -
default_type	Определяет подразумеваемый по умолчанию MIME-тип ответа. Используется в случае, когда MIME-тип файла не удается сопоставить ни с одним из определенных в директиве types	Допустимые контексты: http, server, location По умолчанию: text/ plain
deny	Запрещает доступ с указанного IP-адреса, из указанной сети или отовсюду (all)	Допустимые контексты: http, server, location, limit_except По умолчанию: -
directio	Разрешает использовать зависящий от операционной системы флаг при чтении файлов, размер которых больше или равен указанному. Обязательна при использовании директивы aio в Linux	Допустимые контексты: http, server, location По умолчанию: off
directio_alignment	Устанавливает выравнивание для directio. Обычно подразумеваемого по умолчанию значения 512 достаточно, но при использовании XFS в Linux рекомендуется увеличить до 4K	Допустимые контексты: http, server, location По умолчанию: 512
disable_symlinks	См. таблицу «Директивы HTTP-сервера, относящиеся к путям в файловой системе» в разделе «Поиск файлов» главы 6 «NGINX как HTTP-сервер»	Допустимые контексты: http, server, location По умолчанию: off
empty_gif	Порождает прозрачный GIF размером 1x1 для данного местоположения	Допустимый контекст: location По умолчанию: -

Директивы	Описание	Контекст/ Умолчание
env	Определяет переменные окружения, которые используются в следующих случаях: • наследование во время обновления NGINX на лету; • в модуле perl; • предоставление в распоряжение рабочих процессов. Если задать только имя переменной, то будет взято значение из окружения nginx. Чтобы присвоить переменной значение, следует воспользоваться формой var=value. N.B. Переменная NGINX является внутренней, пользователь не должен ее устанавливать	Допустимый контекст: main По умолчанию: TZ
error_log	Это файл, в который записываются сообщения об ошибках. Если ни в каком другом контексте директивы error_log нет, то в этом файле будут регистрироваться вообще все ошибки. Второй параметр директивы обозначает уровень сообщений, попадающих в журнал (debug, info, notice, warn, error, crit, alert, emerg). Сообщения уровня debug выводятся, только если программа была сконфигурирована с параметром --with-debug	Допустимые контексты: main, http, server, location По умолчанию: logs/ error.log error
error_page	Определяет URL-адрес страницы, которую нужно вернуть, если код ответа попадает в диапазон ошибок. Необязательный параметр, следующий за знаком = , позволяет изменить код ответа. Если после знака равенства не указан код ответа, то он берется из URI-адреса, при этом соответствующая страница должна возвращаться каким-то проксируемым сервером	Допустимые контексты: http, server, location, if в location По умолчанию: -

Директивы	Описание	Контекст/ Умолчание
`etag`	Отключает автоматическую генерацию заголовка ответа `ETag` для статических ресурсов	Допустимые контексты: `http`, `server`, `location` По умолчанию: `on`
`events`	Определяет новый контекст, в котором задаются директивы обработки соединений	Допустимый контекст: `main` По умолчанию: -
`expires`	См. таблицу «Директивы для модификации заголовков» в разделе «Кэширование в файловой системе» главы 7 «NGINX для разработчика»	Допустимые контексты: `http`, `server`, `location` По умолчанию: `off`
`fastcgi_bind`	Определяет адрес исходящих соединений с FastCGI-сервером	Допустимые контексты: `http`, `server`, `location` По умолчанию: -
`fastcgi_buffer_size`	Размер буфера для первой части ответа от FastCGI-сервера, в которой находятся заголовки	Допустимые контексты: `http`, `server`, `location` По умолчанию: `4k|18k` (в зависимости от платформы)
`fastcgi_buffers`	Количество и размер буферов для получения ответа от FastCGI-сервера в расчете на одно соединение	Допустимые контексты: `http`, `server`, `location` По умолчанию: `4k|18k` (в зависимости от платформы)
`fastcgi_busy_buffers_size`	Суммарный размер буферов, которые могут быть заняты для отправки ответа клиенту, пока ответ от FastCGI-сервера ещё не прочитан целиком. Обычно устанавливается вдвое большим, чем размер, указанный в директиве `fastcgi buffers`	Допустимые контексты: `http`, `server`, `location` По умолчанию: `4k|18k` (в зависимости от платформы)
`fastcgi_cache`	Определяет зону разделяемой памяти, используемую для кэширования	Допустимые контексты: `http`, `server`, `location` По умолчанию: `off`
`fastcgi_cache_bypass`	Одна или несколько строковых переменных. Если хотя бы одна из них непуста и не содержит нуль, то ответ будет запрошен у FastCGI-сервера, а не взят из кэша	Допустимые контексты: `http`, `server`, `location` По умолчанию: -
`fastcgi_cache_key`	Строка, которая используется как ключ для поиска значения в кэше	Допустимые контексты: `http`, `server`, `location` По умолчанию: -

Директивы	Описание	Контекст/ Умолчание
`fastcgi_cache_lock`	Если включено, то предотвращается отправка нескольких запросов FastCGI-серверу в случае отсутствия в кэше	Допустимые контексты: `http`, `server`, `location` По умолчанию: `off`
`fastcgi_cache_lock_timeout`	Сколько времени запрос может ждать появления записи в кэше или освобождения блокировки `fastcgi_cache_lock`	Допустимые контексты: `http`, `server`, `location` По умолчанию: `5s`
`fastcgi_cache_min_uses`	Сколько запросов с данным ключом должно поступить, прежде чем ответ будет помещен в кэш	Допустимые контексты: `http`, `server`, `location` По умолчанию: `1`
`fastcgi_cache_path`	См. таблицу «Директивы модуля FastCGI для управления потоковой передачей» в разделе «Использование NGINX совместно с PHP-FPM» главы 6 «NGINX как HTTP-сервер»	Допустимые контексты: `http`, `server`, `location` По умолчанию: -
`fastcgi_cache_use_stale`	В каких случаях допустимо использовать устаревшие кэшированные данные, если при доступе к FastCGI-серверу произошла ошибка. Параметр `updating` разрешает использовать кэшированный ответ, если как раз в данный момент загружаются более свежие данные	Допустимые контексты: `http`, `server`, `location` По умолчанию: `off`
`fastcgi_cache_valid`	Сколько времени считать действительным кэшированный ответ с кодом 200, 301 или 302. Если перед параметром `time` указан необязательный код ответа, то заданное время относится только к ответу с таким кодом. Специальный параметр `any` означает, что в течение заданного времени следует кэшировать ответ с любым кодом	Допустимые контексты: `http`, `server`, `location` По умолчанию: -
`fastcgi_connect_timeout`	Максимальное время, в течение которого NGINX ожидает установления соединения при отправке запроса FastCGI-серверу	Допустимые контексты: `http`, `server`, `location` По умолчанию: `60s`
`fastcgi_hide_header`	Список заголовков, которые не следует передавать клиенту	Допустимые контексты: `http`, `server`, `location` По умолчанию: -

Директивы	Описание	Контекст/ Умолчание
`fastcgi_ignore_client_abort`	Если значение равно `on`, то NGINX не станет разрывать соединение с FastCGI-сервером в случае, когда клиент разрывает свое соединение	Допустимые контексты: `http`, `server`, `location` По умолчанию: `off`
`fastcgi_ignore_headers`	Какие заголовки можно игнорировать при обработке ответа от FastCGI-сервера	Допустимые контексты: `http`, `server`, `location` По умолчанию: -
`fastcgi_index`	Задает имя файла, дописываемое в конец строки `$fastcgi_script_name` после знака косой черты	Допустимые контексты: `http`, `server`, `location` По умолчанию: -
`fastcgi_intercept_errors`	Если значение равно `on`, то NGINX будет отображать страницу, заданную директивой `error_page`, вместо ответа, полученного от FastCGI-сервера	Допустимые контексты: `http`, `server`, `location` По умолчанию: `off`
`fastcgi_keep_conn`	Разрешает соединения типа `keepalive` с FastCGI-сервером, инструктируя его не закрывать соединение немедленно	Допустимые контексты: `http`, `server`, `location` По умолчанию: `off`
`fastcgi_max_temp_file_size`	Максимальный размер временного файла, в который записывается часть ответа в случае, когда он не умещается целиком в буферах памяти	Допустимые контексты: `http`, `server`, `location` По умолчанию: `1024m`
`fastcgi_next_upstream`	См. таблицу «Директивы модуля FastCGI для управления потоковой передачей» в разделе «Использование NGINX совместно с PHP-FPM» главы 6 «NGINX как HTTP-сервер»	Допустимые контексты: `http`, `server`, `location` По умолчанию: error timeout
`fastcgi_no_cache`	Одна или несколько строковых переменных. Если хотя бы одна из них непуста и не содержит нуль, то NGINX не будет помещать в кэш ответ, полученный от FastCGI-сервера	Допустимые контексты: `http`, `server`, `location` По умолчанию: -
`fastcgi_param`	Задает имя и значение параметра, передаваемого FastCGI-серверу. Если следует передавать только параметры с непустыми значениями, то необходимо задать дополнительный параметр `if_not_empty`	Допустимые контексты: `http`, `server`, `location` По умолчанию: -

Директивы	Описание	Контекст/ Умолчание
`fastcgi_pass`	Определяет FastCGI-сервер, которому передается запрос, в виде пары `address:port` или `unix:path` для сокета в домене UNIX	Допустимые контексты: `http`, `server`, `location` По умолчанию: -
`fastcgi_pass_header`	Отменяет сокрытие заголовков, определенных в директиве `fastcgi_hide_header`, разрешая передавать их клиенту	Допустимые контексты: `http`, `server`, `location` По умолчанию: -
`fastcgi_read_timeout`	Сколько времени может пройти между двумя последовательными операциями чтения данных от FastCGI-сервера, прежде чем соединение будет закрыто	Допустимые контексты: `http`, `server`, `location` По умолчанию: `60s`
`fastcgi_send_lowat`	Эта директива предназначена только для FreeBSD. Если значение отлично от нуля, то при взаимодействии с проксируемым сервером NGINX будет использовать либо флаг NOTE_LOWAT метода `kqueue`, либо параметр сокета `SO_SNDLOWAT` с указанным размером. В Linux, Solaris и Windows игнорируется	Допустимые контексты: `http`, `server`, `location` По умолчанию: `0`
`fastcgi_send_timeout`	Сколько времени может пройти между двумя последовательными операциями записи данных на FastCGI-сервер, прежде чем соединение будет закрыто	Допустимые контексты: `http`, `server`, `location` По умолчанию: `60s`
`fastcgi_split_path_info`	Задает регулярное выражение с двумя запоминаемыми подвыражениями. Первая запомненная строка становится значением переменной `$fastcgi_script_name`, вторая - значением переменной `$fastcgi_path_info`	Допустимые контексты: `location` По умолчанию: -

Директивы	Описание	Контекст/ Умолчание
`fastcgi_store`	Разрешает сохранение полученных от FastCGI-сервера ответов в файлах на диске. Если параметр равен `on`, то в качестве пути к каталогу с сохраняемыми файлами используется значение, заданное в директиве `alias` или `root`. Можно вместо этого задать строку, определяющую другой каталог для хранения файлов	Допустимые контексты: `http`, `server`, `location` По умолчанию: `off`
`fastcgi_store_access`	Какие права доступа следует задать для новых файлов, создаваемых в соответствии с директивой `fastcgi_store`	Допустимые контексты: `http`, `server`, `location` По умолчанию: `user:rw`
`fastcgi_temp_file_ write_size`	Ограничивает размер данных в одной операции записи во временный файл, чтобы NGINX не слишком долго блокировала исполнение программы при обработке одного запроса	Допустимые контексты: `http`, `server`, `location` По умолчанию: `8k|16k` (в зависимости от платформы)
`fastcgi_temp_path`	Каталог для хранения временных файлов, получаемых от FastCGI-сервера. Может быть многоуровневым. Второй, третий и четвертый параметры (если заданы) определяют иерархию подкаталогов в виде количества знаков в имени подкаталога	Допустимые контексты: `http`, `server`, `location` По умолчанию: `fastcgi_ temp`
`flv`	Активирует модуль `flv` для данного местоположения	Допустимый контекст: `location` По умолчанию: -

