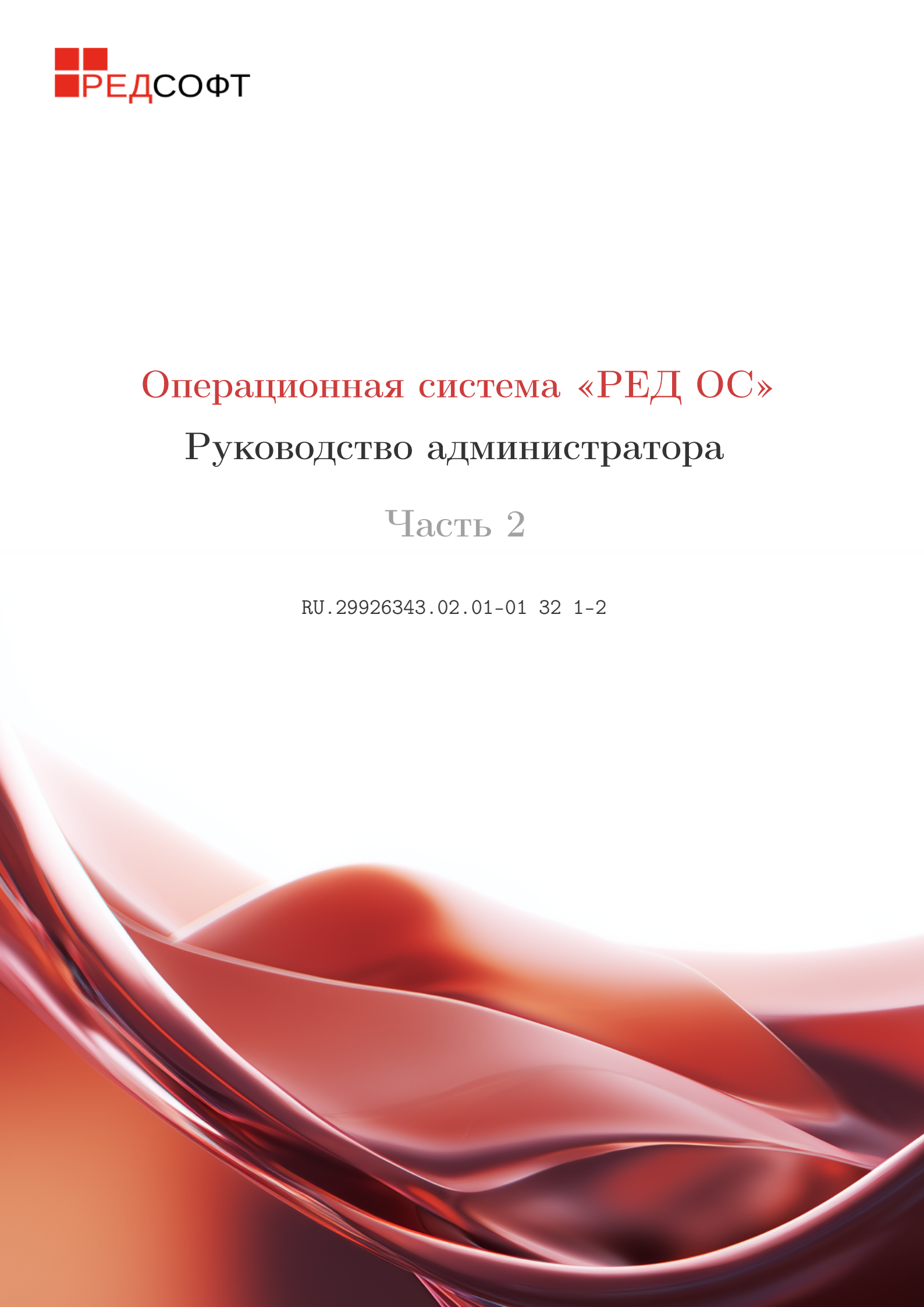
РЕДСОФТ

Операционная система «РЕД ОС»

Руководство администратора

Часть 2

RU.29926343.02.01-01 32 1-2



Оглавление

Настройка веб-серверов

https://redos.red-soft.ru/

Настоящий документ является второй частью руководства администратора операционной системы «РЕД ОС» версии 8 (далее РЕД ОС, ОС) для процессоров архитектур `x86_64`. В данной части руководства администратора рассматриваются и описываются действия по администрированию веб-серверов Apache и NGINX, а также сервера веб-приложений Tomcat.

В данной части руководства администратора рассмотрены следующие ключевые моменты:

- процесс установки и настройки веб-серверов;
- формат используемых в настройке файлов конфигурации;
- основные директивы и их параметры, необходимые для конфигурирования веб-серверов;
- возможность дополнительной настройки веб-серверов для использования их в качестве прокси-сервера в случае с Apache и в качестве обратного прокси-сервера в случае с NGINX;
- настройка обеспечения высокой доступности и распределения нагрузки на веб-серверы;
- а также основные принципы журналирования событий веб-серверов.

Руководство администратора часть 2 предназначено для системных администраторов или инженеров, имеющих опыт эксплуатации веб-серверов.

Октябрь 2025

Настройка веб-серверов

1. Веб-сервер Apache

1.1 Общие сведения

Веб-сервер Apache – это кроссплатформенное программное обеспечение для размещения и поддержки веб-сервера.

Главными компонентами веб-сервера Apache являются ядро, модули и файлы конфигурации.

Ядро сервера выполняет следующие основные функции:

- обработка конфигурационных файлов;
- обеспечение работы по протоколам `HTTP` и `HTTPS`;
- дополнение функциональных возможностей с использованием системы загрузки модулей и пр.

Модули отвечают за определенные возможности программы, такие как поддержка различных языков программирования, повышение безопасности, аутентификация пользователя, исправление ошибок и т. д. Каждый модуль отвечает за отдельный компонент работы с запросом.

Конфигурационный файл – это файл, который хранит настройки операционной системы и приложений, а также позволяет вносить в них изменения.

Работа Apache построена на процессной модели. В рамках процессной модели каждое соединение (обработка запроса) с сервером помещается в отдельный поток и проходит определенные этапы. При таком подходе несколько соединений невозможно обработать параллельно. Сервер обработает запрос от второго пользователя только тогда, когда будет обработан запрос от первого пользователя, а запрос от третьего пользователя будет обработан только тогда, когда будет обработан запрос от второго пользователя. Новое соединение не может начаться, пока предыдущая операция не завершится и не освободит поток.

1.2 Установка Apache

Установка веб-сервера Apache производится командой:

```
dnf install httpd
```

После установки запустите и добавьте в автозагрузку службу `httpd`:

```
systemctl enable --now httpd
```

Проверьте статус службы:

```
systemctl status httpd
```

В статусе должно отображаться `active (running)`.

1.3 Файл конфигурации

Файл конфигурации веб-сервера Apache по умолчанию называется `httpd.conf` и размещается в каталоге `/etc/httpd/conf`.

Значения переменных, хранящихся в конфигурационном файле, называются *директивами*. Существует большое подмножество директив, которые называются основными и устанавливаются по умолчанию. Другие директивы распознаются в зависимости от того, какой набор модулей подключен в процессе подготовки конкретного сервера Apache.

1.3.1 Синтаксис файла конфигурации

Файл конфигурации `httpd` содержит по одной директиве в каждой строке. Обратная косая черта \ может использоваться в качестве последнего символа в строке, чтобы указать, что директива переходит на следующую строку. Между обратной косой чертой и завершением строки не должно быть других символов или пробелов.

Аргумент директивы от самой директивы отделяется пробелом. Если аргумент содержит пробелы, необходимо заключить этот аргумент в кавычки.

Формат файла конфигурации имеет следующий вид:

```
<директива> <аргумент>
```

Директивы в файле конфигурации нечувствительны к регистру, но аргументы директив часто чувствительны к регистру. Строки, начинающиеся с символа решетки #, считаются комментариями и игнорируются. Комментарии не могут быть включены в одну строку с директивой конфигурации. Пробелы, добавленные перед директивой, игнорируются, поэтому для удобства чтения директивы допускается указывать с отступом. Пустые строки также игнорируются.

Для проверки файла конфигурации на наличие синтаксических ошибок без запуска сервера, используется команда:

```
httpd -t
```

1.3.2 Модули

Сервер Apache разработан таким образом, что всегда существует возможность варьирования основных функциональных возможностей. Для расширения функционала веб-сервера доступны специальные субсекции – *модули*. Модули существуют двух типов – статические и динамические.

Статические модули уже добавлены в ядро и вызываются использованием соответствующих директив в файле конфигурации.

Динамические модули можно добавить в любое время с помощью директив `AddModule` и `ClearModuleList`. В случае разделяемых объектных файлов модулями можно динамически управлять с помощью директивы `LoadModule`.

Большое значение имеет порядок загрузки модулей. Возможность управлять порядком загрузки модулей (и, вероятно, исключить некоторые из модулей, загружаемых по умолчанию) реализуется с помощью директивы `ClearModuleList`. После этого необходимо указать список загружаемых модулей последовательностью директив `AddModule`:

```
AddModule mod_access.c
```

Если модуль включен, он становится составной частью исполняемого процесса `httpd` с тем же идентификатором процесса и доступом к тем же системным ресурсам.

Для получения перечня перечня подключенных модулей используется команда:

```
httpd -l
```

1.4 Управление конфигурацией

1.4.1 Управление через системную службу

После настройки веб-сервера Apache необходимо запустить и добавить в автозагрузку службу `httpd`:

```
systemctl enable --now httpd
```

Для проверки статуса службы используется команда:

```
systemctl status httpd
```

При успешном запуске службы в статусе должно быть отображено `active (running)`.

Для переопределения конфигурации, например, после внесения в файл конфигурации каких-либо изменений, необходимо перезапустить службу `httpd`:

```
systemctl reload httpd
```

Для остановки работы службы `httpd` необходимо выполнить:

```
systemctl stop httpd
```

1.4.2 Управление с помощью сигналов

Для управления конфигурацией `HTTP`-сервера Apache также допускается отправка сигналов запущенным процессам `httpd`.

Доступно два способа отправки сигналов:
- с помощью утилиты `kill`;
- с помощью сценария управления `apachectl`.

Управление конфигурацией с помощью утилиты `kill`

Для управления конфигурацией веб-сервера можно использовать утилиту `kill`. В системе может быть запущено большое количество дочерних процессов `httpd`, однако отправлять сигналы необходимо только родительскому (главному) процессу. Идентификатор (`PID`) главного процесса по умолчанию записывается в файл `httpd.pid`, который находится в каталоге `/var/run`.

