Основная функциональность

Пример конфигурации

```
user www www;
worker_processes 2;
```

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

events {
    use kqueue;
    worker_connections 2048;
}

...
```

Директивы

Синтаксис: **`accept_mutex`** `on | off;`
Умолчание: `accept_mutex off;`
Контекст: `events`

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

Нет необходимости включать `accept_mutex` на системах, поддерживающих флаг EPOLLEXCLUSIVE (1.11.3), или при использовании reuseport.

До версии 1.11.3 по умолчанию использовалось значение `on`.

Синтаксис: **`accept_mutex_delay`** *`время`*`;`
Умолчание: `accept_mutex_delay 500ms;`
Контекст: `events`

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

Синтаксис: **`daemon`** `on | off;`
Умолчание: `daemon on;`
Контекст: `main`

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

Синтаксис: **debug_connection** *адрес* | *CIDR* | unix:;
Умолчание: —
Контекст: events

Включает отладочный лог для отдельных клиентских соединений. Для остальных соединений используется уровень лога, заданный директивой error_log. Отлаживаемые соединения задаются IPv4 или IPv6 (1.3.0, 1.2.1) адресом или сетью. Соединение может быть также задано при помощи имени хоста. Отладочный лог для соединений через UNIX-сокеты (1.3.0, 1.2.1) включается параметром “ `unix:` ”.

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

Для работы директивы необходимо сконфигурировать nginx с параметром `--with-debug` , см. “Отладочный лог”.

Синтаксис: **debug_points** abort | stop;
Умолчание: —
Контекст: main

Эта директива используется для отладки.

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

Синтаксис: **env** *переменная*[=*значение*];
Умолчание: env TZ;
Контекст: main

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

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

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

Пример использования:

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

Переменная окружения NGINX используется для внутренних целей nginx и не должна устанавливаться непосредственно самим пользователем.

Синтаксис: **`error_log`** *`файл`* `[`*`уровень`*`] [json];`
Умолчание: `error_log logs/error.log error;`
Контекст: `main`, `http`, `mail`, `stream`, `server`, `location`

Конфигурирует запись в лог. На одном уровне конфигурации может использоваться несколько логов (1.5.2). Если на уровне конфигурации `main` запись лога в файл явно не задана, то используется файл по умолчанию.

Первый параметр задаёт *`файл`*, который будет хранить лог. Специальное значение `stderr` выбирает стандартный файл ошибок. Запись в syslog настраивается указанием префикса "`syslog:`". Запись в кольцевой буфер в памяти

настраивается указанием префикса “`memory:`” и *`размера`* буфера и как правило используется для отладки (1.7.11).

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

Для работы уровня лога `debug` необходимо сконфигурировать nginx с `--with-debug`, см. “Отладочный лог”.

Параметр `json` (1.29.8) включает запись лог-файла ошибок в формате JSON, с возможностью использования контекстных тегов:

```
{
  "level": "error",
  "timestamp": "2026-05-13T10:30:15.042+00:00",
  "pid": 12345,  "tid": 12345,  "cnum": 3,
  "msg": "connect() failed",
  "client": "192.168.1.10",  "server": "example.com",
  "request": "GET /api HTTP/1.1",
  "upstream": "http://127.0.0.1:8080/api",
  "errno": 111,
  "errtext": "Connection refused"
}
```

Размер строки не может превышать 2KB, данные сверх этого ограничения сокращаются до `“truncated”:1`. Уровень лога `debug` не поддерживается для JSON.

Параметр доступен как часть коммерческой подписки.

Директива может быть указана на уровне `stream` начиная с версии 1.7.11 и на уровне `mail` начиная с версии 1.9.0.

Синтаксис: **`events`** `{ ... }`
Умолчание: —
Контекст: `main`

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

Синтаксис: **include** *файл* | *маска*;
Умолчание: —
Контекст: любой

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

Пример использования:

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

Синтаксис: **load_module** *файл*;
Умолчание: —
Контекст: main

Эта директива появилась в версии 1.9.11.

Загружает динамический модуль.

Пример:

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

Синтаксис: **lock_file** *файл*;
Умолчание: lock_file logs/nginx.lock;
Контекст: main

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

Синтаксис: **master_process** on | off;
Умолчание: master_process on;
Контекст: main

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

Синтаксис: **multi_accept** on | off;
Умолчание: multi_accept off;
Контекст: events

Если `multi_accept` выключен, рабочий процесс за один раз будет принимать только одно новое соединение. В противном случае рабочий процесс за один раз будет принимать сразу все новые соединения.

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

Синтаксис: **pcre_jit** on | off;
Умолчание: pcre_jit off;
Контекст: main

Эта директива появилась в версии 1.1.12.

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

Использование PCRE JIT способно существенно ускорить обработку регулярных выражений.

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

Синтаксис: **pid** *файл*;
Умолчание: pid logs/nginx.pid;
Контекст: main

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

Синтаксис: **ssl_engine** *устройство*;
Умолчание: —

Контекст: `main`

Задаёт название аппаратного SSL-акселератора.

При тестировании конфигурации библиотека OpenSSL может загрузить динамический модуль.

Синтаксис: **`ssl_object_cache_inheritable`** `on | off;`
Умолчание: `ssl_object_cache_inheritable on;`
Контекст: `main`

Эта директива появилась в версии 1.27.4.

Если включено, то SSL-объекты (SSL-сертификаты, секретные ключи, доверенные сертификаты, отозванные сертификаты (CRL)), будут наследоваться при перезагрузках конфигурации.

SSL-объекты, загруженные из файла, наследуются, если время изменения и файловый индекс не изменились с момента последней перезагрузки конфигурации. Секретные ключи, указанные как `engine:name:id`, никогда не наследуются. Секретные ключи, указанные как `data:literal`, всегда наследуются.

SSL-объекты, загруженные из переменных, не могут наследоваться.

Пример:

```
ssl_object_cache_inheritable on;

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

Синтаксис: **`thread_pool`** *имя* `threads=`*число* `[max_queue=`*число*`];`
Умолчание: `thread_pool default threads=32 max_queue=65536;`
Контекст: `main`

Эта директива появилась в версии 1.7.11.

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

Параметр `threads` задаёт число потоков в пуле.

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

Синтаксис: **timer_resolution** *интервал*;
Умолчание: —
Контекст: main

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

Пример использования:

```
timer_resolution 100ms;
```

Внутренняя реализация интервала зависит от используемого метода:

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

Синтаксис: **use** *метод*;
Умолчание: —
Контекст: events

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

Синтаксис: **user** *пользователь* [*группа*];

Умолчание: `user nobody nobody;`
Контекст: `main`

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

Синтаксис: **`worker_aio_requests`** *`число`*`;`
Умолчание: `worker_aio_requests 32;`
Контекст: `events`

Эта директива появилась в версиях 1.1.4 и 1.0.7.

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

Синтаксис: **`worker_connections`** *`число`*`;`
Умолчание: `worker_connections 512;`
Контекст: `events`

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

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

Синтаксис: **`worker_cpu_affinity`** *`маска_CPU`* `...;`
`worker_cpu_affinity` `auto [`*`маска_CPU`*`];`
Умолчание: —
Контекст: `main`

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

Например,

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

привязывает каждый рабочий процесс к отдельному процессору, тогда как

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

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

Специальное значение `auto` (1.9.10) позволяет автоматически привязать рабочие процессы к доступным процессорам:

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

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

```
worker_cpu_affinity auto 01010101;
```

Директива доступна только на FreeBSD и Linux.

Синтаксис: **`worker_priority`** *`число`*`;`
Умолчание: `worker_priority 0;`
Контекст: `main`

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

Пример использования:

```
worker_priority -10;
```

Синтаксис: **`worker_processes`** *`число`* `| auto;`
Умолчание: `worker_processes 1;`
Контекст: `main`

Задаёт число рабочих процессов.

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

> Параметр `auto` поддерживается только начиная с версий 1.3.8 и 1.2.5.

Синтаксис: **`worker_rlimit_core`** *`размер`*`;`
Умолчание: —
Контекст: `main`

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

Синтаксис: **`worker_rlimit_nofile`** *`число`*`;`
Умолчание: —
Контекст: `main`

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

Синтаксис: **`worker_shutdown_timeout`** *`время`*`;`
Умолчание: —
Контекст: `main`

Эта директива появилась в версии 1.11.11.

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

Синтаксис: **`working_directory`** *`каталог`*`;`
Умолчание: —
Контекст: `main`

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