Директивы	Описание	Контекст/ Умолчание
`geo`	Определяет новый контекст, в котором переменной присваивается значение, зависящее от IP-адреса, находящегося в другой переменной. Если другая переменная не задана, берется IP-адрес из переменной `$remote_addr`. Формат определения контекст следующий: `geo [$address-variable] $variable_to-be-set { ... }` Распознаются следующие параметры: • `delete`: удалить указанную сеть; • `default`: в переменную будет записано это значение, если нет подходящего IP-адреса; • `include`: включает файл с определениями соответствий между адресом и значением переменной; • `proxy`: определяет адреса (индивидуальные или целой подсети), при запросе с которых будет использоваться IP-адрес, взятый из заголовка запроса `X-Forwarded-For`; • `proxy-recursive`: используется вместе с параметром proxy и означает, что если заголовок `X-Forwarded-For` содержит несколько значений, то следует брать последнее; • `ranges`: если присутствует, то означает, что следующие далее адреса заданы в виде диапазонов	Допустимый контекст: `http` По умолчанию: -

Директивы	Описание	Контекст/ Умолчание
`geoip_city`	Путь к базе данных GeoIP, содержащей сооответствия между IP-адресами и городами. Становятся доступны следующие переменные: • `$geoip_city_country_code`: двух-буквенный код страны; • `$geoip_city_country_code3`: трех-буквенный код страны; • `$geoip_city_country_name`: название страны; • `$geoip_region`: название региона страны; • `$geoip_city`: название города; • `$geoip_postal_code`: почтовый индекс	Допустимый контекст: `http` По умолчанию: -
`geoip_country`	Путь к базе данных GeoIP, содержащей сооответствия между IP-адресами и странами. Становятся доступны следующие переменные: • `$geoip_city_country_code`: двух-буквенный код страны; • `$geoip_city_country_code3`: трех-буквенный код страны; • `$geoip_city_country_name`: название страны	Допустимый контекст: `http` По умолчанию: -
`geoip_org`	Путь к базе данных GeoIP, содержащей сооответствия между IP-адресами и организациями. Становится доступна следующая переменная: `$geoip_org`: название организации	Допустимый контекст: `http` По умолчанию: -
`geoip_proxy`	Определяет адреса (индивидуальные или целой подсети), при запросе с которых будет использоваться IP-адрес, взятый из заголовка запроса `X-Forwaided-For`	Допустимый контекст: `http` По умолчанию: -

Директивы	Описание	Контекст/ Умолчание
geoip_proxy_ recursuve	Используется вместе с директивой geopip_proxy и означает, что если заголовок X-Forwarded-For содержит несколько значений, то следует брать последнее	Допустимый контекст: http По умолчанию: off
gunzip	Разрешает или запрещает распаковку сжатых алгоритмом gzip файлов в случае, когда клиент не поддерживает gzip	Допустимые контексты: http, server, location По умолчанию: off
gunzip_buffers	Задает количество и размер буферов для распаковки ответа	Допустимые контексты: http, server, location По умолчанию: 32 4k|16 8k (в зависимости от платформы)
gzip	Разрешает или запрещает сжатие ответов	Допустимые контексты: http, server, location, if в location По умолчанию: off
gzip_buffers	Определяет количество и размеры буферов для сжатия ответа	Допустимые контексты: http, server, location По умолчанию: 32 4k|16 8k (в зависимости от платформы)
gzip_comp_level	Уровень сжатия gzip (1-9)	Допустимые контексты: http, server, location По умолчанию: 1
gzip_disable	Регулярное выражение, описывающее те пользовательские агенты, которые не должны получать сжатое содержимое. Специальное значение msie6 является сокращением выражения MSIE [4-6]\., которое исключает MSIE 6.0 ... SV1	Допустимые контексты: http, server, location По умолчанию: -
gzip_http_version	Минимальная версия протокола HTTP в запросе, до которой вопрос о сжатии вообще не рассматривается	Допустимые контексты: http, server, location По умолчанию: 1.1
gzip_min_length	Минимальная длина ответа (определяемая заголовком Content-length), до которой вопрос о сжатии вообще не рассматривается	Допустимые контексты: http, server, location По умолчанию: 20

Директивы	Описание	Контекст/ Умолчание
gzip_proxied	См. таблицу «Директивы модуля gzip» в разделе «Сжатие» главы 5 «Обратное проксирование, дополнительные вопросы»	Допустимые контексты: http, server, location По умолчанию: off
gzip_static	Разрешает или запрещает проверку наличия предварительного сжатого файла, который можно было бы доставить клиенту, поддерживающему gzip	Допустимые контексты: http, server, location По умолчанию: off
gzip_types	Типы MIME (в дополнение к text/html), которые следует сжимать. Задание * разрешает все типы MIME	Допустимые контексты: http, server, location По умолчанию: text/html
gzip_vary	Разрешает или запрещает включение в ответ заголовка Vary: Accept-Encoding, если активна директива gzip или gzip static	Допустимые контексты: http, server, location По умолчанию: off
http	Определяет конфигурационный контекст, в котором задаются директивы HTTP-сервера	Допустимый контекст: main По умолчанию: -
if	См. таблицу «Директивы модуля rewrite» в разделе «Введение в модуль rewrite» в приложении В «Руководство по правилам переписывания»	Допустимые контексты: server, location По умолчанию: -
if_modified_since	Управляет порядком сравнения времени модификации ответа со значением в заголовке запроса If-Modified-Since. • off: заголовок If-Modified-Since игнорируется. • exact: точное соответствие (по умолчанию). • before: время модификации ответа меньше или равно значению в заголовке If-Modified-Since	Допустимые контексты: http, server, location По умолчанию: exact

Директивы	Описание	Контекст/ Умолчание
ignore_invalid_ headers	Разрешает или запрещает игнорировать заголовки с недопустимыми именами (по умолчанию on). Допустимыми считаются имена, содержащие буквы в кодировке ASCII, цифры, знак минус и, возможно, знак подчеркивания(определяется директивой underscores in headers)	Допустимые контексты: http, server По умолчанию: on
image_filter	См. таблицу «Директивы модуля image_filter» в разделе «Генерация изображений» главы 7 «NGINX для разработчика»	Допустимый контекст: location По умолчанию: -
image_filter_ buffer	Размер буфера для обработки изображений. Если потребуется больше памяти, сервер вернет ошибку 415(Unsupported Media Type)	Допустимые контексты: http, server, location По умолчанию: 1M
image_filter_jpeg_ quality	Качество результирующего изображения в формате JPEG. Не рекомендуется указывать значение больше 95	Допустимый контекст: http, server, location По умолчанию: 75
image_filter _ sharpen	Повышает резкость результирующего изображения на указанный процент	Допустимые контексты: http, server, location По умолчанию: 0
image_filter_ transparency	Определяет, сохранять ли прозрачность при обработке изображений в форматах PNG и GIF. Подразумеваемое по умолчанию значение on означает сохранение прозрачности	Допустимые контексты: http, server, location По умолчанию: on
imap_auth	Задает поддерживаемый механизм аутентификации. Допустимы значения (одно или несколько) из следующего списка: login, plain, cram-md5	Допустимые контексты: mail, server По умолчанию: plain
imap_capabilities	Определяет, какие возможности протокола IMAP4 поддерживает проксируемый сервер	Допустимые контексты: mail, server По умолчанию: IMAP4 IMAP4rev1 UIDPLUS
imap_client_buffer	Задает размер буфера для чтения команд IMAP	Допустимый контекст: mail, server По умолчанию: 4k|8к (в зависимости от платформы)

Директивы	Описание	Контекст/ Умолчание
include	Путь к файлу, содержащему дополнительные конфигурационные директивы. Может быть задан в виде маски, которой отвечает несколько файлов	Допустимый контекст: любой По умолчанию: -
index	Определяет, какой файл возвращать клиенту в ответ на URI, завершающийся знаком /. Можно указать несколько значений	Допустимые контексты: http, server, location По умолчанию: index.html
internal	Означает, что данное местоположение можно использовать только для внутренних запросов (переадресации, определенной в других директивах, переписывания, страниц ошибок и т. д.)	Допустимые контексты: location По умолчанию: -
ip_hash	Обеспечивает равномерное распределение клиентских соединений по всем серверам за счет хеширования IP-адреса по его сети класса С	Допустимый контекст: upstream По умолчанию: -
keepalive	Количество соединений с проксируемыми серверами, кэшированных в одном рабочем процессе. При использовании с HTTP-соединениями значение proxy_http_version должно быть равно 1.1, а значение proxy_set_header - Connection ""	Допустимый контекст: upstream По умолчанию: -
keepalive_disable	Запрещает соединения типа keep-alive для некоторых браузеров	Допустимые контексты: http, server, location По умолчанию: msie6
keepalive_requests	Определяет, сколько запросов можно принять по одному соединению типа keep-alive, прежде чем закрывать его	Допустимые контексты: http, server, location По умолчанию: 100
keepalive_timeout	Определяет, сколько времени соединение типа keep-alive может оставаться открытым. Можно задать второй параметр, используемый для формирования заголовка ответа «Keep-Alive»	Допустимые контексты: http, server, location По умолчанию: 75s
large_client_header_buffers	Задает максимальное число и размер буферов для чтения большого заголовка запроса клиента	Допустимые контексты: http, server По умолчанию: 4 8k

Директивы	Описание	Контекст/ Умолчание
least_conn	Активирует алгоритм балансировки нагрузки, согласно которому очередной запрос передается серверу с наименьшим числом активных соединений	Допустимый контекст: upstream По умолчанию: -
limit_conn	Определяет зону разделяемой памяти (настраиваемую с помощью директивы limit_conn_zone) и максимальное количество соединений с одинаковым значением ключа	Допустимые контексты: http, server, location По умолчанию: -
limit_conn_log_level	Если NGINX ограничивает соединения согласно директиве limit_conn, то эта директива определяет уровень протоколирования для сообщения о достижении пороговой величины	Допустимые контексты: http, server, location По умолчанию: error
limit_conn_zone	В первом параметре задается ключ, к которому относятся ограничения, указанные в директиве limit_conn. Второй параметр задает имя зоны разделяемой памяти, в которой хранится не более указанного числа соединений для каждого ключа, а также размер этой зоны (name:size)	Допустимый контекст: http По умолчанию: -
limit_except	Ограничивает HTTP-методы, доступные внутри секции location (GET включает также HEAD)	Допустимый контекст: location По умолчанию: -
limit_rate	Ограничивает скорость (в байтах/с) отдачи содержимого клиентам. Ограничение действует на уровне соединения, то есть один клиент может повысить свою пропускную способность, открыв несколько соединений	Допустимые контексты: http, server, location, if в location По умолчанию: 0
limit_rate_after	Начинает применять ограничение limit_rate после того, как передано указанное количество байтов	Допустимые контексты: http, server, location, if в location По умолчанию: 0

Директивы	Описание	Контекст/ Умолчание
limit_req	Задает ограничение при резком увеличении (всплеске) количества запросов для указанного ключа в зоне разделяемой памяти (заданной в директиве limit_req_zone). Всплеск описывается вторым параметром. Если до возникновения всплеска задерживать запросы не нужно, следует включить третий параметр nodelay	Допустимые контексты: http, server, location По умолчанию: -
limit_req_log_level	Если NGINX ограничивает количество запросов согласно директиве limit_req, то эта директива определяет уровень протоколирования для сообщения о достижении пороговой величины. Сообщение о применении задержки имеет уровень, на единицу меньший указанного в этой директиве	Допустимые контексты: http, server, location По умолчанию: -
limit_req_zone	В первом параметре задается ключ, к которому относятся ограничения, указанные в директиве limit_req. Второй параметр задает имя зоны разделяемой памяти, в которой хранится не более указанного числа запросов для каждого ключа, а также размер этой зоны (name:size). Третий параметр определяет количество запросов в секунду (r/s) или в минуту (r/m), при превышении которого начинает применяться ограничение	Допустимый контекст: http По умолчанию: -
limit_zone	Директива устарела. Использовать вместо нее limit_conn_zone	Допустимый контекст: http По умолчанию: -
lingering_close	Определяет, следует ли оставлять соединение открытым в ожидании дополнительных данных от клиента	Допустимые контексты: http, server, location По умолчанию: on
lingering_time	Связана с директивой lingering_close и определяет, сколько времени держать соединение открытым для обработки дополнительных данных	Допустимые контексты: http, server, location По умолчанию: 30s