Главный процесс поддерживает следующие сигналы:
- `TERM` – быстрое завершение работы дочерних процессов, затем главного процесса, обслуживание текущих запросов прекращается, новые запросы не обрабатываются;
- `USR1` – ожидание завершения обслуживания текущих запросов и корректное завершение работы дочерних процессов, переопределение файла конфигурации, повторное открытые файлов журналов, запуск новых дочерних процессов;
- `HUP` – быстрое завершение работы дочерних процессов, обслуживание текущих запросов прекращается, новые запросы не обрабатываются, переопределение файла конфигурации, повторное открытые файлов журналов, запуск новых дочерних процессов;
- `WINCH` – постепенное завершение рабочих процессов.

Для отправки сигнала родительскому процессу выполните команду вида:

```
kill <СИГНАЛ> <PID_главного_процесса>
```

Например:

```
kill -TERM '/var/run/httpd/httpd.pid'
```

Управление конфигурацией с помощью `httpd`

Управление конфигурацией `httpd` можно производить отправкой соответствующего сигнала главному процессу. Сигнал можно отправить, выполнив команду вида:

```
httpd -k <СИГНАЛ>
```

где `<СИГНАЛ>` может принимать следующие значения:
- `stop` – быстрое завершение работы дочерних процессов, затем главного процесса, обслуживание текущих запросов прекращается, новые запросы не обрабатываются (аналогичен сигналу `TERM`);
- `graceful` – ожидание завершения обслуживания текущих запросов и корректное завершение работы дочерних процессов, переопределение файла конфигурации, повторное открытые файлов журналов, запуск новых дочерних процессов (аналогичен сигналу `USR1`);
- `restart` – быстрое завершение работы дочерних процессов, обслуживание текущих запросов прекращается, новые запросы не обрабатываются, переопределение файла конфигурации, повторное открытые файлов журналов, запуск новых дочерних процессов (аналогичен сигналу `HUP`);

- `graceful-stop` – постепенное завершение рабочих процессов (аналогичен сигналу `WINCH`).

После отправки сигнала `httpd` подробную информацию о ходе его обработки можно просмотреть, выполнив команду:

```
tail -f /var/log/httpd/error_log
```

1.5 Настройка Apache в качестве веб-сервера

Адреса в Интернете записываются с помощью `URL` (унифицированный указатель ресурса), который указывает на используемый протокол (например, `http`), имя сервера (например, `www.apache.org`), `URL`-путь (например, `/docs/current/getting-started.html`) и, возможно, строку запроса (например, `?arg=value`), используемую для передачи серверу дополнительных аргументов.

Клиент (например, веб-браузер) подключается к серверу (например, `HTTP`-серверу Apache), используя определённый протокол, и отправляет запрос на ресурс, используя `URL`-путь.

`URL`-путь может представлять файл, обработчик или файл какой-либо программы. Содержимое сайта может принимать различные формы, но в широком смысле разделяется на статический и динамический контент.

Статический контент – это, например, `HTML`-файлы, файлы изображений, `CSS`-файлы и другие файлы, которые расположены на диске. Директива `DocumentRoot` указывает, где в файловой системе, должны быть размещены данные файлы. Директива устанавливается глобально или отдельно для каждого виртуального хоста.

Динамический контент – это всё, что генерируется во время запроса и может изменяться от запроса к запросу. Существует множество способов создания динамического контента. Для генерации содержимого доступны различные обработчики. Также могут быть написаны специальные `CGI`-программы для генерации контента на сайте.

После обработки запроса сервер отправляет ответ, содержащий код состояния и тело ответа (опционально). Код состояния указывает, успешно ли обработан запрос. Если нет, то какая ошибка произошла. Код ошибки сообщает клиенту, что ему необходимо делать далее.

Детали транзакции и условия возникновения ошибки записываются в файлы журналов.

Далее будут рассмотрены основные директивы, используемые для настройки Apache в качестве веб-сервера.

1.5.1 Директива `Listen`

Определяет, на каких `IP`-адресах и портах Apache будет прослушивать входящие запросы. По умолчанию запросы прослушиваются на всех `IP`-интерфейсах.

Несколько директив `Listen` могут использоваться для указания количества адресов и портов для прослушивания. Сервер будет отвечать на запросы с любого из перечисленных адресов и портов.

Синтаксис:

```
Listen [<адрес>] <порт>
```

где:

- `<адрес>` – IP-адрес, на котором сервер будет слушать. Если этот параметр не указан, Apache будет слушать на всех доступных интерфейсах (`0.0.0.0`);
- `<порт>` – номер порта, на котором сервер будет принимать соединения. По умолчанию используется `80` для `HTTP` и `443` для `HTTPS`.

Например, для подключения к серверу по порту `80` в файле конфигурации необходимо указать:

```
Listen 80
```

Для подключения к серверу по нескольким интерфейсам в файле конфигурации необходимо указать:

```
Listen 192.170.2.1:80
Listen 192.170.2.5:8000
```

1.5.2 Директива `SernerName`

Используется для указания основного имени хоста, под которым сервер будет доступен.

При отсутствии соответствующего `ServerName`, Apache может использовать директиву `ServerAlias` для обработки альтернативных имен.

Синтаксис:

```
ServerName <имя_сервера>[:<порт>]
```

где:

- `<имя_сервера>` – полное доменное имя (`FQDN`) или IP-адрес, который будет использоваться для доступа к серверу;
- `<порт>` (необязательный) – номер порта, на котором сервер будет слушать. По умолчанию используется порт `80` для `HTTP` и `443` для `HTTPS`.

Директива `ServerName` может использоваться в контексте виртуального хоста (`<VirtualHost>`), а также в основном конфигурационном файле сервера (`httpd.conf`).

Пример использования директивы в контексте виртуального сервера:

```
<VirtualHost *:80>
       ServerName www.example.com
       DocumentRoot /var/www/example
</VirtualHost>
```

Если запрос поступает с именем, соответствующим `ServerName`, Apache будет обрабатывать его в рамках этого виртуального хоста.

При отсутствии соответствующего `ServerName`, Apache может использовать директиву `ServerAlias` для обработки альтернативных имен.

В случае если не задано ни одного виртуального хоста, Apache будет использовать

значение ServerName из основной конфигурации как имя по умолчанию.

1.5.3 Директива <VirtualHost>

Используется для настройки виртуальных хостов, что позволяет одному серверу обслуживать несколько доменных имен или сайтов на одном IP-адресе. Применение данной директивы особенно полезно для хостинг-провайдеров и организаций, которым необходимо оптимизировать использование серверных ресурсов.

Синтаксис:

```
<VirtualHost [<IP-адрес>]:[<порт>]>
      <Директивы>
</VirtualHost>
```

Если <IP-адрес> не указан, виртуальный хост будет слушать на всех доступных интерфейсах.

Также доступна настройка нескольких виртуальных хостов в одном файле конфигурации, например:

```
<VirtualHost *:80>
    ServerName site1.com
    DocumentRoot /var/www/site1
</VirtualHost>

<VirtualHost *:80>
    ServerName site2.com
    DocumentRoot /var/www/site2
</VirtualHost>
```

1.5.4 Директива DocumentRoot

Указывает основную директорию, из которой сервер будет обслуживать файлы для запросов HTTP. Директива DocumentRoot может использоваться в основном конфигурационном файле Apache (httpd.conf) или внутри блоков <VirtualHost>, чтобы указать разные корневые директории для разных виртуальных хостов.

Синтаксис:

```
DocumentRoot "/<путь_к_директории>"
```

Путь должен быть абсолютным и указывать на директорию, в которой находятся файлы настраиваемого веб-сайта.

Например:

```
DocumentRoot "/var/www/html"
```

В этом примере Apache будет искать файлы для обслуживания в директории /var/www/html.

При использовании виртуальных хостов каждая конфигурация <VirtualHost> может иметь свою собственную директиву DocumentRoot, что позволяет на одном сервере размещать несколько сайтов с различными корневыми директориями.

Пример использования директивы в контексте виртуального сервера:

```
<VirtualHost *:80>
   ServerName example.com
   DocumentRoot "/var/www/example"
</VirtualHost>

<VirtualHost *:80>
   ServerName another-example.com
   DocumentRoot "/var/www/another-example"
</VirtualHost>
```

Важным аспектом является также правильная настройка прав доступа к директории, указанной в `DocumentRoot`. Apache должен иметь доступ к этой директории для чтения файлов. Однако не следует предоставлять доступ к системным файлам или директориям, которые не предназначены для публичного доступа.

Для настройки дополнительных параметров доступа к указанной директории рекомендуется использовать директиву `<Directory>`.

1.5.5 Директива `<Directory>`

Используется для настройки параметров доступа и поведения сервера для определенной директории на файловой системе. Позволяет управлять доступом к файлам и настройками для конкретных каталогов, что особенно полезно для повышения безопасности и настройки функциональности веб-приложений.

Синтаксис:

```
<Directory "</путь_к_директории>">
      <Директивы>
</Directory>
```

Путь должен быть абсолютным и указывать на директорию, для которой будут применяться настройки.

Пример использования директивы для запрета доступа к директории:

```
<Directory "/var/www/private">
    Options None
    AllowOverride None
    Require all denied
</Directory>
```

1.5.6 Директива `ServerRoot`

Определяет основной каталог, в котором находятся файлы конфигурации и другие ресурсы сервера.

Синтаксис:

```
ServerRoot "<путь_к_каталогу>"
```

где `<путь_к_каталогу>` – это абсолютный путь к каталогу, который будет использоваться в качестве корневого каталога для сервера, должен быть указан в кавычках.

Например:

```
ServerRoot "/etc/httpd"
```

1.5.7 Директива Include

Используется для включения других конфигурационных файлов в основной файл конфигурации сервера (`httpd.conf`).

Синтаксис:

```
Include "<путь_к_файлу_конфигурации>"
```

где `<путь_к_файлу_конфигурации>` – это путь к файлу или каталогу, который необходимо включить. Путь может быть абсолютным или относительным.

Для включения нескольких файлов конфигурации, находящихся в одном каталоге, при указании имен можно использовать шаблоны. Например:

```
Include /etc/httpd/conf.d/*.conf
```

Примечание. Порядок, в котором включаются файлы, имеет значение. Если один файл переопределяет настройки, определенные в другом файле, то будет применяться последняя загруженная настройка.

1.5.8 Пример конфигурации

Далее приведен пример простой конфигурации веб-сервера Apache:

```
Listen 80

<VirtualHost *:80>
    ServerName example.com
    DocumentRoot "/var/www/example"

    <Directory "/var/www/example">
        Options Indexes FollowSymLinks
        AllowOverride All
        Require all granted
    </Directory>

    ErrorLog $APACHE_LOG_DIR/example_error.log
    CustomLog $APACHE_LOG_DIR/example_access.log combined
</VirtualHost>
```

1.6 Настройка Apache как прокси-сервера

HTTP-сервер Apache можно настроить как в режиме прямого, так и обратного прокси-сервера (также известного как шлюз). Для реализации прокси используется модуль `mod_proxy`.