Директивы	Описание	Контекст/ Умолчание
`lingering_timeout`	Также связана с директивой `lingering_close` и определяет, сколько времени держать соединение открытым в ожидании дополнительных данных	Допустимые контексты: `http`, `server`, `location` По умолчанию: `5s`
`listen` (http)	См. таблицу «Параметры директивы listen» в разделе «Секция с описанием виртуального сервера» главы 2 «Руководство по настройке»	Допустимый контекст: `server` По умолчанию: `*:80 | *:8000`
`listen` (mail)	Директива `listen` в NGINX однозначно идентифицирует привязку к сокету и принимает следующий параметр: • `bind`: выполнять отдельный вызов `bind()` для данной пары `address:port`	Допустимый контекст: `server` По умолчанию: -
`location`	Определяет новый контекст на основе URI запроса	Допустимые контексты: `server`, `location` По умолчанию: -
`lock_file`	Префикс имени файлов-блокировок. На некоторых платформах для реализации директивы `accept_mutex` и сериализации доступа к памяти может понадобиться файл-блокировка	Допустимый контекст: `main` По умолчанию: `logs/ngnix.lock.`
`log_format`	Определяет состав и формат полей в журнале	Допустимый контекст: `http` По умолчанию комбинация: `$remote_addr - $remote_user [$time_local], "$request" $status $body_bytes_sent, "$http_referer""$http_user_agent"'`
`log_not_found`	Подавляет запись в журнал сообщений об ошибке 404	Допустимые контексты: `http`, `server`, `location` По умолчанию: `on`

Директивы	Описание	Контекст/ Умолчание
`log_subrequest`	Разрешает или запрещает протоколирование подзапросов в журнале доступа	Допустимые контексты: `http`, `server`, `location` По умолчанию: `off`
`mail`	Определяет конфигурационный контекст, в котором задаются директивы почтового сервера	Допустимый контекст: `main` По умолчанию: -
`map`	Определяет новый контекст, в котором некоторой переменной присваивается значение, зависящее от значения исходной переменной. Формат определения контекста следующий: `map $source-variable $variable-to-be-set {...}` Сопоставляемая строка (или строки) может также быть регулярным выражением. В контексте распознаются следующие параметры: • `default`: задает значение переменной по умолчанию в случае, если значение исходной переменной не сопоставилось ни с одной из заданных строк или регулярных выражений; • `hostnames`: означает, что в качестве исходных значений можно использовать маску для первой или последней части имени хоста; • `include`: включает файл, содержащий соответствия между строками и значениями	Допустимый контекст: `http` По умолчанию: -
`map_hash_bucket_size`	Размер кластера в хеш-таблицах для директивы `map`	Допустимый контекст: `http` По умолчанию: `32|64|128`
`map_hash_max_size`	Максимальный размер хеш-таблиц для директивы `map`	Допустимый контекст: `http` По умолчанию: `2048`
`master_process`	Определяет, нужно ли запускать рабочие процессы	Допустимый контекст: `main` По умолчанию: `on`

Директивы	Описание	Контекст/ Умолчание
`max_ranges`	Задает максимальное количество диапазонов, допустимых в запросе с указанием диапазонов байтов. Если задано значение 0, то поддержка диапазонов байтов отключается	Допустимые контексты: `http`, `server`, `location` По умолчанию: -
`memcached_bind`	Определяет адрес исходящих соединений с сервером memcached	Допустимые контексты: `http`, `server`, `location` По умолчанию: -
`memcached_buffer_size`	Размер буфера для ответа от memcached. Этот ответ синхронно отправляется клиенту	Допустимые контексты: `http`, `server`, `location` По умолчанию: `4k|8k`
`memcached_connect_timeout`	Максимальное время, в течение которого NGINX ожидает установления соединения при отправке запроса серверу memcached	Допустимые контексты: `http`, `server`, `location` По умолчанию: `60s`
`memcached_gzip_flag`	Задает значение, которое, будучи найдено в ответе от сервера memcached, приведет к записи значения `gzip` в заголовок `Content-Encoding`	Допустимые контексты: `http`, `server`, `location` По умолчанию: -
`memcached_next_upstream`	См. таблицу «Директивы модуля memcached» в разделе «Кэширование в базе данных» главы 7 «NGINX для разработчика»	Допустимые контексты: `http`, `server`, `location` По умолчанию: error timeout
`memcached_pass`	Определяет имя или адрес сервера memcached и его порт. Можно также указывать группу серверов, объявленную в контексте `upstream`	Допустимые контексты: `location`, `if` в `location` По умолчанию: -
`memcached_read_timeout`	Сколько времени может пройти между двумя последовательными операциями чтения данных от сервера memcached, прежде чем соединение будет закрыто	Допустимые контексты: `http`, `server`, `location` По умолчанию: `60s`
`memcached_send_timeout`	Сколько времени может пройти между двумя последовательными операциями записи данных на сервер memcached, прежде чем соединение будет закрыто	Допустимые контексты: `http`, `server`, `location` По умолчанию: `60s`

Директивы	Описание	Контекст/ Умолчание
merge_slashes	Разрешает или запрещает удаление идущих подряд знаков косой черты. Подразумеваемое по умолчанию значение on означает. что NGINX будет заменять несколько соседних знаков / одним	Допустимые контексты: http, server По умолчанию: on
min_delete_depth	Разрешает методу WebDAV DELETE удалять файлы при условии, что число элементов в пути запроса не меньше заданного	Допустимые контексты: http, server, location По умолчанию: 0
modern_browser	Задает параметры browser и version, которые в совокупности определяют, что этот браузер следует считать современным, и в этом случае в переменную $modern_browser записывается значение, указанное в директиве modern_bowser_value. Параметр browser может принимать следующие значения: msie, gecko, opera, safari, konqueror. Альтернативно можно задать параметр unlisted, который означает, что браузер, не найденный ни в одном из списков ancient_browser и modern_browser или не приславший заголовок User-Agent, следует считать современным	Допустимые контексты: http, server, location По умолчанию: -
modern_browser_ value	Значение, записываемое в переменную $modern_browser	Допустимые контексты: http, server, location По умолчанию: 1
mp4	Активирует модуль mp4 в данной секции location	Допустимый контекст: location По умолчанию: -
mp4_buffer_size	Задает начальный размер буфера для доставки MP4-файлов	Допустимые контексты: http, server, location По умолчанию: 512K
mp4_max_buffer_size	Задает максимальный размер буфера для обработки метаданных MP4	Допустимые контексты: http, server, location По умолчанию: 10M

Директивы	Описание	Контекст/ Умолчание
msie_padding	Разрешает или запрещает добавлять комментарии в ответы со статусом больше 400 для увеличения размера ответа до 512 байт при работе с MSIE	Допустимые контексты: http, server, location По умолчанию: on
msie_refresh	Разрешает или запрещает отправлять MSIE-клиентам ответ Refresh вместо перенаправления	Допустимые контексты: http, server, location По умолчанию: off
multi_accept	Инструктирует рабочий процесс принимать сразу все новые соединения. Игнорируется в случае использования метода обработки соединений kqueue, т.к. данный метод сам сообщает число новых соединений, ожидающих приёма	Допустимый контекст: events По умолчанию: off
open_file_cache	Настраивает кэш, в котором могут храниться дескрипторы открытых файлов, информация о существовании каталогов и информация об ошибках поиска файлов	Допустимые контексты: http, server, location По умолчанию: off
open_file_cache_errors	Разрешает или запрещает кэширование ошибок поиска файлов в кэше open_file_cache	Допустимые контексты: http, server, location По умолчанию: off
open_file_cache_min_uses	Задаёт минимальное число обращений к файлу в течение времени, заданного параметром inactive директивы open_file_cache, необходимое для того, чтобы дескриптор файла оставался в кэше открытых дескрипторов	Допустимые контексты: http, server, location По умолчанию: 1
open_file_cache_valid	Задает время между последовательными проверками актуальности данных, хранящихся в кэше open_file_cache	Допустимые контексты: http, server, location По умолчанию: 60s
open_log_file_cache	См. таблицу «Директивы протоколирования из модуля HTTP» из раздела «Протоколирование» главы 6 «NGINX как HTTP-сервер»	Допустимые контексты: http, server, location По умолчанию: off
optimize_server_names	Директива устарела. Использовать вместо нее server_name_in_redirect	Допустимые контексты: http, server По умолчанию: off

Директивы	Описание	Контекст/ Умолчание
override_ charset	Определяет, следует ли осуществлять перекодирование из кодировки, указанной в заголовке Content-Type ответа, который получен от сервера, заданного в директиве proxy_pass или fastcgi_pass. Если ответ является результатом подзапроса, то перекодирование в кодировку главного запроса производится безусловно	Допустимые контексты: http, server, location, if в location По умолчанию: off
pcre_jit	Разрешает или запрещает JIT-компиляцию совместимых с Perl регулярных выражений, известных на этапе конфигурирования. Чтобы воспользоваться этой оптимизацией, необходимо включить поддержку JIT-компиляции в библиотеке PCRE	Допустимый контекст: main По умолчанию: off
perl	Активирует обработчик Perl в данном местоположении. В аргументе указывается имя обработчика или строка, содержащая полный код подпрограммы	Допустимые контексты: location, limit_except По умолчанию: -
perl_modules	Определяет дополнительные пути поиска Perl-модулей	Допустимый контекст: http По умолчанию: -
perl_require	Задает имя Perl-модуля, который должен загружаться при каждом изменении конфигурационного файла NGINX. Может встречаться несколько раз, если требуется указать несколько модулей	Допустимый контекст: http По умолчанию: -
perl_set	Устанавливает Perl-обработчик, который присваивает значение переменной. В аргументе указывается имя обработчика или строка, содержащая полный код подпрограммы	Допустимый контекст: http По умолчанию: -
pid	Файл, в котором хранится идентификатор главного процесса. Переопределяет значение, заданное на этапе конфигурирования и компиляции	Допустимый контекст: main По умолчанию: nginx.pid
pop3_auth	Задает поддерживаемый механизм аутентификации. Допустимы значения (одно или несколько) из следующего списка: plain, apop, cram-md5	Допустимые контексты: mail, server По умолчанию: plain

Директивы	Описание	Контекст/ Умолчание
pop3_capabilities	Определяет, какие возможности протокола POP3 поддерживает проксируемый сервер	Допустимые контексты: mail, server По умолчанию: TOP USER UIDL
port_in_redirect	Определяет, нужно ли указывать данный порт при переадресации средствами NGINX	Допустимые контексты: http, server, location По умолчанию: on
postpone_output	Задает минимальный размер порции данных, отправляемых клиенту. Если возможно, данные не будут отправляться, пока не накопится указанное количество	Допустимые контексты: http, server, location По умолчанию: 1460
protocol	Определяет, какой протокол поддерживает данный контекст виртуального сервера. Может принимать значения imap, pop3, smtp	Допустимый контекст: server По умолчанию: -
proxy	Разрешает или запрещает проксирование почты	Допустимый контекст: server По умолчанию: -
proxy_bind	Определяет адрес исходящих соединений с проксируемым сервером	Допустимые контексты: http, server, location По умолчанию: -
proxy_buffer	Позволяет задать размер буфера, используемого для проксирования. По умолчанию размер буфера равен размеру страницы	Допустимые контексты: mail, server По умолчанию: 4к|8к (в зависимости от платформы)
proxy_buffer_size	Размер буфера для первой части ответа от проксируемого сервера, в которой находятся заголовки	Допустимые контексты: http, server, location По умолчанию: 4k|8k (в зависимости от платформы)
proxy_buffering	Разрешает или запрещает буферизацию проксированного содержимого. Если буферизация выключена, то ответы отправляются клиенту синхронно по мере получения	Допустимые контексты: http, server, location По умолчанию: on

Директивы	Описание	Контекст/ Умолчание
proxy_buffers	Количество и размер буферов для хранения ответов от проксируемых серверов	Допустимые контексты: http, server, location По умолчанию: 8 4k|8k (в зависимости от платформы)
proxy_busy_buffers_size	Суммарный размер буферов, которые могут быть заняты для отправки ответа клиенту, пока ответ ещё не прочитан целиком. Обычно устанавливается в два раза больше, чем размер, указанный в директиве proxy_buffers	Допустимые контексты: http, server, location По умолчанию: 8k|16k (в зависимости от платформы)
proxy_cache	Определяет размер зоны разделяемой памяти, отведенной под кэш	Допустимые контексты: http, server, location По умолчанию: off
proxy_cache_bypass	Одна или несколько строковых переменных. Если хотя бы одна из них непуста и не содержит нуль, то ответ будет запрошен у проксируемого сервера, а не взят из кэша	Допустимые контексты: http, server, location По умолчанию: -
proxy_cache_key	Строка, которая используется как ключ для поиска значения в кэше. Можно использовать переменные, но следует внимательно следить за тем, чтобы не кэшировалось несколько копий одного и того же содержимого	Допустимые контексты: http, server, location По умолчанию: $scheme$proxy_host$request_uri
proxy_cache_lock	Если эта директива принимает значение on, то предотвращается отправка нескольких запросов проксируемому серверу (или серверам) в случае отсутствия в кэше	Допустимые контексты: http, server, location По умолчанию: off
proxy_cache_lock_timeout	Сколько времени запрос может ждать появления записи в кэше или освобождения блокировки proxy_cache_lock	Допустимые контексты: http, server, location По умолчанию: 5s

Директивы	Описание	Контекст/ Умолчание
proxy_cache_min_uses	Сколько запросов с данным ключом должно поступить, прежде чем ответ будет помещен в кэш	Допустимые контексты: http, server, location По умолчанию: 1
proxy_cache_path	См. таблицу «Директивы модуля proxy для управления кэшированием» в разделе «Кэширование в файловой системе» главы 7 «Обратное проксирование, дополнительные вопросы»	Допустимый контекст: http По умолчанию: -
proxy_cache_use_stale	В каких случаях допустимо использовать устаревшие кэшированные данные, если при доступе к проксируемому серверу произошла ошибка. Параметр updating разрешает использовать кэшированный ответ, если как раз в данный момент загружаются более свежие данные	Допустимые контексты: http, server, location По умолчанию: off
proxy_cache_valid	Сколько времени считать действительным кэшированный ответ с кодом 200, 301 или 302. Если перед параметром time указан необязательный код ответа, то заданное время относится только к ответу с таким кодом. Специальный параметр any означает, что в течение заданного времени следует кэшировать ответ с любым кодом	Допустимые контексты: http, server, location По умолчанию: -
proxy_connect_timeout	Максимальное время ожидания соединения с проксируемым сервером	Допустимые контексты: http, server, location По умолчанию: 60s
proxy_cookie_domain	Подменяет атрибут domain в заголовке Set-Cookie от проксируемого сервера; можно указать строку, регулярное выражение или ссылку на переменную	Допустимые контексты: http, server, location По умолчанию: off

Директивы	Описание	Контекст/ Умолчание
proxy_cookie_path	Подменяет атрибут path в заголовке Set-Cookie от проксируемого сервера; можно указать строку, регулярное выражение или ссылку на переменную	Допустимые контексты: http, server, location По умолчанию: off
proxy_headers_hash_ bucket_size	Максимальный размер имен заголовков (ни одно имя не может быть длиннее указанной в этой директиве величины)	Допустимые контексты: http, server, location По умолчанию: 64
proxy_headers_hash_ max_size	Общий размер заголовков, полученных от проксируемого сервера	Допустимые контексты: http, server, location По умолчанию: 512
proxy_hide_header	Список заголовков, которые не следует передавать клиенту	Допустимые контексты: http, server, location По умолчанию: -
proxy_http_version	Версия протокола HTTP, которой следует придерживаться при взаимодействии с проксируемым сервером (для соединений типа keepalive должна быть 1.1)	Допустимые контексты: http, server, location По умолчанию: 1.0
proxy_ignore_client_ abort	Если значение равно on, то NGINX не станет разрывать соединение с проксируемым сервером в случае, когда клиент разрывает свое соединение	Допустимые контексты: http, server, location По умолчанию: off
proxy_ignore_headers	Какие заголовки можно игнорировать при обработке ответа от проксируемого сервера	Допустимые контексты: http, server, location По умолчанию: -
proxy_intercept_ errors	Если значение равно on, то NGINX будет отображать страницу, заданную директивой error_page, вместо ответа, полученного от проксируемого сервера	Допустимые контексты: http, server, location По умолчанию: off
proxy_max_temp_file_ size	Максимальный размер временного файла, в который записывается часть ответа в случае, когда он не умещается целиком в буферах памяти	Допустимые контексты: http, server, location По умолчанию: 1024m

Директивы	Описание	Контекст/ Умолчание
`proxy_next_upstream`	Определяет условия, при которых для ответа будет выбран следующий проксируемый сервер. Это не происходит, если клиент уже что-то отправил. Условия задаются с помощью следующих параметров: • `error`: произошла ошибка при взаимодействии с проксируемым сервером; • `timeout`: произошел таймаут при взаимодействии с проксируемым сервером; • `invalid_header`: проксируемый сервер вернул пустой или недопустимый ответ; • `http_500`: проксируемый сервер вернул ответ с кодом ошибки 500; • `http_503`: проксируемый сервер вернул ответ с кодом ошибки 503; • `http_504`: проксируемый сервер вернул ответ с кодом ошибки 504; • `http_404`: проксируемый сервер вернул ответ с кодом ошибки 404; • `off`: запретить передачу запроса следующему проксируемому серверу в случае ошибки	Допустимые контексты: `http`, `server`, `location` По умолчанию: `error timeout`
`proxy_no_cache`	Определяет условия, при которых ответ не сохраняется в кэше. Одна или несколько строковых переменных. Если хотя бы одна из них непуста и не содержит нуль, то ответ не кэшируется	Допустимые контексты: `http`, `server`, `location` По умолчанию: -

Директивы	Описание	Контекст/ Умолчание
`proxy_pass`	Определяет проксируемый сервер, которому передается запрос в виде URL	Допустимые контексты: `location`, `if` в `location`, `limit_except` По умолчанию: -
`proxy_pass_ error_message`	Полезна, если процедура аутентификации на проксируемом сервере возвращает осмысленное сообщение клиенту	Допустимые контексты: `mail`, `server` По умолчанию: off
`proxy_pass_ header`	Отменяет сокрытие заголовков, определенных в директиве `proxy_ hide_header`, разрешая передавать их клиенту	Допустимые контексты: `http`, `server`, `location` По умолчанию: -
`proxy_pass_ request_ body`	Если значение равно `off`, то тело запроса не передается проксируемому серверу	Допустимые контексты: `http`, `server`, `location` По умолчанию: `on`
`proxy_pass_ request_ headers`	Если значение равно `off`, то заголовки запроса не передаются проксируемому серверу	Допустимые контексты: `http`, `server`, `location` По умолчанию: `on`
`proxy_read_ timeout`	Сколько времени может пройти между двумя последовательными операциями чтения данных от проксируемого сервера, прежде чем соединение будет закрыто. Значение следует увеличить, если проксируемый сервер обрабатывает запросы медленно	Допустимые контексты: `http`, `server`, `location` По умолчанию: `60s`
`proxy_redirect`	Перезаписывает заголовки `Location` и `Refresh`, полученные от проксируемого сервера; полезно для обхода допущений, принятых каркасом разработки приложений	Допустимые контексты: `http`, `server`, `location` По умолчанию: `default`
`proxy_send_lowat`	Если значение отлично от нуля, NGINX попытается минимизировать количество операций записи по соединению с проксируемым сервером. В Linux, Solaris и Windows игнорируется	Допустимые контексты: `http`, `server`, `location` По умолчанию: `0`
`proxy_send_ timeout`	Сколько времени может пройти между двумя последовательными операциями записи данных на проксируемый сервер, прежде чем соединение будет закрыто	Допустимые контексты: `http`, `server`, `location` По умолчанию: `60s`

Директивы	Описание	Контекст/ Умолчание
`proxy_set_body`	Эта директива позволяет изменить тело запроса, отправляемое проксируемому серверу	Допустимые контексты: `http`, `server`, `location` По умолчанию: -
`proxy_set_header`	Перезаписывает заголовки, отправляемые проксируемому серверу. Может также применяться для подавления некоторых заголовков (если в качестве значения указать пустую строку)	Допустимые контексты: `http`, `server`, `location` По умолчанию: `Host $proxy_host.Connection close`
`proxy_ssl_session_reuse`	Определяет, можно ли повторно использовать SSL-сеансы при проксировании	Допустимые контексты: `http`, `server`, `location` По умолчанию: `on`
`proxy_store`	Разрешает сохранение полученных от проксируемого сервера ответов в файлах на диске. Если параметр равен `on`, то в качестве пути к каталогу с сохраняемыми файлами используется значение, заданное в директиве `alias` или `root`. Можно вместо этого задать строку, определяющую другой каталог для хранения файлов	Допустимые контексты: `http`, `server`, `location` По умолчанию: `off`
`proxy_store_access`	Какие права доступа следует задать для новых файлов, создаваемых в соответствии с директивой `proxy_store`	Допустимые контексты: `http`, `server`, `location` По умолчанию: `user:rw`
`proxy_temp_file_write_size`	Ограничивает размер данных в одной операции записи во временный файл, чтобы NGINX не слишком долго блокировала исполнение программы при обработке одного запроса	Допустимые контексты: `http`, `server`, `location` По умолчанию: `8k|16k` (в зависимости от платформы)
`proxy_temp_path`	Каталог для хранения временных файлов, получаемых от проксируемого сервера. Может быть многоуровневым. Второй, третий и четвертый параметры (если заданы) определяют иерархию подкаталогов в виде количества знаков в имени подкаталога	Допустимые контексты: `http`, `server`, `location` По умолчанию: `proxy_temp`

Директивы	Описание	Контекст/ Умолчание
proxy_timeout	Используется, если таймаут должен быть больше значения по умолчанию - 24 часа	Допустимые контексты: mail, server По умолчанию: 24h
random_index	Активирует выбор случайного файла для отправки пользователю, если URI заканчивается знаком /	Допустимый контекст: location По умолчанию: off
read_ahead	Если возможно, ядро будет сразу считывать из файла столько байтов, сколько указано в параметре size. Поддерживается в текущих версиях FreeBSD и Linux (в Linux параметр size игнорируется)	Допустимые контексты: http, server, location По умолчанию: 0
real_ip_header	Задает название заголовка, значение которого рассматривается как IP-адрес клиента, когда директива set_real_ip_from соответствует IP-адресу отправителя	Допустимые контексты: http, server, location По умолчанию: x-Real-IP
real_ip_recursive	Означает, что в случае, когда заголовок real_ip_header содержит несколько адресов, использовать нужно последний	Допустимые контексты: http, server, location По умолчанию: off
recursive_error_pages	Разрешает производить несколько переадресаций с помощью директивы error page	Допустимые контексты: http, server, location По умолчанию: off
referer_hash_bucket_size	Размер кластера в хеш-таблице допустимых рефереров	Допустимые контексты: server, location По умолчанию: 64
referer_hash_max_size	Максимальный размер хеш-таблицы допустимых рефереров	Допустимые контексты: server, location По умолчанию: 2048
request_pool_size	Позволяет производить точную настройку выделения памяти под конкретные запросы	Допустимые контексты: http, server По умолчанию: 4k
reset_timedout_connection	Если значение этой директивы равно on, то сокеты, для которых истек таймаут, сбрасываются немедленно, в результате чего освобождается выделенная для них память. По умолчанию сокет остается в состоянии FIN_WAIT1. На соединения типа keep-alive эта директива не распространяется, они всегда закрываются обычным образом	Допустимые контексты: http, server, location По умолчанию: off

Директивы	Описание	Контекст/ Умолчание
resolver	Задает имена одного или нескольких серверов, используемых для получения IP-адресов по доменным именам. Необязательный параметр valid переопределяет время TTL, заданное в записи о доменном имени	Допустимые контексты: http, server, location По умолчанию: -
resolver_ timeout	Задает таймаут разрешения имен	Допустимые контексты: http, server, location По умолчанию: 30s
return	Прекращает обработку и возвращает указанный код клиенту. При возврате нестандартного кода 444 соединение закрывается без отправки заголовков ответа. Если помимо кода указывается еще и текст, то он помещается в тело ответа. Если же вместо текста указан URL-адрес, то он становится значением заголовка Location. Если указан URL-адрес без кода, то подразумевается код 302	Допустимые контексты: server, location, if По умолчанию: -
rewrite	См. таблицу «Директивы модуля rewrite» в разделе «Введение в модуль rewrite» в приложении В «Руководство по правилам переписывания»	Допустимые контексты: server, location, if По умолчанию: -
rewrite_log	Разрешает или запрещает записывать в error_log на уровне notice результаты перезаписи URL	Допустимые контексты: http, server, if в server, location, if в location По умолчанию: off
root	Задает путь к корню документов. Для поиска файлов строка, указанная в URI-адреса, дописывается в конец этого значения	Допустимые контексты: http, server, location, if в location По умолчанию: html
satisfy	Разрешает доступ, если все (all) или хотя бы одна (any) из директив access или auth_basic разрешает доступ. Подразумеваемое по умолчанию значение all говорит, что пользователь должен находиться в определенной сети и ввести правильный пароль	Допустимые контексты: http, server, location По умолчанию: all