Обычный прокси-сервер – это промежуточный сервер, который находится между клиентом и исходным сервером. Чтобы получить контент с исходного сервера, клиент

отправляет запрос прокси-серверу, называя исходный сервер целевым. Затем прокси-сервер запрашивает контент с исходного сервера и возвращает его клиенту. Клиент должен быть специально настроен на использование прямого прокси-сервера для доступа к другим сайтам.

Типичное использование прямого прокси-сервера – предоставление доступа в Интернет внутренним клиентам, которые ограничены брандмауэром. Прямой прокси-сервер также может использовать кеширование (обеспечиваемое модулем `mod_cache`) для уменьшения использования сети.

Прямой прокси активируется с помощью директивы `ProxyRequests`. Поскольку перенаправленные прокси позволяют клиентам получать доступ к произвольным сайтам через внутренний сервер и скрывать их истинное происхождение, важно, чтобы сервер был защищен – только авторизованные клиенты могут получить доступ к прокси, прежде чем активировать перенаправленный прокси.

Обратный прокси-сервер (или шлюз), напротив, выглядит для клиента как обычный веб-сервер. Никакой специальной настройки на клиенте не требуется. Клиент выполняет обычные запросы контента в пространстве имен обратного прокси-сервера. Затем обратный прокси-сервер решает, куда отправлять эти запросы, и возвращает контент, как если бы он сам был источником.

Типичное использование обратного прокси-сервера – предоставить пользователям Интернета доступ к серверу, находящемуся за брандмауэром. Обратные прокси-серверы также можно использовать для балансировки нагрузки между несколькими внутренними серверами или для обеспечения кеширования (модуль `mod_cache`) более медленного внутреннего сервера. Кроме того, обратные прокси-серверы можно использовать просто для объединения нескольких серверов в одно и то же пространство `URL`.

Обратный прокси активируется с помощью директивы `ProxyPass` или флага `[P]` директивы `RewriteRule`. Для настройки обратного прокси не обязательно включать `ProxyRequests`.

1.6.1 Директива `ProxyRequests`

Директива `ProxyRequests` используется для управления поведением прокси-сервера, особенно в контексте прямого проксирования. Директива определяет, разрешено ли серверу принимать запросы от клиентов, которые хотят использовать его как открытый прокси.

Синтаксис:

```
ProxyRequests On | Off
```

где:

- `On` – включает возможность прямого проксирования. Сервер будет принимать запросы от клиентов и перенаправлять их на другие серверы. В этом режиме любой клиент может отправить запрос на проксирование, что может привести к проблемам с безопасностью, если сервер не защищен.
- `Off` – отключает возможность прямого проксирования. В этом режиме сервер будет работать только как обратный прокси, принимая запросы от клиентов и перенаправляя их на определенные бэкенд-серверы, указанные в директивах `ProxyPass` и `ProxyPassReverse`. Является значением по умолчанию.

Пример конфигурации для прямого прокси:

```
ProxyRequests On

<Proxy *>
    Order allow,deny
    # Позволить всем клиентам использовать прокси:
    Allow from all
</Proxy>
```

Пример конфигурации для обратного прокси:

```
ProxyRequests Off

<VirtualHost *:80>
    ServerName example.com

    ProxyPass / http://backend-server/
    ProxyPassReverse / http://backend-server/

    <Proxy *>
        Order deny,allow
        # Разрешить доступ только к обратному прокси:
        Allow from all
    </Proxy>
</VirtualHost>
```

1.6.2 Директива `ProxyPass`

Используется для настройки проксирования запросов от клиента к бэкенд-серверу. Директива позволяет перенаправлять запросы, поступающие на определенный `URL`, к другому серверу или приложению, что делает Apache обратным прокси-сервером.

Синтаксис:

```
ProxyPass [<путь>] [<url>] [<опции>]
```

где:

- `<путь>` – путь на внутреннем веб-сервере, который будет проксироваться. Путь может быть относительный (например, `/app`) или полный (например, `/app/`);
- `<url>` – частичный `URL`-адрес удаленного сервера, на который будут перенаправляться запросы;
- `<опции>` – дополнительные параметры (опционально), которые могут изменить поведение проксирования.

1.6.3 Директива `ProxyPassReverse`

Используется в сочетании с директивой `ProxyPass` для корректной обработки заголовков ответа, возвращаемых бэкенд-сервером. Директива позволяет изменять заголовки `Location`, `Content-Location` и `URI` в ответах от бэкенд-сервера, чтобы они соответствовали `URL`-адресам, используемым клиентами.

Синтаксис:

```
ProxyPassReverse [<путь>] [<url>]
```

где:
- `<путь>` – путь на внутреннем веб-сервере, который будет проксироваться. Путь может быть относительный (например, `/app`);
- `<url>` – URL бэкенд-сервера, с которого будут приходить ответы (например, `http://backend-server/app`).

Когда клиент делает запрос к внутреннему серверу, и этот запрос проксируется на бэкенд-сервер с помощью `ProxyPass`, бэкенд-сервер может вернуть ответ с заголовками, которые содержат ссылки на его собственный адрес. Например, если бэкенд-сервер вернет заголовок `Location: http://backend-server/app/ somepage`, клиент не сможет получить доступ к этому ресурсу, так как обращение идет к внутреннему серверу.

Директива `ProxyPassReverse` изменяет эти заголовки, чтобы они указывали на правильный URL внутреннего веб-сервера. В результате клиент получает корректные ссылки, которые может использовать для дальнейшей работы.

1.6.4 Директива `<Proxy>`

Используется для настройки параметров и ограничений для определенных прокси-объектов. Директива позволяет управлять доступом, настройками безопасности и другими параметрами для запросов, которые обрабатываются через прокси.

Синтаксис:

```
<Proxy [<url>]>
    <Директивы>
    ...
</Proxy>
```

где `<url>` определяет, какие запросы будут применяться к данной секции. Допускается использовать `*` для указания всех прокси-запросов или указать конкретный путь.

Далее будут рассмотрены основные директивы, используемые в `<Proxy>`.

Require

Используется для управления доступом к ресурсам на сервере. Директива `Require` заменила устаревшие директивы `Order`, `Allow` и `Deny`, предоставляя более гибкий способ настройки контроля доступа.

Синтаксис:

```
Require [<тип_требования>] [<условие>]
```

где параметр `<тип_требования>` может принимать следующие значения:
- `all granted` – разрешает доступ всем пользователям;
- `all denied` – запрещает доступ всем пользователям;
- `ip` – разрешает или запрещает доступ на основе IP-адресов;
- `user` – разрешает доступ определенным пользователям (при использовании аутентификации);

- `group` – разрешает доступ группам пользователей (при использовании аутентификации);
- `valid-user` – разрешает доступ любому аутентифицированному пользователю.

Timeout

Используется для настройки времени ожидания (таймаута) для различных операций, таких как чтение запроса, отправка ответа и других сетевых операций. Может быть особенно важна в контексте настройки прокси-сервера, где время ожидания может влиять на производительность и стабильность работы.

Синтаксис:

```
Timeout <время>
```

где `<время>` указывается в секундах.

1.6.5 Директива `ProxyTimeout`

Используется для настройки времени ожидания (таймаута) для прокси-запросов. Позволяет установить время, в течение которого сервер будет ждать ответа от бэкенд-сервера, к которому он проксирует запросы.

Синтаксис:

```
ProxyTimeout <время>
```

где `<время>` указывается в секундах.

1.6.6 Пример конфигурации

Далее приведен пример конфигурации Apache как обратного прокси-сервера с использованием модуля `mod_proxy`:

```
<VirtualHost *:80>
    ServerName example.com

    # Включение модуля mod_proxy:
    ProxyRequests Off

    # Настройка проксирования:
    ProxyPass /app http://backend-server/app
    ProxyPassReverse /app http://backend-server/app

    # Ограничение доступа к прокси:
    <Proxy *>
        Order deny,allow
        Deny from all
        # Разрешение доступа только из локальной сети:
        Allow from 192.168.1.0/24
    </Proxy>

    # Настройка таймаутов:
    ProxyTimeout 30
```

```
    # Журналы:
    LogLevel warn
    ErrorLog $APACHE_LOG_DIR/error.log
    CustomLog $APACHE_LOG_DIR/access.log combined
</VirtualHost>
```

1.7 Кеширование

Кеширование – это метод сохранения копий некоторых элементов веб-ресурсов, к которым недавно осуществлялся доступ на удаленном сервере. Кеширование способствует повышению производительности и масштабируемости, т. к. значительно уменьшает время на загрузку информации. Однако кеширование требует дополнительных ресурсов памяти. При переполнении кеша редко используемые элементы удаляются.

Для веб-сервера Apache функциональность кеширования предоставляет модуль `mod_cache`. Модуль может использоваться в сочетании с другими модулями, такими как `mod_disk_cache` и `mod_mem_cache`, для реализации кеширования на диске или в памяти соответственно.

Далее будут рассмотрены основные директивы модуля `mod_cache`.

1.7.1 Директива `CacheEnable`

Включает кеширование для определенного `URL`-пространства. Директива позволяет указать, какой тип кеша (диск или память) будет использоваться для определенных `URI`.

Синтаксис:

```
CacheEnable <тип_кеша> <URL-путь>
```

где:

- `<тип_кеша>` – тип кеша, который будет использоваться (например, `disk` или `mem`);
- `<URL-путь>` – путь, для которого будет включено кеширование. Путь может быть полным или задан префиксом.

1.7.2 Директива `CacheDisable`

Отключает кеширование для определенного `URL`-пространства при необходимости исключения определенных ресурсов из кеширования.

Синтаксис:

```
CacheDisable <URL-путь>
```

где `<URL-путь>` – путь, для которого будет отключено кеширование.

1.7.3 Директива `CacheRoot`

Указывает директорию, в которой будет храниться кеш на диске. Это необходимо для работы модуля `mod_disk_cache`.

Синтаксис:

```
CacheRoot <каталог>
```

где `<каталог>` – полный путь к директории, где будут храниться файлы кеша.

1.7.4 Директива `CacheSize`

Задает максимальный размер кеша в байтах. Директива используется для ограничения объема памяти, выделяемого под кеш.

Синтаксис:

```
CacheSize <размер>
```

где `<размер>` – максимальный размер кеша в байтах.

1.7.5 Директива `CacheMaxExpire`

Задает максимальное время жизни (`TTL`) для кешируемых объектов. Даже если сервер указывает более длительное время в заголовках ответа, кеш будет использовать значение `CacheMaxExpire` как верхний предел.

Синтаксис:

```
CacheMaxExpire <время>
```

где `<время>` – максимальное время жизни в секундах.

1.7.6 Директива `CacheDefaultExpire`

Устанавливает значение по умолчанию для времени жизни кешируемых объектов, если сервер не указывает его явно в заголовках ответа. Данное значение будет использоваться для всех объектов, для которых не указано другое время жизни.