Директивы	Описание	Контекст/ Умолчание
satisfy_any	Директива устарела. Использовать директиву satisfy с параметром any	Допустимые контексты: http, server, location По умолчанию: off
secure_link_secret	Затравка для вычисления MD5-свертки ссылки	Допустимый контекст: location По умолчанию: -
send_lowat	Если значение отлично от нуля, то NGINX пытается минимизировать количество операций отправки данных через клиентские сокеты. В Linux, Solaris и Windows эта директива игнорируется	Допустимые контексты: http, server, location По умолчанию: 0
send_timeout	Задает таймаут между двумя последовательными операциями записи при передаче ответа клиенту	Допустимые контексты: http, server, location По умолчанию: 60s
sendfile	Разрешает использовать системный вызов sendfile (2) для прямого копирования из одного файлового дескриптора в другой	Допустимые контексты: http, server, location, if в location По умолчанию: off
sendfile_max_chunk	Задает максимальный размер данных, который можно скопировать за один вызов sendfile (2). Без этого ограничения одно быстрое соединение может целиком захватить рабочий процесс	Допустимые контексты: http, server, location По умолчанию: 0
server (http)	Создает новый конфигурационный контекст, в котором определяется виртуальный хост. Директива listen определяет IP-адрес(а) и порт(ы). В директиве server_name перечисляются значения заголовков Host, сопоставляемые с этим контекстом	Допустимый контекст: http По умолчанию: -
server (upstream)	См. таблицу «Директивы модуля upstream» в разделе «Модуль upstream» главы 4 «NGINX как обратный прокси-сервер»	Допустимый контекст: upstream По умолчанию: -
server (mail)	Создает новый конфигурационный контекст, в котором определяется почтовый сервер. Директива listen определяет IP-адрес(а) и порт(ы). В директиве server_name указывается имя сервера	Допустимый контекст: mail По умолчанию: -

Директивы	Описание	Контекст/ Умолчание
server_name (http)	Задает имя, на которое отвечает данный виртуальный хост	Допустимый контекст: server По умолчанию: ""
server_name (mail)	Задает имя сервера, используемое в следующих случаях: • приветствие в протоколе POP3/SMTP; • затравка для аутентификации по методу SASL CRAM-MD5; • имя в команде EHLO, когда для взаимодействия с проксируемым SMTP-сервером используется xclient	Допустимые контексты: mail, server По умолчанию: hostname
server_name_in_ redirect	Разрешает использовать первое из указанных в директиве server_name значений при любой переадресации, произведенной NGINX в данном контексте	Допустимые контексты: http, server, location По умолчанию: off
server_names_ hash_bucket_size	Размер кластера в хеш-таблице имен серверов	Допустимый контекст: http По умолчанию: 32|64|128 (в зависимости от процессора)
server_names_ hash_max_size	Максимальный размер хеш-таблицы имен серверов	Допустимый контекст: http По умолчанию: 512
server_tokens	Разрешает или запрещает включение номера версии NGINX в отправляемые сообщения об ошибках и заголовок Server	Допустимые контексты: http, server, location По умолчанию: on
set	Присваивает значение указанной переменной	Допустимые контексты: server, location, if По умолчанию: -
set_real_ip_from	Задает один или несколько адресов, для которых IP-адрес клиента будет извлекаться из заголовка, указанного в директиве real_ip_header. Значение unix: говорит, что таким образом следует трактовать все сокеты в домене UNIX	Допустимые контексты: http, server, location По умолчанию: -
smtp_auth	Задает поддерживаемый механизм аутентификации SASL-клиента. Может принимать одно или несколько значений из списка login, plain и cram-md5	Допустимые контексты: mail, server По умолчанию: login, plain

Директивы	Описание	Контекст/ Умолчание
smtp_capabilities	Определяет, какие возможности поддерживает проксируемый SMTP-сервер	Допустимые контексты: mail, server По умолчанию: -
so_keepalive	Для соединения с проксируемым сервером задается режим TCP keepalive	Допустимые контексты: mail, server По умолчанию: off
source_charset	Определяет кодировку ответа. Если она отличается от указанной в директиве charset, то производится перекодирование	Допустимые контексты: http, server, location, if в location По умолчанию: -
split_clients	Создает контекст, в котором определены переменные для A/B-тестирования. Строка, заданная в первом параметре, хешируется согласно алгоритму MurmurHash2. Затем переменной, заданной во втором параметре, присваивается значение, зависящее от того, в какой диапазон попадает хеш первой строки. Веса задаются либо в процентах, либо в виде символа *	Допустимый контекст: http По умолчанию: -
ssi	Разрешает обработку SSI-файлов	Допустимые контексты: http, server, location, if в location По умолчанию: off
ssi_min_file_chunk	Задаёт минимальный размер файла, начиная с которого имеет смысл посылать его с помощью системного вызова sendfile (2)	Допустимые контексты: http, server, location По умолчанию: 1k
ssi_silent_errors	Подавляет сообщение, которое обычно выводится в случае ошибки во время обработки SSI	Допустимые контексты: http, server, location По умолчанию: off
ssi_types	Указывается список MIME-типов ответа (в дополнение к text/html), для которых обрабатываются команды SSI. Звездочка (*) означает все MIME-типы	Допустимые контексты: http, server, location По умолчанию: text/html
ssi_value_length	Задаёт максимальную длину значений параметров в командах SSI	Допустимые контексты: http, server, location По умолчанию: 256

Директивы	Описание	Контекст/ Умолчание
`ssl` (http)	Включает или выключает поддержку SSL-транзакций в этом контексте	Допустимые контексты: `http`, `server` По умолчанию: `off`
`ssl` (mail)	Определяет, следует ли поддерживать в этом контексте транзакции SSL/TLS	Допустимые контексты: `mail`, `server` По умолчанию: `off`
`ssl_certificate` (http)	Путь к SSL-сертификату в формате PEM для данного виртуального сервера. Если требуются промежуточные сертификаты, то их необходимо поместить в тот же файл после основного сертификата в нужном порядке (корневой - последним)	Допустимые контексты: `http`, `server` По умолчанию: -
`ssl_certificate` (mail)	Путь к SSL-сертификату (или сертификатам) в формате PEM для данного виртуального сервера	Допустимые контексты: `mail`, `server` По умолчанию: -
`ssl_certificate_key` (http)	Путь к файлу, содержащему секретный ключ SSL-сертификата	Допустимые контексты: `http`, `server` По умолчанию: -
`ssl_certificate_key` (mail)	Путь к файлу, содержащему секретный ключ SSL в формате PEM для данного виртуального сервера	Допустимые контексты: `mail`, `server` По умолчанию: -
`ssl_ciphers`	Определяет, какие шифры нужно поддерживать в этом контексте (в формате OpenSSL)	Допустимые контексты: `http`, `server` По умолчанию: `HIGH:!aNULL:!MD5`
`ssl_client_ rertificate`	Путь к файлу, содержащему сертификат(ы) в формате PEM общеизвестных удостоверяющих центров, подписывающих сертификаты клиентов	Допустимые контексты: `http`, `server` По умолчанию: -
`ssl_crl`	Путь к файлу, содержащему **список отозванных сертификатов (CRL)** в формате PEM	Допустимые контексты: `http`, `server` По умолчанию: -
`ssl_dhparam`	Путь к файлу с параметрами для шифров с обменом EDH-ключами	Допустимые контексты: `http`, `server` По умолчанию: -

Директивы	Описание	Контекст/ Умолчание
ssl_engine	Задает аппаратный ускоритель SSL	Допустимый контекст: main По умолчанию: -
ssl_prefer_ server_ ciphers (http)	Указывает, что при использовании протоколов SSLv3 и TLSv1 серверные шифры более приоритетны, чем клиентские	Допустимые контексты: http, server По умолчанию: off
ssl_prefer_ server_ ciphers (mail)	Указывает, что при использовании протоколов SSLv3 и TLSv1 серверные шифры более приоритетны, чем клиентские	Допустимые контексты: mail, server По умолчанию: off
ssl_protocols (http)	Определяет, какие протоколы SSL разрешить	Допустимые контексты: http, server По умолчанию: SSLv3, TLSv1, TLSv1.1, TLSv1.2
ssl_protocols (mail)	Определяет, какие протоколы SSL разрешить	Допустимые контексты: mail, server По умолчанию: SSLv3, TLSv1, TLSv1.1, TLSv1.2
ssl_session_cache (http)	Определяет тип и размер кэша параметров сеансов SSL. Допустимы следующие типы кэша: • off: клиентам сообщается, что сеансы повторно не используются; • none: клиентам сообщается, что сеансы используются повторно, но на самом деле это не так; • builtin: встроенный в OpenSSL кэш используется только одним рабочим процессом; размер кэша задается в сессиях; • shared: кэш разделяется всеми рабочими процессами, задается имя кэша и размер сессии в мегабайтах	Допустимые контексты: http, server По умолчанию: none

Директивы	Описание	Контекст/ Умолчание
ssl_session_cache (mail)	Определяет тип и размер кэша параметров сеансов SSL. Допустимы следующие типы кэша: • off: клиентам сообщается, что сеансы повторно не используются; • none: клиентам сообщается, что сеансы используются повторно, но на самом деле это не так; • builtin: встроенный в OpenSSL кэш используется только одним рабочим процессом; размер кэша задается в сессиях; • shared: кэш разделяется всеми рабочими процессами, задается имя кэша и размер сессии в мегабайтах	Допустимые контексты: mail, server По умолчанию: none
ssl_session_timeout (http)	Определяет, сколько времени клиент может пользоваться одними и теми же параметрами SSL при условии, что они хранятся в кэше	Допустимые контексты: http, server По умолчанию: 5m
ssl_session_timeout (mail)	Определяет, сколько времени клиент может пользоваться одними и теми же параметрами SSL при условии, что они хранятся в кэше	Допустимые контексты: mail, server По умолчанию: 5m
ssl_stapling	Разрешает подшивку (stapling) ответов по протоколу OCSP. Сертификат издателя сертификата сервера должен находиться в файле, который указан в директиве ssl_trusted_certificate. Необходимо также указать DNS-сервер, который может разрешить доменное имя OCSP-сервера	Допустимые контексты: http, server По умолчанию: off
ssl_stapling_file	Путь к файлу в формате DER, содержащему подшиваемый ответ OCSP-сервера	Допустимые контексты: http, server По умолчанию: -
ssl_stapling_ responder	URL-адрес OCSP-сервера. В настоящее время поддерживаются только URL со схемой http://	Допустимые контексты: http, server По умолчанию: -

Директивы	Описание	Контекст/ Умолчание
ssl_stapling_ verify	Разрешает верификацию ответов OCSP-сервера	Допустимые контексты: http, server По умолчанию: -
ssl_trusted_ certificate	Если включён режим ssl_stapling, то задает путь к файлу с SSL-сертификатами в формате PEM удостоверяющих центров, которые подписывают клиентские сертификаты и ответы OCSP	Допустимые контексты: http, server По умолчанию: -
ssl_verify_client	Включает проверку клиентских SSL-сертификатов. Если задан параметр optional, то клиентский сертификат запрашивается и при его наличии проверяется. Если задан параметр optional_no_ca, то клиентский сертификат запрашивается, но не требуется, чтобы он был подписан доверенным УЦ	Допустимые контексты: http, server По умолчанию: off
ssl_verify_depth	Какое максимальное число подписавших сторон следует проверить, прежде чем объявить сертификат недействительным	Допустимые контексты: http, server По умолчанию: 1
starttls	Определяет, поддерживаются ли команды STLS/STARTTLS и (или) требуются ли они для взаимодействия с данным сервером	Допустимые контексты: mail, server По умолчанию: off
sub_filter	Задает заменяемую строку (без учета регистра) и заменяющую строку. Заменяющая строка может содержать переменные	Допустимые контексты: http, server, location По умолчанию: -
sub_filter_once	Если параметр равен off, то директива sub_filter применяется ко всем вхождениям заменяемой строки	Допустимые контексты: http, server, location По умолчанию: on
sub_filter_types	Указывается список MIME-типов ответа (в дополнение к text/html), для которых производится замена. Звездочка (*) означает все MIME-типы	Допустимые контексты: http, server, location По умолчанию: text/ html
tcp_nodelay	Разрешает или запрещает использование параметра TCP_ NODELAY для соединений типа keep-alive	Допустимые контексты: http, server, location По умолчанию: on

Директивы	Описание	Контекст/ Умолчание
`tcp_nopush`	Учитывается только при использовании директивы `sendfile`. Разрешает NGINX отправлять заголовки ответа одним пакетом, а также передавать файл полными пакетами	Допустимые контексты: `http`, `server`, `location` По умолчанию: `off`
`timeout`	Сколько времени NGINX будет ждать установления соединения с проксируемым сервером	Допустимые контексты: `mail`, `server` По умолчанию: `60s`
`timer_resolution`	Определяет, как часто вызывать функцию `gettimeofday()`. По умолчанию она вызывается при каждом получении события от ядра	Допустимый контекст: `main` По умолчанию: -
`try_files`	Проверяет существование файлов, указанных в параметрах. Если ни один файл, кроме последнего, не найден, то последний параметр считается «последней надеждой», поэтому позаботьтесь о том, чтобы такой файл или именованное местоположение существовали	Допустимые контексты: `server`, `location` По умолчанию: -
`types`	Определяет соответствие между MIME-типами и расширениями имен файлов. В дистрибутив NGINX входит файл `conf/mime.types`, содержащий большинство соответствий. Как правило, достаточно просто включить этот файл директивой `include`	Допустимые контексты: `http`, `server`, `location` По умолчанию: `text/html html; image/gif gif; image/jpeg jpg;`
`types_hash_bucket_size`	Размер кластера хеш-таблицы MIME-типов	Допустимые контексты: `http`, `server`, `location` По умолчанию: `32|64|128` (в зависимости от процессора)

Директивы	Описание	Контекст/ Умолчание
types_hash_max_ size	Максимальный размер хеш-таблицы MIME-типов	Допустимые контексты: http, server, location По умолчанию: 1024
underscores_in_ headers	Разрешает или запрещает использование символа подчеркивания в заголовках запросов от клиентов. Если оставлено подразумеваемое по умолчанию значение off, то анализ таких заголовков производится согласно режиму, заданному в директиве ignore_ invalid_headers	Допустимые контексты: http, server По умолчанию: off
uninitialized, variable_warn	Следует ли помещать в журнал сообщения о неинициализированных переменных	Допустимые контексты: http, server, location, if По умолчанию: on
upstream	Открывает именованный контекст, в котором определена группа серверов	Допустимый контекст: http По умолчанию: -
use	Задает способ обработки соединений. Эта директива переопределяет режим, заданный при компиляции, и, если используется, то должна находиться в контексте events. Она особенно полезна, если обнаруживается, что со временем режим по умолчанию начинает приводить к ошибкам	Допустимый контекст: events По умолчанию: -
user	Пользователь и группа, от имени которых исполняются рабочие процессы. Если группа опущена, то подразумевается группа, имя которой совпадает с именем пользователя	Допустимый контекст: main По умолчанию: nobody nobody

Директивы	Описание	Контекст/ Умолчание
`userid`	Активирует модуль в соответствии со следующими параметрами: • `on`: устанавливает куки версии 2 и записывает в журнал полученные куки; • `v1`: устанавливает куки версии 1 и записывает в журнал полученные куки; • `log`: отключает установку куков, но продолжает записывать их в журнал; • `off`: отключает как установку, так и протоколирование куков	Допустимые контексты: `http`, `server`, `location` По умолчанию: `off`
`userid_domain`	Настраивает записываемый в куки домен	Допустимые контексты: `http`, `server`, `location` По умолчанию: `none`
`userid_expires`	Задает срок хранения кука. Если указано значение `max`, то устанавливается дата `31 Dec 2037 23:55:55 GMT`	Допустимые контексты: `http`, `server`, `location` По умолчанию: -
`userid_mark`	Задает первый символ окончания base64-представления кука с именем `userid_name`	Допустимые контексты: `http`, `server`, `location` По умолчанию: `off`
`userid_name`	Задает имя кука	Допустимые контексты: `http`, `server`, `location` По умолчанию: `uid`
`userid_p3p`	Настраивает заголовок P3P для сайтов, которые объявляют свою политику конфиденциальности, следуя протоколу **Platform for Privacy Preferences Project**	Допустимые контексты: `http`, `server`, `location` По умолчанию: -
`userid_path`	Задает путь, указываемый в куке	Допустимые контексты: `http`, `server`, `location` По умолчанию: /
`userid_service`	Идентификатор службы, которая устанавливает кук. Например, для куков версии 2 по умолчанию подразумевается IP-адрес сервера, установившего кук	Допустимые контексты: `http`, `server`, `location` По умолчанию: IP-адрес сервера

Директивы	Описание	Контекст/ Умолчание
valid_referers	Определяет, при каких значениях заголовка Referer переменная $invalid_referer будет содержать пустую строку В остальных случаях в эту переменную будет записана 1. Параметр может принимать одно или несколько значений из следующего списка: • none: заголовка Referer нет; • blocked: заголовок Referer присутствует, но пуст, или содержит URL без схемы; • server_names: в заголовке Referer указано одно из имен, перечисленных в директиве server_name; • произвольная строка: задает имя сервера в заголовке Referer, возможно с префиксом URI и знаком * в начале или в конце; • регулярное выражение: сопоставляется с частью указанного в заголовке Referer значения после схемы	Допустимые контексты: server, location По умолчанию: -
variables_hash_bucket_size	Размер кластера хеш-таблицы переменных	Допустимый контекст: http По умолчанию: 64
variables_hash_max_size	Максимальный размер хеш-таблицы переменных	Допустимые контексты: http, server, location По умолчанию: 512
worker_aio_requests	Количество открытых асинхронных операций ввода-вывода в одном рабочем процессе при использовании aio в режиме epoll	Допустимый контекст: events По умолчанию: 32
worker_connections	Задает максимальное число соединений, одновременно открытых в одном рабочем процессе. Сюда входят в частности соединения с клиентами и с проксируемыми серверами (но не только)	Допустимый контекст: events По умолчанию: 512

Директивы	Описание	Контекст/ Умолчание
worker_cpu_ affinity	Привязывает рабочие процессы к группам процессоров, как описано в битовой маске. Доступна только в ОС FreeBSD и Linux	Допустимый контекст: main По умолчанию: -
worker_priority	Задает приоритет планирования для рабочих процессов. Работает, как команда nice - отрицательные значения означают более высокий приоритет	Допустимый контекст: main По умолчанию: 0
worker_processes	Количество рабочих процессов, создаваемых сразу после запуска. Эти процессы обрабатывают запросы на соединения со стороны клиентов. Правильный выбор значения сложная задача, но для начала можно взять количество процессорных ядер	Допустимый контекст: main По умолчанию: 1
worker_rlimit_core	Изменяет ограничение на размер файла core для рабочих процессов	Допустимый контекст: main По умолчанию: -
worker_rlimit_ nofile	Изменяет ограничение на количество файлов, открытых в рабочем процессе	Допустимый контекст: main По умолчанию: -
worker_rlimit_ sigpending	Изменяет ограничение на размер очереди сигналов, для рабочего процесса при использовании метода обработки соединений rtsig	Допустимый контекст: main По умолчанию: -
working_directory	Текущий каталог для рабочих процессов. Чтобы рабочие процессы могли создавать файлы core, в этот каталог должна быть разрешена запись	Допустимый контекст: main По умолчанию: -
xclient	В протоколе SMTP предусмотрена проверка на основе параметров IP/HELO/LOGIN, которые передаются в команде XCLIENT. Данная директива позволяет NGINX передать эту информацию серверу	Допустимые контексты: mail, server По умолчанию: on
xml_entities	Путь к файлу DTD-схемы, где объявлены знаковые сущности, на которые есть ссылки в обрабатываемом XML-файле	Допустимые контексты: http, server, location По умолчанию: -

Директивы	Описание	Контекст/ Умолчание
`xslt_param`	Параметры, передаваемые в таблицы стилей; значениями являются выражения XPath	Допустимые контексты: `http`, `server`, `location` По умолчанию: -
`xslt_string_param`	Параметры, передаваемые в таблицы стилей; значениями являются строки	Допустимые контексты: `http`, `server`, `location` По умолчанию: -
`xslt_stylesheet`	Путь к таблице стилей XSLT, применяемой для преобразования XML-ответа. Можно передать параметры в виде списка пар ключ-значение	Допустимый контекст: `location` По умолчанию: -
`xslt_types`	Указывается список MIME-типов ответа (в дополнение к `text/xml`), для которых производится замена. Звездочка (`*`) означает все MIME-типы. Если результатом преобразования является HTML-документ, то его MIME-тип будет заменен на `text/html`	Допустимые контексты: `http`, `server`, `location` По умолчанию: `text/xml`

Приложение B. Руководство по правилам переписывания

Это приложение является введением в модуль NGINX `rewrite` и может использоваться как руководство по созданию новых правил и преобразованию правил переписывания из формата Apache в формат NGINX. Мы обсудим следующие вопросы.

- Введение в модуль `rewrite`.
- Создание новых правил переписывания.
- Преобразование правил из формата Apache.

Введение в модуль rewrite

Модуль `rewrite` в NGINX представляет собой простой сопоставитель с регулярными выражениями в сочетании с виртуальной стековой машиной. Первая часть любого правила переписывания - это регулярное выражение. Некоторые его части можно заключить в круглые скобки, это будут «запоминаемые подвыражения», на которые впоследствии можно сослаться с помощью позиционных переменных. Значение позиционной переменной зависит от порядка следования соответствующего ей запоминаемого подвыражения. Позиционные переменные обозначаются числами, так что `$1` ссылается на значение первого запомненного подвыражения, `$2` - на значение второго и т. д. Рассмотрим, например, такое регулярное выражение:

```
^/images/([a-z]{2})/([a-z0-9]{5})/(.*)\.(png|jpg|gif)$
```

Первая позиционная переменная `$1` ссылается на строку из двух букв, которая следует сразу за строкой `/images/` в начале URI-адреса. Вторая позиционная переменная `$2` ссылается на строку из пяти символов - строчных букв и цифр от 0 до 9. Третья позиционная

переменная `$3` предположительно является именем файла. И последняя переменная, выделяемая этим регулярным выражением, `$4`, содержит одну из строк `png`, `jpg` или `gif`, находящуюся в самом конце URI.

Вторая часть правила переписывания - это переписанный URI-адрес запроса. Он может содержать позиционные переменные, выделенные регулярным выражением, а также любые другие переменные, допустимые на данном уровне конфигурационного файла NGINX:

```
/data?file=$3.$4
```

Если этот URI не соответствует ни одному местоположению, определенному в конфигурации NGINX, то он возвращается клиенту в заголовке `Location` с кодом состояния 301 (Moved Permanently) или 302 (Found), означающим, что требуется переадресация (постоянная или временная). Код состояния можно указать и явно, если третий параметр директивы равен `permanent` или `redirect`.

Третьим параметром правила переписывания может быть также ключевое слово `last` или `break`, означающее, что никаких других директив модуля `rewrite` обрабатывать не надо. Флаг `last` инструктирует NGINX искать местоположение, соответствующее переписанному URI.

```
rewrite '^/images/([a-z]{2})/([a-z0-9]{5})/(.*)\.(png|jpg|gif)$' /
data?file=$3.$4 last;
```

Флаг `break` может употребляться и в качестве самостоятельной директивы, чтобы прекратить обработку директивы модуля `rewrite` внутри блока `if` или в другом контексте, где модуль `rewrite` активен. Во фрагменте кода ниже предполагается, что какой-то внешний метод записывает в переменную `$bwhog` непустое и ненулевое значение в случае, когда клиент потребляет слишком большую часть полосы пропускания. Тогда директива `limit_rate` принудительно понизит скорость передачи. Директива `break` в данном случае включена потому, что блок `if` находится внутри модуля `rewrite`, и мы не хотим обрабатывать другие директивы этого модуля.

```
if ($bwhog) {
    limit_rate 300k;
    break;
}
```

Еще один способ прекратить обработку директив модуля `rewrite` состоит в том, чтобы с помощью директивы return вернуть управление главному модулю `http`, обрабатывающему запрос. Это может означать, что NG1NX возвращает информацию непосредственно клиенту, по чаще `return` употребляется в сочетании с `error_page`, чтобы либо представить клиенту отформатированную HTML-страницу, либо активировать другой модуль, который завершит обработку запроса. В директиве `return` можно указать только код состояния, код состояния с текстом или код состояния с URI-адресом. Если в качестве единственного параметра указан URI-адрес, подразумевается код состояния 302. Если за кодом состояния следует текст, то он становится телом ответа. Если же после кода состояния указан URI-адрес, то он становится значением заголовка `Location`, то есть адресом, на который переадресуется клиент.