Синтаксис:

```
CacheDefaultExpire <время>
```

где `<время>` – максимальное время жизни в секундах.

1.7.7 Пример конфигурации

Далее приведен пример конфигурации Apache с настройкой кеширования:

```
CacheRoot /var/cache/apache2/mod_cache
CacheEnable disk /
CacheDefaultExpire 3600
CacheMaxExpire 86400

<VirtualHost *:80>
    ServerName example.com

    <Location "/images">
        CacheEnable disk
        CacheDefaultExpire 7200
        CacheMaxExpire 86400
```

```
    </Location>

    <Location "/private">
        CacheDisable
    </Location>
</VirtualHost>
```

1.8 Балансировка нагрузки

Балансировка нагрузки – это метод распределения входящих запросов между несколькими серверами для обеспечения высокой доступности и производительности веб-приложений. В Apache существует несколько способов настройки балансировки нагрузки, включая использование модуля `mod_proxy` и его подмодулей `mod_proxy_balancer` и `mod_proxy_http`.

1.8.1 Директива `BalancerMember`

Определяет отдельные серверы (члены кластера), которые будут участвовать в балансировке нагрузки. В качестве аргументов можно указать `URL`-адрес сервера и дополнительные параметры (опции).

Синтаксис:

```
BalancerMember <URL> [<опции>]
```

В качестве дополнительных опций, задающих поведение сервера, можно указать:

- `status=on | off` – указывает, активен ли данный сервер. Если установлено значение `off`, запросы не будут направляться на указанный сервер;
- `timeout` – устанавливает таймаут соединения с указанным сервером (в секундах);
- `retry` – определяет количество попыток повторного соединения с сервером в случае его недоступности;
- `loadfactor` – позволяет установить приоритет для указанного сервера. Серверы с более высоким значением будут получать больше запросов.

Например:

```
BalancerMember http://server1.example.com retry=5 timeout=300
```

Для указания нескольких серверов, участвующих в распределении нагрузки, можно использовать директиву `<Proxy balancer>`.

1.8.2 Директива `<Proxy balancer>`

Определяет группу удаленных серверов, между которыми будет осуществляться балансировка нагрузки.

Синтаксис:

```
<Proxy balancer://<имя_группы»
    BalancerMember <URL> [<опции>]
    ...
</Proxy>
```

где:

- `<имя_группы>` – имя группы балансировщика, которое используется для идентификации этой группы в других директивах, таких как `ProxyPass` или `ProxyPass Reverse`;
- `BalancerMember <URL>` – определяет отдельный бэкенд-сервер, который будет частью группы. `URL` указывает на адрес сервера, к которому будут направляться запросы. Дополнительно можно указать параметры (опции) для настройки поведения этого сервера.

Пример конфигурации с использованием группы удаленных серверов:

```
<Proxy balancer://mycluster>
    BalancerMember http://server1.example.com
    BalancerMember http://server2.example.com
    ProxySet lbmethod=byrequests
</Proxy>
```

здесь создается группа балансировщика с именем `mycluster`, в которую входят два удаленных сервера – `server1.example.com` и `server2.example.com`, а также устанавливается метод балансировки `byrequests`, который будет распределять запросы по количеству обращений к каждому серверу.

1.8.3 Директива `ProxySet`

Используется для настройки параметров поведения прокси-сервера для определенного члена кластера. Можно указать алгоритм балансировки или параметры сессий.

Синтаксис:

```
ProxySet lbmethod=<метод>
```

где `<метод>` задает метод балансировки нагрузки и может принимать следующие значения:

- `byrequests` – балансировка по количеству запросов;
- `bytraffic` – балансировка по объему переданных данных;
- `bybusyness` – балансировка по количеству активных соединений.

Например, для балансировки нагрузки по количеству запросов необходимо указать:

```
ProxySet lbmethod=byrequests
```

Для настройки подключения одного клиента к одному и тому же удаленному серверу можно использовать так называемые «липкие» сессии. Задать такую настройку можно с помощью параметра `stickysession` директивы `ProxySet`:

```
ProxySet stickysession=JSESSIONID | jsessionid
```

где `JSESSIONID | jsessionid` – стандартный идентификатор сессии, который автоматически создается сервером приложений для отслеживания состояния сессии пользователя.

В зависимости от настроек сервера и среды выполнения, имена переменных могут

быть чувствительными к регистру. Поэтому использование обоих вариантов (с заглавными и строчными буквами) позволяет обеспечить совместимость с различными серверами и приложениями.

Также директива `ProxySet` может принимать такие параметры, как:

- `status=on | off` – указывает, активен ли данный сервер. Если установлено значение `off`, запросы не будут направляться на указанный сервер;
- `timeout` – устанавливает таймаут соединения с указанным сервером (в секундах);
- `retry` – определяет количество попыток повторного соединения с сервером в случае его недоступности.

1.8.4 Директива `<Location>`

Используется для определения `URL`-пути, который будет обрабатываться балансировщиком.

Синтаксис:

```
<Location /<путь»
    <Директивы>
    ...
</Location>
```

Директивы внутри блока `<Location>` будут применяться только к запросам, соответствующим указанному пути. Путь может быть абсолютным, относительным или использовать регулярные выражения.

Пример конфигурации обработки обращений клиентов балансировщиком:

```
<Location /app>
    ProxyPass balancer://mycluster
    ProxyPassReverse balancer://mycluster
</Location>
```

1.9 Настройка доступа к веб-серверу

Независимо от задач, которые должен решать настраиваемый веб-сервер, вопросы его безопасности (авторизация пользователей, права доступа к каталогам, ограничение доступа и т.д.) требуют особого внимания.

Контроль доступа к веб-серверу Apache может осуществляться с помощью таких модулей, как `mod_authz_core`, `mod_authn_core` и `mod_authz_host`.

Модуль `mod_authz_core` определяет директивы, которые позволяют контролировать доступ к ресурсам на основе различных условий, таких как аутентификация пользователя, `IP`-адреса, группы и пр., а также предоставляет основные механизмы для управления авторизацией.

Модуль `mod_authn_core` предоставляет основные возможности аутентификации, позволяющие разрешить или запретить доступ к каким-либо ресурсам веб-сервера.

Модуль `mod_authz_host` управляет доступом к ресурсам на основе адреса хоста или `IP`-адреса клиента. Модуль позволяет контролировать, какие хосты или `IP`-адреса имеют доступ к определённым ресурсам на сервере.

В рамках текущего раздела будут рассмотрены основные директивы, позволяющие осуществлять контроль доступа к ресурсам сервера.

1.9.1 Директива AuthType

Директива AuthType используется для указания типа аутентификации, который будет применяться к защищённым ресурсам. Директива определяет, каким образом сервер будет проверять подлинность пользователей, пытающихся получить доступ к ресурсам веб-сервера.

Синтаксис:

```
AuthType <тип_аутентификации>
```

где <тип_аутентификации> определяет тип аутентификации и может принимать следующие значения:

- None – отключить аутентификацию;
- Basic – простая аутентификация, при которой имя пользователя и пароль передаются в кодировке Base64;
- Digest – аутентификация с использованием хеширования, более безопасный тип аутентификации по сравнению с Basic. Пароли не передаются в открытом виде;
- Form – аутентификация через веб-форму, позволяет пользователям вводить учетные данные (например, имя пользователя и пароль) в HTML-форме, вместо использования стандартных диалоговых окон браузера.

1.9.2 Директива AuthName

Директива AuthName используется для указания имени области аутентификации. Имя отображается пользователю в диалоговом окне аутентификации, когда он пытается получить доступ к защищенному ресурсу.

Синтаксис:

```
AuthName "<область_аутентификации>"
```

Пример конфигурации:

```
<Location "/protected">
    AuthType Basic
    AuthName "Restricted Area"
    Require valid-user
</Location>
```

где:

- AuthType Basic – указывает, что используется базовая аутентификация;
- AuthName "Restricted Area" – задает имя области аутентификации как "Restricted Area". Когда пользователь попытается получить доступ к /protected, браузер отобразит указанное имя в диалоговом окне запроса учетных данных;
- Require valid-user – указывает, что доступ к ресурсу может получить только аутентифицированный пользователь.

1.9.3 Директива Require

Директива Require – основная директива для определения условий доступа, используется на этапе авторизации, чтобы гарантировать, что пользователю разрешен или запрещен доступ к какому-либо ресурсу.

Модуль mod_authz_host расширяет типы авторизации с помощью дополнительных

параметров – ip, host, forward-dns и local. Параметры определяют, какие хосты могут получить доступ к той или иной области сервера. Доступ можно контролировать по имени хоста, IP-адресу или диапазону IP-адресов.

Синтаксис:

```
Require <условие>
```

где <условие> может принимать следующие значения:

- ip <IP-адрес> – разрешает доступ с указанного IP-адреса или диапазона;
- host <имя_хоста> – разрешает доступ с указанного имени хоста;
- local – разрешает доступ к серверу, если выполняется одно из следующих условий:
  - адрес клиента соответствует 127.0.0.0/8;
  - адрес клиента ::1;
  - адрес клиента и сервера одинаковы.
- valid-user – разрешает доступ любому аутентифицированному пользователю;
- user <имя_пользователя> – разрешает доступ конкретному пользователю;
- group <имя_группы> – разрешает доступ пользователям из указанной группы.

1.9.4 Директива <RequireAll>

Указывает, что все условия внутри блока должны быть выполнены для предоставления доступа.

Синтаксис:

```
<RequireAll>
    Require <условие_1>
    Require <условие_2>
    ...
</RequireAll>
```

Подробнее о возможных принимаемых значениях параметра <условие> см. в п. 1.9.3 «Директива Require».

1.9.5 Директива <RequireAny>

Указывает, что достаточно выполнить хотя бы одно из условий внутри блока для предоставления доступа.

Синтаксис:

```
<RequireAny>
    Require <условие_1>
    Require <условие_2>
    ...
</RequireAny>
```

Подробнее о возможных принимаемых значениях параметра <условие> см. в п. 1.9.3 «Директива Require».

1.9.6 Директива <RequireNone>

Указывает, что ни одно из условий внутри блока не должно быть выполнено для предоставления доступа. Директива используется для явного исключения определен-

ных условий.

Синтаксис:

```
<RequireNone>
    Require <условие_1>
    Require <условие_2>
    ...
</RequireNone>
```

Подробнее о возможных принимаемых значениях параметра `<условие>` см. в п. 1.9.3 «Директива Require».

1.9.7 Директива `Satisfy`

Директива `Satisfy` определяет, как должны выполняться условия доступа. Директива `Satisfy` может принимать следующие значения:

- `all` – все условия должны быть выполнены для предоставления доступа;
- `any` – достаточно, чтобы было выполнено хотя бы одно из условий.