Пусть, например, мы хотим задать короткий текст сообщения о том, что в некотором местоположении не найден файл. Местоположение опишем со знаком равенства, то есть будем сравнивать на точное совпадение:

```
location = /image404.html {
    return 404 "image not found\n";
}
```

На любое обращение к этому URI-адресу будет возвращен ответ с кодом 404 и текстом **image not found\n**. Таким образом, мы могли бы использовать местоположение `/image404.html` в конце директивы `try_files` или как страницу ошибки для графических файлов.

Помимо директив, относящихся собственно к переписыванию URI, модуль `rewrite` включает также директиву `set`, которая создает переменные и присваивает им значения. Это полезно для разных целей: от создания флагов при выполнении некоторых условий до передачи именованных аргументов в другое местоположение и протоколирования произведенных действий.

В примере ниже демонстрируются некоторые из этих идей и применение соответствующих директив.

```
http {

  # специальный формат журнала, в котором используются переменные,
  # определяемые ниже
  log_format imagelog '[$time_local] ' $image_file ' ' $image_type '
      ' $body_bytes_sent ' ' $status;
```

```
# хотим включить отладку правил переписывания, чтобы убедиться,
# что правило работает, как задумано


rewrite_log on;
  server {

 root /home/www;
  location / {
    # определяем, в какой журнал должны записываться отладочные
    # сообщения, касающиеся правил переписывания

    error_log logs/rewrite.log notice;
    # Наше правило, в котором используются запоминаемые подвыражения
    # и позиционные переменные. Обратите внимание на кавычки вокруг
    # регулярного выражения - они необходимы, потому что фигурные
    # скобки {} встречаются в самом выражении
    rewrite '^/images/([a-z]{2})/([a-z0-9]{5})/(.*)\.

    (png|jpg|gif)$' /data?file=$3.$4;
    # Обратите внимание на отсутствие параметра 'last' в выражении выше.
    # Если бы мы включили его, то переменные не были бы созданы, потому
    # NGINX прекратила бы обработку директив модуля rewrite.
    # Здесь устанавливаются переменные, используемые в формате
    # журнала 'imagelog'

    set $image_file $3;

    set $image_type $4;
  }
  location /data {
    # мы хотим протоколировать обращения ко всем изображениям в
    # этом журнале со специальным форматом, чтобы потом было проще
    # выделить тип и размер

    access_log logs/images.log imagelog;
    root /data/images;
    # можно было бы воспользоваться определенными ранее переменными
    # $image-*, но ссылка на аргумент выглядит понятнее

    try_files /$arg_file /image404.html;
    }
 location = /image404.html {

      # специальное сообщение об ошибке для несуществующих
      # графических файлов
      return 404 "image not found\n";

    }
  }
 }
```

В таблице ниже перечислены все директивы модуля `rewrite`, рассматриваемые в этом разделе.

Директивы модуля rewrite

Директива	Описание
`break`	Завершает обработку директив модуля `rewrite`, находящихся в текущем контексте
`if`	Вычисляет условие и, если оно истинно, то выполняет следующие далее директивы модуля `rewrite`: `if (условие) { ... }` Допустимые условия: • имя переменной: `false`, если значение пусто или является строкой, которая начинается символом 0; • сравнение строк: с использованием операторов `=` и `! =`; • сопоставление с регулярным выражением: с использованием операторов `~` (с учетом регистра), `~*` (без учета регистра) и соответствующих операторов отрицания `!~` и `!~*`; • существование файла: с использованием операторов `-f` и `! -f`; • существование каталога: с использованием операторов `-d` и `! -d`; • существование файла, каталога или символической ссылки: с использованием операторов `-e` и `! -e`; • исполнимость файла: с использованием операторов `-x` и `! -x`;
`return`	Прекращает обработку и возвращает клиенту указанный код. Нестандартный код 444 означает закрытие соединения без отправки заголовков ответа. Если с кодом дополнительно ассоциирован текст, то он помещается в тело ответа. Если же за кодом следует URL-адрес, то он становится значением заголовка Location. При наличии одного лишь URL-адреса, без кода, подразумевается код 302
`rewrite`	Подменяет URI-адрес, сопоставившийся с регулярным выражением, заданным в первом параметре, строкой из второго параметра. В качестве необязательного третьего параметра может быть задан один из следующих флагов: • `last`: прекращает обработку директив модуля rewrite и ищет местоположение, соответствующее измененному URI-адресу; • `break`: прекращает обработку директив модуля `rewrite`; • `redirect`: возвращает код временной переадресации (302); используется, если URI не начинается схемой; • `permanent`: возвращает код постоянной переадресации (301)
`rewrite_log`	Разрешает или запрещает записывать в `error_log` на уровне `notice` результаты обработки директив модуля `rewrite`
`set`	Присваивает значение указанной переменной
`unitialized_variable_warn`	Определяет, нужно ли записывать в журнал предупреждения о неинициализированных переменных

Создание новых правил переписывания

Создавая повое правило с нуля, подумайте, чего точно вы хотите добиться. Задайте себе следующие вопросы.

- Как устроены мои URL-адреса?
- Можно ли попасть на некоторую страницу несколькими способами?
- Хочу ли я запоминать части URL в переменных?
- Буду ли я осуществлять переадресацию на сайт, находящийся на другом сервере, или могу снова наткнуться на свое же правило?
- Хочу ли я заменять аргументы в строке запроса?

Уяснить, как устроены URL-адреса, можно путем анализа структуры сайта или веб-приложения. Если на некоторую страницу можно попасть несколькими способами, то создайте правило переписывания, которое отправит клиенту код постоянной переадресации. Имея эту информацию, вы сможете построить каноническое представление сайта или приложения. Это не только поможет навести порядок в URL, но и позволит проще находить ваш сайт в Интернете.

Например, пусть имеется контроллер `home`, обрабатывающий адреса по умолчанию, и пусть попасть на тот же контроллер можно через индексную страницу. Таким образом, при переходе по любому из следующих URL пользователь получит одну и ту же информацию:

```
/
/home
/home/
/home/index
/home/index/
/index
/index.php
/index.php/
```

Было бы эффективнее переадресовать запросы, содержащие имя контроллера и (или) индексную страницу, на корень:

```
rewrite ^/(home(/index)?|index(\.php)?)/?$ $scheme://$host/ permanent;
```

Мы воспользовались переменными `$scheme` и `$host`, потому что производим постоянную переадресацию (код 301) и хотим, чтобы NGINX построила URL с теми же параметрами, которые были в исходном URL.

Чтобы поместить в журнал отдельные части URL, можно было бы включить в регулярное выражение выделяющие их запоминаемые подвыражения, затем скопировать позиционные переменные в именованные, на которые уже сослаться в определении формата журнала. Подобный пример мы видели в предыдущем разделе. Компоненты выделяются следующим образом:

```
log_format imagelog '[$time_local] ' $image_file ' ' $image_type ' '
$body_bytes_sent ' ' $status;

rewrite '^/images/([a-z]{2})/([a-z0-9]{5})/(.*)\.(png|jpg|gif)$' /
data?file=$3.$4;

set $image_file $3;
set $image_type $4;

access_log logs/images.log imagelog;
```

Если правило переписывания завершается внутренней переадресацией или инструктирует клиента обратиться к местоположению, в котором определено само это правило, то следует позаботиться о том, чтобы не зациклиться. Например, в контексте `server` правило можно определить с флагом `last`, но при определении в местоположении, на которое оно ссылается, необходимо использовать флаг `break`.

```
server {
  rewrite ^(/images)/(.*)\.(png|jpg|gif)$ $1/$3/$2.$3 last;

  location /images/ {
    rewrite ^(/images)/(.*)\.(png|jpg|gif)$ $1/$3/$2.$3 break;
  }
}
```

Передача дополнительных аргументов в командной строке - одна из целей, преследуемых при создании правил переписывания. Однако если исходные аргументы следует отбросить, а использовать только те, что определены в правиле, то в конце списка новых аргументов нужно поставить знак ?.

```
rewrite ^/images/(.*)_(\d+)x(\d+)\.(png|jpg|gif)$ /resizer/$1.$4?width =
$2&height=$3? last;
```

Преобразование правил из формата Apache

Созданию правил переписывания для наделенного развитой функциональностью модуля Apache `mod_rewrite` посвящено немало ресурсов в Интернете. Для преобразования правил, записанных в формате Apache, в формат, понятный NGINX, нужно придерживаться нескольких простых рекомендаций.

Рекомендация 1: заменить проверки существования каталогов и файлов директивой try_files

Встретив правило переписывания для Apache такого вида:

```
RewriteCond %{REQUEST_FILENAME} !-f
RewriteCond %{REQUEST_FILENAME} !-d
RewriteRule ^(.*)$ index.php?q=$1 [L]
```

преобразуйте его в такой фрагмент конфигурации NGINX:

```
try_files $uri $uri/ /index.php?q=$uri;
```

Эти правила говорят, что в случае, когда заданному в URI имени не соответствует ни файл, ни каталог на диске, запрос следует переадресовать файлу `index.php` в корне текущего контекста, передав в аргументе `q` исходный URI-адрес.

Пока в NGINX не появилась директива `try_files`, не было никакого другого выбора, кроме как проверять существование файла или каталога:

```
if (!-e $request_filename) {
    rewrite ^/(.*)$ /index.php?q=$1 last;
}
```

Не делайте этого. В Интернете можно встретить советы поступать именно таким образом, по они либо были даны очень давно, либо скопированы из устаревших конфигурационных файлов. Хотя директива `try_files` принадлежит базовому модулю `http` и, строго говоря, не может считаться правилом переписывания, указанную задачу она решает гораздо эффективнее, так как именно для этого и создавалась.

Рекомендация 2: заменить сравнение с REQUEST_URI секцией location

В Apache правила переписывания часто помещают в файлы `.htaccess`, так как исторически сложилось, что пользователи должны были иметь доступ к этим файлам. Типичный администратор коллективного хостинга вряд ли даст пользователю прямой доступ к конфигурационному контексту виртуального хоста, описывающего его сайт, по зато позволит помещать практически любую информацию в файл `.htaccess`. Это и привело к нынешней ситуации, когда файлы `.htaccess` забиты правилами переписывания.

Хотя в Apache тоже имеется директива `Location`, она редко используется для сопоставления с URI, потому что может встречаться либо в конфигурации главного сервера, либо в конфигурации виртуального хоста и нигде более. Поэтому-то мы то и дело натыкаемся на правила переписывания, в которых производится сравнение с REQUEST_URI:

```
RewriteCond %{REQUEST_URI} ^/niceurl
RewriteRule ^(.*)$ /index.php?q=$1 [L]
```

В NGINX для этой цели лучше использовать местоположение:

```
location /niceurl {
  include fastcgi_params;
  fastcgi_index index.php;
  fastcgi_pass 127.0.0.1:9000;
}
```

Разумеется, директивы в контексте `location` всецело зависят от вашего сайта, но принцип остается в силе - сопоставление с URI лучше производить с помощью местоположений.

То же относится и к директивам `RewriteRule` с неявным `REQUEST URI`. Как правило, они просто преобразуют URI из старого формата в новый. Так, в примере ниже `show.do` больше не нужен:

```
RewriteRule ^/controller/show.do$ http://example.com/controller [L,R=301]
```

Такое правило транслируется в следующий фрагмент конфигурации NGINX:

```
location = /controller/show.do {
    rewrite ^ http://example.com/controller permanent;
}
```

Чтобы не создавать слишком много местоположений при преобразовании директив `RewriteRule`, следует помнить, что регулярные выражения переносятся без изменений.

Рекомендация 3: заменить сравнение с HTTP_ HOST секцией server

Эта рекомендация тесно связана с предыдущей; мы рассматриваем конфигурации, в которых нужно добавить либо удалить из доменного имени часть `www`. Такие правила переписывания часто встречаются в файлах `.htaccess` или и виртуальных хостах с перегруженными `ServerAlias`:

```
RewriteCond %{HTTP_HOST} !^www
RewriteRule ^(.*)$ http://www.example.com/$1 [L,R=301]
```

Здесь мы транслируем URL без `www` в начале доменного имени в URL, содержащий `www`;

```
server {
  server_name example.com;
  rewrite ^ http://www.example.com$request_uri permanent;
}
```

В противоположном случае, когда требуется убрать `www`, мы встречаемся с таким правилом:

```
RewriteCond %{HTTP_HOST} ^www
RewriteRule ^(.*)$ http://example.com/$1 [L,R=301]
```

Оно транслируется в такую конфигурацию NGINX:

```
server {
  server_name www.example.com;
  rewrite ^ http://example.com$request_uri permanent;
}
```

Контекст `server` для переадресованного варианта не показан, потому что к переписыванию он отношения не имеет.