Синтаксис:

```
Satisfy any | all
```

1.9.8 Примеры конфигурации

Разрешение доступа всем пользователям

Для разрешения доступа к каталогу `/var/www/html/public` всем пользователям сервера необходимо указать следующую конфигурацию:

```
<Directory "/var/www/html/public">
    Require all granted
</Directory>
```

Запрет доступа всем пользователям

Для запрета доступа к каталогу `/var/www/html/private` всем пользователям сервера необходимо указать следующую конфигурацию:

```
<Directory "/var/www/html/private">
    Require all denied
</Directory>
```

Ограничение доступа по `IP`-адресам

Для разрешения доступа к каталогу `/var/www/html/private` только для клиентов с `IP`-адресами из диапазона `192.168.1.0/24` необходимо указать следующую конфигурацию:

```
<Directory "/var/www/html/private">
    Require ip 192.168.1.0/24
</Directory>
```

Комбинированные условия

Для разрешения доступа к каталогу `/var/www/html/restricted` только для клиентов с IP-адресами из диапазона `192.168.1.0/24` и только при условии их аутентификации необходимо указать следующую конфигурацию:

```
<Directory "/var/www/html/restricted">
    Require all denied
    Require ip 192.168.1.0/24
    Require valid-user
</Directory>
```

Для разрешения доступа к каталогу `/secure` только тем пользователям, которые аутентифицированы и имеют IP-адрес в диапазоне `192.168.1.0/24`, необходимо указать следующую конфигурацию:

```
<Directory /var/www/html/secure>
    <RequireAll>
        Require valid-user
        Require ip 192.168.1.0/24
    </RequireAll>
</Directory>
```

Для разрешения доступа к каталогу `/var/www/html/public` либо аутентифицированным пользователям, либо пользователям из диапазона IP-адресов `192.168.1.0/24` необходимо указать следующую конфигурацию:

```
<Directory /var/www/html/public>
    <RequireAny>
        Require valid-user
        Require ip 192.168.1.0/24
    </RequireAny>
</Directory>
```

1.10 Журналирование событий

Сервер Apache создает два основных типа регистрационных журналов: журнал регистрации ошибок и журнал регистрации обмена данных. Администратор может задать уровень серьезности (критичности) регистрируемых ошибок, начиная с которого будет производиться регистрация событий в журнале, однако тип регистрируемой информации при этом задать нельзя. Возможно также задать тип сохраняемой информации об обмене данных, однако нет возможности при этом задать уровень критичности этой информации.

Журнал регистрации ошибок представляет собой файл, в котором сервер регистрирует информацию о событиях, произошедших на сервере.

Регистрацию ошибок можно настроить таким образом, что сервер будет фиксировать фактически любое событие. По умолчанию сервер записывает информацию об ошибках в файле `/etc/httpd/logs/error_log` (задается директивой `ErrorLog`).

Журнал регистрации обмена данных (или журнал регистрации доступа) сохраняет детальную информацию о переданной и полученной информации на сервере. Данная

функциональная возможность обеспечивается модулем `mod_log_config`, который может быть усилен модулем `mod_log_common`. В отличие от журнала ошибок, журналы обмена данными не являются обязательными.

Далее будут рассмотрены основные директивы настройки файлов журналов для веб-сервера Apache.

1.10.1 Директива `ErrorLog`

Указывает путь к файлу журнала, в который будут записываться сообщения об ошибках, возникающих во время работы сервера.

Синтаксис:

```
ErrorLog <файл_журнала>
```

где `<файл_журнала>` – это путь к файлу журнала. Путь может быть как абсолютный, так и относительный. Если указанный путь не начинается с символа косой черты `/`, задающей абсолютный путь из корневого каталога файловой системы, то будет рассмотрен путь относительно каталога, указанного в директиве `ServerRoot`.

Также имеется возможность задать пересылку сообщений об ошибках в системный журнал регистрации ошибок. Для этого в директиве ErrorLog необходимо указать `syslog`.

```
ErrorLog syslog
```

Для определения критичности событий, которые необходимо фиксировать в журнале, используется утилита `LogLevel`.

1.10.2 Директива `LogLevel`

Определяет, какие сообщения будут записываться в журнал сервера, и позволяет контролировать объем выводимой информации, которая в дальнейшем используется для диагностики и отладки.

Доступны следующие уровни критичности отслеживаемых сообщений:

- `emerg` – сообщения о критических ошибках, которые могут привести к сбою системы;
- `alert` – сообщения об ошибках, требующих немедленного вмешательства;
- `crit` – сообщения о критических ошибках, которые не требуют немедленного вмешательства, но указывают на серьезные проблемы;
- `error` – сообщения об ошибках, которые не влияют на работу сервера, но могут затруднить его функционирование;
- `warn` – предупреждения о потенциальных проблемах;
- `notice` – информационные сообщения, которые могут быть полезны для администраторов;
- `info` – общие информационные сообщения о работе сервера.

Синтаксис:

```
LogLevel <уровень>
```

где `<уровень>` – уровень критичности отслеживаемых сообщений.

При указании определённого уровня критичности событий в директиве `LogLevel` в журнал попадают все сообщения указанного уровня, а также сообщения более

высоких уровней. Например, при уровне критичности `error` в журнал попадают сообщения уровней `error`, `crit`, `alert` и `emerg`. Если параметр не задан, по умолчанию используется `warn`.

1.10.3 Директива `LogFormat`

Определяет формат ведения журнала. Директива принимает два аргумента: имя формата (или метка) и строку формата, в которой используются специальные переменные для представления различных данных о запросе.

Синтаксис:

```
LogFormat "<формат>" <имя_формата>
```

где:

- `<формат>` – задает последовательность записи данных в журнал и может принимать следующие значения:
  - `%h` – IP-адрес клиента;
  - `%l` – идентификатор пользователя (обычно "-", если не используется);
  - `%u` – имя пользователя, если аутентификация прошла успешно (обычно "-", если не используется);
  - `%t` – время запроса в формате `[день/месяц/год:часы:минуты:секунды часовой_пояс]`;
  - `%r` – строка запроса (например, `GET /index.html HTTP/1.1`);
  - `%>s` – код ответа HTTP;
  - `%b` – размер ответа в байтах (без учета заголовков);
  - `%{Referer}i` – URL референера (откуда пришел запрос);
  - `%{User-Agent}i` – информация о клиенте (браузере) из заголовка `User-Agent`;
  - `%D` – время обработки запроса в микросекундах;
  - `%T` – время обработки запроса в секундах;
  - `%{...}e` – переменные окружения (например, `%{REMOTE_ADDR}e`).
- `<имя_формата>` – задает формат ведения журнала и может принимать следующие значения:
  - `common` – стандартный формат доступа, который включает IP-адрес клиента, идентификатор пользователя, дату и время запроса, строку запроса, код ответа и размер ответа:

    ```
    LogFormat "%h %l %u %t "%r" %>s %b" common
    ```

  - `combined` – расширенный формат, который включает все поля формата `common`, а также информацию о референере и пользовательском агенте:

    ```
    LogFormat "%h %l %u %t "%r" %>s %b "%{Referer}i"
    "%{User-Agent}i"" combined
    ```

  - `custom` – позволяет создать собственный формат с любыми переменными, которые необходимы:

    ```
    LogFormat "%{REMOTE_ADDR}e %U %T" custom
    ```

1.10.4 Директива CustomLog

Используется для указания файла, в который будут записываться логи, а также для выбора формата логирования, который будет применяться к этим записям.

Синтаксис:

```
CustomLog <путь_к_файлу> <имя_формата>
```

где:

- <путь_к_файлу> – задает путь к файлу журнала;
- <имя_формата> – задает формат ведения журнала и может принимать следующие значения:
  - common – стандартный формат доступа, который включает IP-адрес клиента, идентификатор пользователя, дату и время запроса, строку запроса, код ответа и размер ответа:

    ```
    LogFormat "%h %l %u %t "%r" %>s %b" common
    ```

  - combined – расширенный формат, который включает все поля формата common, а также информацию о реферере и пользовательском агенте:

    ```
    LogFormat "%h %l %u %t "%r" %>s %b "%{Referer}i"
    "%{User-Agent}i"" combined
    ```

  - custom – позволяет создать собственный формат с любыми переменными, которые необходимы:

    ```
    LogFormat "%{REMOTE_ADDR}e %U %T" custom
    ```

1.10.5 Пример конфигурации

Далее приведен пример конфигурации, в котором задаются форматы журналов и указываются файлы для записи:

```
# Определение формата логов
LogFormat "%h %l %u %t "%r" %>s %b" common
LogFormat "%{REMOTE_ADDR}e %U %T" custom

# Запись логов в файлы
CustomLog "/var/log/apache2/access.log" common
CustomLog "/var/log/apache2/custom.log" custom
```

1.10.6 Ротация журналов

Для настройки ротации файлы журналов веб-сервера необходимо сначала переименовать, а затем передать сигнал HUP главному процессу. Сигнал HUP перезапустит сервер для создания новых файлов журналов, при этом старые журналы будут сохранены.

Рекомендуется использовать такой метод ротации файлов журналов в сочетании с утилитой logrotate, отвечающей за автоматическое управление ротацией, сжатием и удалением устаревших файлов журналов.

Для настройки автоматического управления ротацией откройте файл конфигурации `logrotate` для Apache:

```
nano /etc/logrotate.d/httpd
```

и укажите необходимые настройки, например:

```
/var/log/httpd/*log {
  daily
  missingok
  rotate 14
  compress
  delaycompress
  notifempty
  create 0640 www-data adm
  sharedscripts
  postrotate
  /bin/systemctl reload httpd.service > /dev/null 2>/dev/null || true
  endscript
}
```

где:

- `daily` – задает частоту ротации, в данном примере – каждый день;
- `missingok` – не выдавать ошибку, если файл журнала отсутствует;
- `rotate 14` – количество хранимых ротаций, в данном примере `14`;
- `compress` – сжимать старые файлы;
- `delaycompress` – откладывать сжатие до следующей ротации;
- `notifempty` – не проводить ротацию для пустых файлов;
- `create 0640 www-data adm` – создавать новые файлы с указанными правами и владельцем;
- `sharedscripts` – запускать скрипты только один раз для всех файлов;
- `postrotate` – команды, которые будут выполнены после ротации.

2. Веб-сервер NGINX

2.1 Общие сведения

NGINX сочетает в себе такие понятия, как HTTP-сервер, обратный прокси-сервер с поддержкой кеширования и балансировки нагрузки, TCP/UDP прокси-сервер, а также почтовый прокси-сервер.

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 рабочим процессом 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 кеша и диспетчер кеша. Загрузчик кеша отвечает за подготовку метаданных, необходимых рабочим процессам для использования кеша, а диспетчер кеша – за проверку записей кеша 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.

2.2 Установка NGINX

Установка NGINX производится командой:

```
dnf install nginx
```

После установки запустите и добавьте в автозагрузку службу `nginx`:

```
systemctl enable --now nginx
```

Проверьте статус службы:

```
systemctl status nginx
```

В статусе должно отображаться `active (running)`.

2.3 Файл конфигурации NGINX

2.3.1 Формат файла конфигурации

Файл конфигурации веб-сервера NGINX по умолчанию называется `nginx.conf` и размещается в каталоге `/etc/nginx`.

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

```
<секция>
   <директива> <параметры>;
```

Примечание. Строки, содержащие директивы, оканчиваются точкой с запятой (`;`) – это признак конца строки.

Фигурные скобки вводят новый конфигурационный контекст.

2.3.2 Единицы измерения

Далее приведен перечень поддерживаемых NGINX единиц измерения для указания размеров, смещений и временных интервалов в конфигурационных файлах.

Размеры и смещения

Размеры можно указывать в байтах, килобайтах или мегабайтах, используя следующие суффиксы:

- `k` и `K` – для килобайт;
- `m` и `M` – для мегабайт.

Например, `1024`, `8k`, `1m`.

Смещения также можно указывать в гигабайтах, используя суффиксы `g` или `G`.

Интервалы времени

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

- `ms` – миллисекунды;
- `s` – секунды (по умолчанию);
- `m` – минуты;

- `h` – часы;
- `d` – дни;
- `w` – недели;
- `M` – месяцы (считаются равными 30 дням);
- `y` – годы (считаются равными 365 дням).

При указании одного значения можно комбинировать различные единицы, указывая их в порядке убывания, при необходимости разделяя их пробелами. Например, значение `1h 30m` задаёт тот же промежуток времени, что и значение `90m` или `5400s`.

Важные моменты:

- по умолчанию значение без суффикса интерпретируется как секунды;
- рекомендуется всегда указывать суффикс;
- некоторые интервалы времени можно задать лишь с точностью до секунд.

2.3.3 Модули в NGINX

Как было указано в п. 2.1 «Общие сведения» NGINX имеет модульную структуру – каждая из директив в конфигурационном файле связана с определенным модулем. Новые функции и возможности могут быть добавлены с помощью программных модулей, которые, при необходимости, можно подключить к работающему экземпляру NGINX.

Модули существуют двух видов — статические (встраиваемые) и динамические. *Статические* — это модули, которые входят в состав NGINX. Для их использования достаточно просто указать нужные директивы в конфигурационном файле (`nginx.conf`).

Динамические — это модули, которые можно подключать к NGINX и загружать во время работы. Для этого необходимо сначала собрать модуль как динамический, а затем загрузить его в конфигурации.

Для подключения динамического модуля используется директива `load_module` в конфигурационном файле:

```
load_module modules/ngx_http_some_module.so;
```

Также необходимо убедиться, что файл `*.so` находится в указанной директории.

В рамках текущего руководства будут рассмотрены основные статические модули, используемые для настройки веб-сервера NGINX.

2.3.4 Глобальные параметры

В глобальной секции задаются параметры, содержащие общие настройки сервера. Глобальная секция может включать как конфигурационные директивы, например `user` и `worker_processes`, так и контексты, например `events`. Глобальная секция не заключается в фигурные скобки.

В таблице 2.1 приведены наиболее распространенные директивы, задаваемые в глобальной секции.

Таблица 2.1 — Описание глобальных директив.

Директива	Описание
`user`	Пользователь и группа, от имени которого исполняются рабочие процессы. Если группа не указана, 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; если решаются задачи, требующие интенсивного ввода-вывода, – то умножать это количество на коэффициент от 1,5 до 2.
`error_log`	Файл, в который записываются сообщения об ошибках. Если ни в каком другом контексте директивы `error_log` нет, то в этом файле будут регистрироваться 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.
`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.

Пример использования глобальных директив:

```
# nginx работает от имени пользователя 'nginx'
user nginx;

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

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

# путь к pid-файлу
pid /var/run/nginx.pid;
```

```
# секция (или контекст) для модуля 'events'
events {

# произведение этого числа и значения worker_processes
# показывает, сколько может быть одновременно открыто соединений
# для одной пары <IP:порт>
worker_connections 1024;
}
```

Глобальные директивы должны находиться в начале конфигурационного файла `nginx.conf`.

2.3.5 Контексты

Некоторые глобальные контексты группируют директивы, применимые к различным типам трафика. К таким контекстам относят:

- `events` – контекст с описанием общей обработки соединений;
- `http` – контекст с описанием обработки HTTP-трафика;
- `mail` – контекст с описанием обработки почтового трафика;
- `stream` – контекст с описанием обработки трафика TCP и UDP.

Предполагается, что директивы, размещенные вне указанных контекстов, находятся в глобальной секции (`main`).

Пример файла конфигурации с несколькими контекстами:

```
user nobody; # директива основного контекста

events {
   # конфигурация обработки соединения
}

http {
   # Конфигурация, специфичная для HTTP и влияющая на все
   # виртуальные серверы

   server {
      # настройка виртуального HTTP-сервера 1
      location /one {
         # конфигурация для обработки URI, начинающихся с '/one'
      }
      location /two {
         # конфигурация для обработки URI, начинающихся с '/two'
      }
   }

   server {
      # настройка виртуального HTTP-сервера 2
   }
}
```

```
stream {
   # Конфигурация, специфичная для TCP/UDP и влияющая на все
   # виртуальные серверы
   server {
      # настройка виртуального TCP-сервера 1
   }
}
```

В рамках одного конфигурационного файла одинаковые директивы могут использоваться в разных контекстах. Как правило, дочерний контекст наследует настройки директив, включенных в родительский контекст. Для переопределения директив, унаследованных от родительского контекста, необходимо включить данную директиву в дочерний контекст.

2.3.6 Включаемые файлы

Для упрощения обработки конфигурации рекомендуется использовать включаемые файлы, хранящиеся в каталоге `/etc/nginx/conf.d`, и директиву `include` в основном файле `/etc/nginx/nginx.conf` для вызова и обработки содержимого этих файлов.

Включаемые файлы должны быть синтаксически корректны с точки зрения записи директив и блоков. Для включения файла необходимо указать путь к нему, например:

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

Метасимволы в пути позволяют включить сразу несколько файлов:

```
include /etc/nginx/default.d/*.conf;
```

Если указан неполный путь, то NGINX считает, что путь задан относительно главного местоположения конфигурационного файла. Проверить правильность конфигурационного файла можно, выполнив:

```
nginx -t -c <путь_к_файлу_конфигурации>
```

При этом на наличие ошибок проверяются и все включаемые файлы.

2.3.7 Виртуальные серверы

В каждый контекст обработки трафика включается один или несколько контекстов `server` для определения виртуальных серверов, управляющих обработкой запросов. Директивы, которые можно включить в контекст `server`, различаются в зависимости от типа трафика.

Для `HTTP`-трафика (контекст `http`) каждая директива сервера управляет обработкой запросов на ресурсы в определенных доменах или `IP`-адресах. Один или несколько контекстов `location` в контексте `server` определяют, как обрабатывать определенные наборы `URI`.

Для почты и трафика `TCP`/`UDP` (контексты `mail` и `stream`) каждая директива `server` управляет обработкой трафика, поступающего на определенный порт `TCP` или сокет `UNIX`.

Виртуальный сервер определяется сочетанием директив `listen` и `server_name`.

Директива `listen`

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 сокета в NGINX. Параметры, используемые в директиве `listen` приведены в таблице 2.2.

Таблица 2.2 — Описание параметров директивы `listen`.

Параметр	Описание
IP-адрес	Указывает конкретный IP-адрес, на котором будет слушать сервер. Если не указать, будет использоваться любой доступный интерфейс. Пример: `listen 192.168.1.1:80;;`
Порт	Указывает порт, на котором сервер будет слушать. По умолчанию используется порт 80 для HTTP и 443 для HTTPS. Пример: `listen 80;`
IPv6	Для прослушивания на IPv6-адресах используется синтаксис с квадратными скобками. Пример: `listen [::]:80;`
default_ server	Означает, что данный сервер будет сервером по умолчанию для указанной пары <адрес:порт>. Если ни один другой сервер не соответствует запросу, будет использован этот блок. Пример: `listen 80 default_server;`
backlog	Указывает размер очереди для ожидающих соединений. Это может помочь в случае высокой нагрузки. Пример: `listen 80 backlog=1024;`

Таблица 2.2 — Описание параметров директивы `listen` (продолжение таблицы).

Параметр	Описание
`reuseport`	Позволяет нескольким процессам NGINX прослушивать один и тот же порт, что может улучшить производительность на многопроцессорных системах. Пример: `listen 80 reuseport;`
`bind`	Указывает, что NGINX должен привязать сокет к указанному адресу и порту, даже если другие сокеты уже связаны с этим адресом. Пример: `listen 80 bind;`
`ssl`	Указывает, что данный блок будет использовать `SSL`/`TLS` для шифрования соединений. Обычно используется в сочетании с портом `443`. Пример: `listen 443 ssl;`
`http2`	Включает поддержку `HTTP/2` для данного блока. Обычно используется вместе с `ssl`. Пример: `listen 443 ssl http2;`

Директива `server_name`

Директива `server_name` в NGINX используется для указания доменных имен, которые будут обрабатываться контекстом `server`. Это позволяет NGINX различать разные виртуальные хосты на одном и том же `IP`-адресе, направляя запросы на соответствующий контекст `server` в зависимости от запрашиваемого домена.

Параметры, используемые в директиве `server_name` приведены в таблице 2.3.

Таблица 2.3 — Описание параметров директивы `server_name`.

Параметр	Описание
FQDN	Указывает полное доменное имя, которое будет обрабатываться сервером. Пример: `server_name example.com;`

Таблица 2.3 — Описание параметров директивы `server_name` (продолжение таблицы).