Тот же принцип применим и в случаях, когда нужно не просто распознать наличие или отсутствие `www`. Его можно использовать всегда, когда в условии `RewriteCond` встречается `%{HTTP_HOST}`. В NGINX для этого лучше заводить контексты `server`, по одному для каждого проверяемого условия.

Пусть, например, имеется такая конфигурация Apache с несколькими сайтами:

```
RewriteCond %{HTTP_HOST} ^site1
RewriteRule ^(.*)$ /site1/$1 [L]
RewriteCond %{HTTP_HOST} ^site2
RewriteRule ^(.*)$ /site2/$1 [L]
RewriteCond %{HTTP_HOST} ^site3
RewriteRule ^(.*)$ /site3/$1 [L]
```

Она транслируется в конфигурацию, где производится сравнение с именем хоста и для каждого хоста задается своя директива `root`.

```
server {
  server_name site1.example.com;
root /home/www/site1;
  }
  server {
      server_name site2.example.com;
      root /home/www/site2;
  }
  server {
     server_name site3.example.com; root /
     home/www/site3;
  }
```

По существу, это виртуальные хосты, так что и описывать их в конфигурационном файле следует соответственно.

Рекомендация 4: заменить RewriteCond проверкой переменной в директиве if

Эта рекомендация применяется после рекомендаций 1-3. Если после их применения остались еще какие-то условия, то для проверки переменных можно использовать директиву `if`. Любой HTTP-переменной соответствует переменная NGINX, имя которой образовано путем перевода имени исходной переменной в нижний регистр и добавления префикса `$http_`. Дефисы (-) при этом следует заменить знаками подчеркивания (_).

Следующий пример (взятый из документации Apache по модулю `mod_rewrite` на странице `http://httpd.apache.org/docs/2.2/mod/mod_rewrite.html`) позволяет вернуть пользователю ту или иную страницу в зависимости от заголовка `User-Agent`:

```
RewriteCond %{HTTP_USER_AGENT} ^Mozilla
RewriteRule ^/$ /homepage.max.html [L]
RewriteCond %{HTTP_USER_AGENT} ^Lynx
RewriteRule ^/$ /homepage.min.html [L]
RewriteRule ^/$ /homepage.std.html [L]
```

Эквивалентная конфигурация NGINX выглядит следующим образом:

```
if ($http_user_agent ~* ^Mozilla) {
    rewrite ^/$ /homepage.max.html break;
}
if ($http_user_agent ~* ^Lynx) {
    rewrite ^/$ /homepage.min.html break;
}
index homepage.std.html;
```

Конечно, специальные переменные, доступные только в модуле Apache `mod_rewrite`, в NGINX проверить невозможно.

Резюме

В этом приложении мы рассмотрели модуль NGINX `rewrite`. С ним ассоциировано лишь несколько директив, но это не мешает создавать достаточно сложные конфигурации. Хочется надеяться, что пошаговое рассмотрение процедуры создания новых правил убедило вас, что это не так уж трудно. Но для написания сколько-нибудь сложных правил необходимо уверенное владение регулярными выражениями. В конце мы обсудили, как преобразовать правила переписывания из формата Apache в конфигурацию, попятную NGINX. Попутно мы обнаружили, что многие задачи, которые в Apache решаются с помощью правил переписывания, в NGINX можно решить по-другому.

Приложение C. Сообщество NGINX

Ныне NGINX поддерживается не только полным жизни сообществом, но и компанией. Игорь Сысоев, автор NGINX, в 2011 году стал одним из учредителей компании NGINX, Inc., которая предлагает профессиональную поддержку организациям, использующим NGINX. Но и он сам, и другие разработчики NGINX не отдалились от сообщества. В этом приложении приводится краткий обзор общедоступных сетевых ресурсов. Рассматриваются следующие вопросы.

- Список рассылки.
- IRC-канал.
- Веб-ресурсы.
- Как правильно составить отчет об ошибке.

Список рассылки

Список рассылки по адресу `nginx@nginx.org` поддерживается с 2005 года. Чтобы понять, как эффективнее всего поставить этот список себе на пользу, подпишитесь на него и посмотрите, какие в нем задают вопросы и как на них отвечают. Прежде чем задавать вопрос, поищите ответ в Сети. Имеется также список ЧАВО по адресу http://wiki.nginx.org/Faq. Покопайтесь в архивах на сайте http://mailman.nginx.org/pipermail/nginx/ - быть может, кто-то недавно задавал аналогичный вопрос. Повторять недавно рассматривавшийся вопрос зазорно, к тому же это раздражает читателей списка рассылки.

IRC-канал

IRC-канал #nginx на сайте `irc.freenode.net` - ресурс для тех, кто хочет познакомиться с разработчиками и немедленно получить полезный ответ на короткий вопрос. Но при посещении канала со-

блюдайте этикет IRC. Длинные куски текста - конфигурационные файлы или вывод компилятора - следует размещать на каком-нибудь стороннем сайте, а в канале оставлять только URL. Дополнительные сведения об этом канале можно найти по адресу http://wiki.nginx.org/IRC.

Веб-ресурсы

Вики по адресу http://wiki.nginx.org уже много лет является полезным ресурсом. Здесь вы найдете полный справочник по директивам, список модулей и много примеров конфигураций. Однако помните, что это вики, поэтому информация не обязательно точна, актуальна или отвечает вашей конкретной ситуации. В этой книге мы неоднократно видели, что прежде чем предлагать решение, очень важно ясно понять, чего мы хотим достичь.

Компания NGINX, Inc. публикует официальную справочную документацию по адресу http://nginx.org/en/docs/. Там же имеется краткое введение в NGINX, различные пособия и описания всех модулей и директив.

Как правильно составить отчет об ошибке

Обращаясь за помощью, нужно знать, как правильно составить отчет об ошибке. Ответ придет гораздо быстрее, если вы сможете описать проблему ясно и так, чтобы ее можно было воспроизвести. В этом разделе мы покажем, как это сделать.

Самое трудное в отчете об ошибке - определить, в чем состоит проблема. Сначала подумайте, чего вы добиваетесь. Формулируйте задачу четко и кратко, например:

```
Я хочу, чтобы все запросы к имени subdomain.example.com
обслуживались сервером server 1.
```

Не пишите в таком духе:

```
Когда я обращаюсь к subdomain.example.com, все запросы обслуживаются из
локальной файловой системы, а не проксируются на server1.
```

Улавливаете разницу? В первом случае вы четко формулируете цель, а во втором описываете результат, а не исходную задачу.

Определив, в чем проблема, опишите, как ее воспроизвести:

```
Обращение к http://subdomain.example.com/serverstatus дает
"404 File Not Found".
```

Теперь всякий, кто займется вашей проблемой, будет знать, на что смотреть. И как доказать, что проблема решена - не работавший пример должен заработать.

Далее полезно описать, в какой среде наблюдалась проблема. Некоторые ошибки проявляются только в определенных операционных системах и при компоновке с конкретной версией той или иной библиотеки.

В отчет следует также включить все конфигурационные файлы, необходимые для воспроизведения проблемы. Если файл имеется в каком-нибудь архиве ПО, достаточно ссылки на него.

Прежде чем отправлять отчет об ошибке, перечитайте его. Часто обнаруживается, что какой-то информации не хватает. А иногда оказывается, что для решения проблемы достаточно было всего лишь четко сформулировать ее.

Резюме

В этом приложении мы рассказали о сообществе, сложившемся вокруг NGINX. Мы познакомились с основными игроками и с доступными сетевыми ресурсами. Мы также узнали, как правильно составить отчет об ошибке, который поможет найти решение проблемы.

Приложение D. Сохранение сетевых настроек в Solaris

В главе 8 «Техника устранения неполадок» мы видели, как изменить различные настройки сети в разных операционных системах. В этом приложении описываются детали, имеющие отношение к ОС Solaris версии 10 и выше.

Следующий скрипт запускается каркасом **Service Management Framework (SMF)**, чтобы установить сетевые параметры с помощью программы `ndd`. Сохраните его в файле `/lib/svc/method/network-tuning.sh` и сделайте исполняемым.

```
# vi /lib/svc/method/network-tuning.sh
```

Ниже приведено содержимое файла `/lib/svc/method/network-tuning.sh`:

```
#!/sbin/sh
# Set the following values as desired
ndd -set /dev/tcp tcp_max_buf 16777216
ndd -set /dev/tcp tcp_smallest_anon_port 1024
ndd -set /dev/tcp tcp_largest_anon_port 65535
ndd -set /dev/tcp tcp_conn_req_max_q 1024
ndd -set /dev/tcp tcp_conn_req_max_q0 4096
ndd -set /dev/tcp tcp_xmit_hiwat 1048576
ndd -set /dev/tcp tcp_recv_hiwat 1048576
# chmod 755 /lib/svc/method/network-tuning.sh
```

В приведенном ниже манифесте определена служба настройки сети и сказано, что скрипт нужно выполнить на этапе загрузки. Обратите внимание на атрибут ***transient***, который говорит SMF, что это запускаемый однократно скрипт, а не постоянно работающий демон.

```
<?xml version="1.0"?>
<!DOCTYPE service_bundle SYSTEM "/usr/share/lib/xml/dtd/service_bundle.dtd.1">
<service_bundle type='manifest' name='SUNW:network_tuning'>
  <service
     name='site/network_tuning'
     type='service'
     version='1'>
    <create_default_instance enabled='true' />
    <single_instance />
    <dependency
       name='usr'
       type='service'
       grouping='require_all'
       restart_on='none'>
      <service_fmri value='svc:/system/filesystem/minimal' />
    </dependency>

    <!-- Run ndd commands after network/physical is plumbed. -->
    <dependency
       name='network-physical'
       grouping='require_all'
       restart_on='none'
       type='service'>
      <service_fmri value='svc:/network/physical' />
    </dependency>

    <!-- but run the commands before network/initial -->
    <dependent
       name='ndd_network-initial'
       grouping='optional_all'
       restart_on='none'>
      <service_fmri value='svc:/network/initial' />
    </dependent>
    <exec_method
       type='method'
       name='start'
       exec='/lib/svc/method/network-tuning.sh'
       timeout_seconds='60' />
    <exec_method
       type='method'
       name='stop'
       exec=':true'
       timeout_seconds='60' />
    <property_group name='startd' type='framework'>
      <propval name='duration' type='astring'
    value='transient' />
    </property_group>
    <stability value='Unstable' />
    <template>
```

```
      <common_name>
        <loctext xml:lang='C'>
          Network Tunings
        </loctext>
      </common_name>
    </template>
  </service>
</service_bundle>
```

Сохраните этот код в файле `/var/svc/manifest/site/network-tuning.xml` и импортируйте его такой командой:

```
# svccfg import /var/svc/manifest/site/network-tuning.xml
```

Служба намеренно сделана простой, так как предназначена только для демонстрации. Интересующийся читатель может узнать о SMF из страниц руководства в Solaris и из сетевых ресурсов.

Предметный указатель

A

B

C

D

E

F

G

H

I

K

L

M

N

O

P

R

S

T

U

V

W

X

Y

ж

з

и

к

P

C

т

у

х

ц

ш

УДК 004.738.5:004.42Nginx
ББК 32.973.202
А36

Айвалиотис Д.

А36 Администрирование сервера NGINX. - М.: ДМК Пресс, 2013. - 288 с.: ил.

ISBN 978-5-94074-961-5

NGINX — это высокопроизводительный сервер, который реализует функции прокси для веб-серверов и почтовых серверов и потребляет очень мало системных ресурсов. В Интернете хватает руководств по его настройке и примеров конфигураций, но при этом трудно понять, как правильно настроить NGINX для конкретных нужд.

Эта книга расчистит мутные воды конфигурирования NGINX и научит вас настраивать его для решения различных задач. Попутно вы узнаете, что означают некоторые покрытые мраком параметры, и поймете как разработать конфигурацию, отвечающую именно вашим целям.

Вначале дается краткий обзор процедуры компиляции NGINX и описывается формат конфигурационного файла. Затем автор переходит к модулям и рассказывает о многочисленных настройках, позволяющих использовать NGINX в качестве обратного прокси-сервера. Завершается книга обсуждением поиска и устранения неполадок.

Издание предназначено для системных администраторов или инженеров, имеющих опыт эксплуатации веб-серверов.

УДК 004.738.5:004.42Nginx
ББК 32.973.202

ISBN 978-1-84951-744-7 (анг.)
ISBN 978-5-94074-961-5 (рус.)