Параметр	Описание
Метасимволы	NGINX допускает использование в директиве `server_name` метасимвола `*`: • метасимвол в начале имени соответствует любому субдомену – `*.example.com`; • метасимвол в конце имени соответствует любому домену верхнего уровня – `www.example.*`.
Несколько доменных имен	Можно указать несколько доменных имен, разделяя их пробелами. Пример: `server_name example.com www.example.com;`
Значение `_`	Используется для всех запросов, которые не соответствуют другим `server_name`. Обычно данный параметр используется как резервный. Пример: `server_name _;`

2.3.8 Местоположения

Контекст `location` может встречаться в секции с описанием виртуального сервера. В качестве параметра директиве передается `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 {
   # Директивы
}
```

где:

- `<модификатор>` – необязательный параметр, указывает тип совпадения. Может принимать следующие значения:
  - `=` – проверка на точное совпадение и завершение поиска,
  - `~` – сопоставление с регулярным выражением с учетом регистра,
  - `~*` – сопоставление с регулярным выражением без учета регистра,
  - `^~` – для префиксного сопоставления, имеет более высокий приоритет, чем блоки с регулярными выражениями.
- `URI` – путь, к которому применяется данный блок.

В таблице 2.4 далее приведены директивы, которые могут использоваться в кон-

тексте `location`.

Таблица 2.4 — Описание параметров директивы `location`.

Директива	Описание

`proxy_pass`	Указывает, куда передавать запросы, которые обрабатываются данным блоком `location`. Обычно используется для проксирования запросов к другим серверам.
	Пример:
	`location /api/ {`
	`proxy_pass http://backend-server;`
	`}`

`root`	Указывает корневую директорию для поиска файлов. Используется для обслуживания статических файлов.
	Пример:
	`location /images/ {`
	`root /var/www/html;`
	`}`

`alias`	Похож на `root`, но заменяет часть пути, а не добавляет к нему. Используется для создания виртуальных путей.
	Пример:
	`location /files/ {`
	`alias /var/www/files/;`
	`}`

`index`	Указывает файл, который будет использоваться в качестве индекса, если запрашивается директория.
	Пример:
	`location / {`
	`index index.html index.htm;`
	`}`

`try_files`	Проверяет наличие указанных файлов и возвращает первый найденный. Если файлы не найдены, можно указать перенаправление на другой `URI`.
	Пример:
	`location / {`
	`try_files $uri $uri/ /index.html;`
	`}`

Таблица 2.4 — Описание параметров директивы `location` (продолжение таблицы).

Директива	Описание

`rewrite`	Перенаправляет запросы на другой `URI` с возможностью использования регулярных выражений.
	Пример:
	`location /old-path/ {`
	`rewrite ^/old-path/(.*)$ /new-path/$1 permanent;`
	`}`

`expires`	Устанавливает заголовок `Expires` или `Cache-Control` для управления кешированием.
	Пример:
	`location /images/ {`
	`expires 30d;`
	`}`

`add_header`	Добавляет заголовки к ответам сервера.
	Пример:
	`location / {`
	`add_header X-Content-Type-Options nosniff;`
	`}`

`return`	Возвращает указанный код состояния и (опционально) текст ответа.
	Пример:
	`location /maintenance {`
	`return 503;`
	`}`

`limit_except`	Ограничивает методы `HTTP`, которые могут быть использованы для данного ресурса.
	Пример:
	`location /admin {`
	`limit_except GET POST {`
	`deny all;`
	`}`
	`}`

Таблица 2.4 — Описание параметров директивы `location` (продолжение таблицы).

Директива	Описание

`set`	Устанавливает переменные, которые могут быть использованы в других директивах.
	Пример:
	`location / {`
	`set $my_var "some_value";`
	`}`

`error_page`	Определяет страницы ошибок для определенных кодов состояния.
	Пример:
	`location / {`
	`error_page 404 /404.html;`
	`}`

`satisfy`	Определяет условия доступа, комбинируя различные методы аутентификации.
	Пример:
	`location /protected {`
	`satisfy any;`
	`auth_basic "Restricted";`
	`auth_basic_user_file /etc/nginx/.htpasswd;`
	`}`

`limit_req`	Ограничивает количество запросов от одного `IP`-адреса для защиты от `DdoS`-атак.
	Пример:
	`location /api/ {`
	`limit_req zone=one burst=5;`
	`}`

2.4 Управление конфигурацией NGINX

2.4.1 Управление через системную службу

Когда веб-сервер NGINX настроен, необходимо запустить и добавить в автозагрузку службу `nginx`:

```
systemctl enable --now nginx
```

Для проверки статуса сервиса используется команда:

```
systemctl status nginx
```

При успешном запуске сервиса в статусе должно быть отображено `active (running)`.

Для переопределения конфигурации, например, после внесения в файл конфигурации каких-либо изменений, необходимо перезапустить службу `nginx`:

```
systemctl reload nginx
```

Для остановки работы службы `nginx` необходимо выполнить:

```
systemctl stop nginx
```

2.4.2 Управление с помощью сигналов

NGINX имеет один главный процесс (`master process`) и один или несколько рабочих процессов (`worker processes`). Если кеширование включено, процессы загрузчика кеша и диспетчера кеша также запускаются.

Основная цель главного процесса — чтение и оценка файлов конфигурации, а также обслуживание рабочих процессов.

Рабочие процессы выполняют фактическую обработку запросов. NGINX использует зависящие от ОС механизмы для эффективного распределения запросов между рабочими процессами. Количество рабочих процессов определяется директивой `worker_processes` в файле конфигурации `nginx.conf` и может быть либо установлено на фиксированное число, либо настроено на автоматическую настройку в зависимости от количества доступных ядер ЦП.

Управление конфигурацией NGINX можно производить отправкой соответствующего сигнала главному процессу. Сигнал можно отправить, выполнив команду вида:

```
nginx -s <СИГНАЛ>
```

где `<СИГНАЛ>` может принимать следующие значения:

- `quit` – корректное завершение работы (сигнал `SIGQUIT`);
- `reload` – перезагрузить файл конфигурации (сигнал `SIGHUP`);
- `reopen` – повторно открыть файлы журнала (сигнал `SIGUSR1`);
- `stop` – немедленное выключение (или быстрое выключение, сигнал `SIGTERM`).

Важно! Команда должна быть выполнена под тем же пользователем, под которым был запущен nginx. ■

Утилита `kill` также может использоваться для отправки сигнала непосредственно главному процессу. Идентификатор главного процесса по умолчанию записывается в файл `nginx.pid`, который находится в каталоге `/var/run/`.

Главный процесс поддерживает следующие сигналы:

- `TERM`, `INT` – быстрое завершение работы;
- `QUIT` – постепенное завершение работы;
- `HUP` – переопределение конфигурации, обновление изменившейся временной зоны, запуск новых рабочих процессов с новой конфигурацией, плавное завершение

старых рабочих процессов;
- USR1 – перезапуск файлов журналов;
- USR2 – обновление исполняемого файла;
- WINCH – постепенное завершение рабочих процессов.

Управление рабочими процессами по отдельности также возможно, но не является обязательным. Рабочие процессы поддерживают следующие сигналы:
- TERM, INT – быстрое завершение работы;
- QUIT – постепенное завершение работы;
- USR1 – перезапуск файлов журналов;
- WINCH – аварийное завершение для отладки (требует включения debug_points).

2.5 Настройка NGINX в качестве веб-сервера

На высоком уровне настройка NGINX в качестве веб-сервера — это определение того, какие URL-адреса он обрабатывает и как он обрабатывает HTTP-запросы к ресурсам по этим URL-адресам. На более низком уровне конфигурация определяет набор виртуальных серверов, которые управляют обработкой запросов для определенных доменов или IP-адресов.

Каждый виртуальный сервер для HTTP-трафика определяет специальные экземпляры конфигурации, называемые местоположениями, которые управляют обработкой определенных наборов URI. Каждое местоположение определяет свой собственный сценарий того, что происходит с запросами, сопоставленными с этим местоположением. NGINX обеспечивает полный контроль над этим процессом. Каждое местоположение может проксировать запрос или возвращать файл. Кроме того, URI можно изменить, чтобы запрос перенаправлялся в другое место или на виртуальный сервер. В дополнение к этому может быть возвращен конкретный код ошибки, соответственно, можно настроить определенную страницу для соответствия каждому коду ошибки.

2.5.1 Базовый модуль HTTP – ngx_http_core_module

Базовый модуль ngx_http_core_module в NGINX предоставляет функциональность для обработки HTTP-запросов и ответов. Он является одним из основных компонентов веб-сервера NGINX и отвечает за маршрутизацию запросов, управление соединениями, обработку статических файлов и динамического контента, а также за взаимодействие с другими модулями.

Далее будут рассмотрены основные директивы базового модуля ngx_http_core_module.

client_header_buffer_size

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

Синтаксис:

```
client_header_buffer_size <размер>;
```

где <размер> – задает размер буфера, по умолчанию 1 Кб.

large_client_header_buffers

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

Синтаксис:

```
large_client_header_buffers <число> <размер>;
```

где:

- <число> – число буферов, по умолчанию 4;
- <размер> – размер буфера, по умолчанию 8 Кб.

server

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

Подробнее о директиве server см. в п. 2.3.7 «Виртуальные серверы».

server_tokens

Разрешает или запрещает выдавать версию NGINX на страницах ошибок и в поле Server заголовка ответа.

Синтаксис:

```
server_tokens on | off | build | <строка>;
```

где:

- on – разрешает выдавать версию NGINX, является значением по умолчанию;
- off – запрещает выдавать версию NGINX;
- build – наряду с версией NGINX будет также выдаваться имя сборки;
- <строка> – подписи на страницах ошибок и значение поля Server заголовка ответа можно задать явно с помощью строки с переменными; пустая строка запрещает выдачу поля Server.

location

Устанавливает конфигурацию в зависимости от URI запроса.

Подробнее о директиве location см. в п. 2.3.8 «Местоположения».

limit_except

Ограничивает HTTP-методы, доступные внутри location.

Синтаксис:

```
limit_except <метод>
```

```
{
...
}
```

где <метод> может принимать следующие значения:

- GET,
- HEAD,
- POST,
- PUT,
- DELETE,
- MKCOL,
- COPY,
- MOVE,
- OPTIONS,
- PROPFIND,
- PROPPATCH,
- LOCK,
- UNLOCK,
- PATCH.

Если разрешён метод GET, то метод HEAD также будет разрешён.

root

Задаёт корневой каталог для запросов.

Синтаксис:

```
root <путь>;
```

где в значении параметра <путь> можно использовать переменные, кроме $document_root и $realpath_root. По умолчанию используется значение html.

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

alias

Задаёт замену для указанного location.

Синтаксис:

```
alias <путь>;
```

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

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

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

`internal`

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

- запросы, перенаправленные директивами `error_page`, `index`, `internal_redirect`, `random_index` и `try_files`;
- запросы, перенаправленные с помощью поля `X-Accel-Redirect` заголовка ответа вышестоящего сервера;
- подзапросы, формируемые командой `include virtual` модуля `ngx_http_ssi_module`, директивами модуля `ngx_http_addition_module`, а также директивами `auth_request` и `mirror`;
- запросы, изменённые директивой `rewrite`.

Пример:

```
error_page 404 /404.html;

location = /404.html
   internal;
```

Примечание. Для предотвращения зацикливания, которое может возникнуть при использовании некорректных конфигураций, количество внутренних перенаправлений ограничено количеством `10`. По достижении этого ограничения будет возвращена ошибка `500` (`Internal Server Error`). В таком случае в файле журнала ошибок можно увидеть сообщение `rewrite or internal redirection cycle`.

`client_max_body_size`

Задаёт максимально допустимый размер тела запроса клиента. Если размер больше заданного, то клиенту возвращается ошибка `413` (`Request Entity Too Large`).

Синтаксис:

```
client_max_body_size <размер>;
```

где `<размер>` по умолчанию равен `1 Мб`. Установка параметра `<размер>` в `0` отключает проверку размера тела запроса клиента.

2.5.2 Модуль `ngx_http_ssl_module`

Модуль `ngx_http_ssl_module` в NGINX предоставляет поддержку `SSL` и `TLS` для защищенных соединений `HTTP`. Он позволяет серверу обрабатывать `HTTPS`-запросы, шифруя данные между клиентом и сервером. Для включения поддержки `SSL` и `TLS` для нужного блока `server` необходимо указать директиву `ssl` в контексте `listen`:

```
server {
   listen 443 ssl;
   ...
}
```

Далее будут рассмотрены основные директивы модуля `ngx_http_ssl_module`.

`ssl_certificate`

Указывает путь к файлу сертификата в формате `PEM` для данного виртуального сервера. Если в дополнение к основному сертификату нужно указать промежуточные, то они должны находиться в этом же файле в следующем порядке: сначала основной сертификат, а затем промежуточные. В этом же файле может находиться секретный ключ в формате `PEM`.

Синтаксис:

```
ssl_certificate <путь_к_файлу_сертификата>;
```

Примечание. Из-за ограничения протокола `HTTPS` для максимальной совместимости виртуальные серверы должны слушать на разных `IP`-адресах.

`ssl_certificate_key`

Указывает путь к файлу закрытого ключа в формате `PEM` для данного виртуального сервера.

Синтаксис:

```
ssl_certificate_key <путь_к_секретному_ключу>;
```

`ssl_ciphers`

Указывает набор шифров, которые могут использоваться для шифрования соединений.

Синтаксис:

```
ssl_ciphers <шифры>;
```

где `<шифры>` задаются в формате, поддерживаемом библиотекой `OpenSSL`. По умолчанию значение параметра `<шифры>` равно `HIGH:!aNULL:!MD5ttt`.

`ssl_protocols`

Определяет, какие версии протоколов `SSL`/`TLS` будут использоваться.

Синтаксис:

```
ssl_protocols [SSLv2] [SSLv3] [TLSv1] [TLSv1.1] [TLSv1.2]
[TLSv1.3];
```

По умолчанию используется следующий набор протоколов:

```
ssl_protocols TLSv1 TLSv1.1 TLSv1.2 TLSv1.3;
```

Параметры `TLSv1.1` и `TLSv1.2` работают только при использовании `OpenSSL 1.0.1` и выше.

Параметр `TLSv1.3` работает только при использовании `OpenSSL 1.1.1` и выше.

Параметр `TLSv1.3` используется по умолчанию в NGINX начиная с версии `1.23.4`.

`ssl_prefer_server_ciphers`

Указывает приоритет серверных и клиентских шифров при использовании протоколов `SSLv3` и `TLS`.

Синтаксис:

```
ssl_prefer_server_ciphers on | off;
```

Если значение директивы установлено в `on`, серверные шифры будут иметь более высокий приоритет, чем клиентские. По умолчанию значение директивы установлено в `off`.

`ssl_dhparam`

Указывает путь к файлу с параметрами для `DHE`-шифров.

По умолчанию параметры не заданы, и соответственно `DHE`-шифры не используются.

Синтаксис:

```
ssl_dhparam <путь_к_файлу_DHE-шифров>;
```

`ssl_session_cache`

Задаёт тип и размеры кешей для хранения параметров сессий.

Синтаксис:

```
ssl_session_cache off | none | [builtin[:<размер>]] [shared:
<название>:<размер>];
```

где:

- `off` – жесткий запрет использования кеша сессий; NGINX явно сообщает клиенту, что сессии не могут использоваться повторно.
- `none` – мягкий запрет использования кеша сессий; NGINX сообщает клиенту, что сессии могут использоваться повторно, но на самом деле не хранит параметры сессии в кеше.
- `builtin` – встроенный в `OpenSSL` кеш, используется в рамках только одного рабочего процесса. Размер кеша задаётся в сессиях. Если размер не задан, то он равен `20480` сессиям. Использование встроенного кеша может вести к фрагментации памяти.
- `shared` – кеш, разделяемый между всеми рабочими процессами. Размер кеша задаётся в байтах, в `1 Мб` может поместиться около 4000 сессий. У каждого разделяемого кеша должно быть произвольное название. Кеш с одинаковым названием может использоваться в нескольких виртуальных серверах. Также он используется для автоматического создания, хранения и периодического обновления ключей `TLS session tickets`, если они не указаны явно с помощью директивы `ssl_session_ticket_key`.

Можно использовать одновременно оба типа кеша, например:

```
ssl_session_cache builtin:1000 shared:SSL:10m;
```

`ssl_session_timeout`

Задаёт время, в течение которого клиент может повторно использовать параметры сессии. Иными словами указывает время жизни сеанса `SSL`.

Синтаксис:

```
ssl_session_timeout <время>;
```

По умолчанию значение параметра `<время>` равно `5m` (5 минут).

`ssl_verify_client`

Разрешает проверку клиентских сертификатов. Результат проверки доступен через переменную `$ssl_client_verify`.

Синтаксис:

```
ssl_verify_client on | off | optional | optional_no_ca;
```

где:

- `on` – проверка клиентских сертификатов включена;
- `off` – проверка клиентских сертификатов выключена;
- `optional` – запрашивает клиентский сертификат, и если сертификат был предоставлен, проверяет его;
- `optional_no_ca` – запрашивает сертификат клиента, но не требует, чтобы он был подписан доверенным центром сертификации (ЦС, CA). Это предназначено для случаев, когда фактическая проверка сертификата осуществляется внешним по отношению к NGINX сервисом. Содержимое сертификата доступно через переменную `$ssl_client_cert`.

2.5.3 Модуль `ngx_http_map_module`

Модуль `ngx_http_map_module` создаёт переменные, значения которых зависят от значений других переменных.

Далее будут рассмотрены основные директивы модуля `ngx_http_map_module`.

`map`

Создаёт новую переменную, значение которой зависит от значений одной или более исходных переменных, указанных в первом параметре.

Параметры внутри блока `map` задают соответствие между исходными и результирующими значениями.

Синтаксис:

```
map <строка> $<переменная> {
...
}
```

Исходные значения задаются строками или регулярными выражениями. Строки проверяются без учёта регистра.

Перед регулярным выражением ставится символ `~`, если при сравнении следует учитывать регистр символов, либо символы `~*` , если регистр символов учитывать не нужно. Регулярное выражение может содержать именованные и позиционные выделения, которые могут затем использоваться в других директивах совместно с результирующей переменной.

Если исходное значение совпадает с именем одного из специальных параметров, описанных ниже, перед ним следует поставить символ `\`.

В качестве результирующего значения можно указать текст, переменную и их комбинации.

Также поддерживаются следующие специальные параметры:

- `default <значение>` – задаёт результирующее значение, если исходное значение не совпадает ни с одним из перечисленных. Если параметр `default` не указан, результирующим значением по умолчанию будет пустая строка.
- `hostnames` – указывает, что в качестве исходных значений можно использовать маску для первой или последней части имени хоста, например:

```
*.example.com 1;
example.*     1;
```

  Вместо двух записей:

```
example.com   1;
*.example.com 1;
```

  можно использовать одну:

```
.example.com  1;
```

  Этот параметр следует указывать перед списком значений.
- `include <файл>` – включает файл со значениями. Включений может быть несколько.
- `volatile` – указывает, что переменная не кешируется.

Если исходному значению соответствует несколько из указанных вариантов, например, одновременно подходят и маска, и регулярное выражение, будет выбран первый подходящий вариант в следующем порядке приоритета:

- строковое значение без маски;
- самое длинное строковое значение с маской в начале, например `*.example. com`;
- самое длинное строковое значение с маской в конце, например `mail.*`;
- первое подходящее регулярное выражение (в порядке следования в конфигурационном файле);
- значение по умолчанию (`default`).

`map_hash_bucket_size`

Задаёт размер корзины в хеш-таблицах для переменных `map`. Значение по умолчанию зависит от размера строки кеша процессора.

Синтаксис:

```
map_hash_bucket_size <размер>;
```

`map_hash_max_size`

Задаёт максимальный размер хеш-таблиц для переменных `map`.

Синтаксис:

```
map_hash_max_size размер;
```

По умолчанию максимальный размер хеш-таблиц равен `2048`. Подробнее о хеш-таблицах см. в п. 2.5.3 «Настройка хеш-таблиц».

Настройка хеш-таблиц

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

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

2.5.4 Модуль `ngx_http_limit_conn_module`

Модуль `ngx_http_limit_conn_module` позволяет ограничить число соединений по заданному ключу, в частности, число соединений с одного IP-адреса.

Учитываются не все соединения, а лишь те, в которых имеются запросы, обрабатываемые сервером, и заголовок запроса уже прочитан.

Далее будут рассмотрены основные директивы модуля `ngx_http_limit_conn_module`.

`limit_conn_zone`

Задаёт параметры зоны разделяемой памяти, которая хранит состояние для разных значений ключа. Состояние в частности содержит текущее число соединений. В качестве ключа можно использовать текст, переменные и их комбинации. Запросы с пустым значением ключа не учитываются.

Синтаксис:

```
limit_conn_zone <ключ> zone=<название>:<размер>;
```

Пример использования:

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

Здесь в качестве ключа используется IP-адрес клиента. Обратите внимание, что вместо переменной `$remote_addr` использована переменная `$binary_remote_ addr`.

Длина значения переменной `$remote_addr` может колебаться от 7 до 15 байт, при этом размер хранимого состояния составляет либо 32, либо 64 байта на 32-битных платформах и всегда 64 байта на 64-битных.

Длина значения переменной `$binary_remote_addr` всегда равна 4 байтам для IPv4-адресов или 16 байтам для IPv6-адресов. При этом размер состояния всегда равен 32 или 64 байтам на 32-битных платформах и 64 байтам на 64-битных.

В зоне размером 1 мегабайт может разместиться около 32 тысяч состояний размером 32 байта или 16 тысяч состояний размером 64 байта. При переполнении зоны в ответ на последующие запросы сервер будет возвращать ошибку.

`limit_conn`

Задаёт зону разделяемой памяти и максимально допустимое число соединений для одного значения ключа. При превышении этого числа в ответ на запрос сервер вернёт ошибку.

Синтаксис:

```
limit_conn <зона> <число>;
```

где:

- `<зона>` – имя заданной в `limit_conn_zone` зоны адресов;
- `<число>` – количество разрешенных соединений.

Директив `limit_conn` может быть несколько.

Директивы наследуются от родительского уровня конфигурации при условии, что в дочернем уровне не описаны свои директивы `limit_conn`.

Пример конфигурации, которая ограничивает число соединений с сервером с одного клиентского IP-адреса и в то же время ограничивает общее число соединений с виртуальным сервером:

```
limit_conn_zone $binary_remote_addr zone=perip:10m;
limit_conn_zone $server_name zone=perserver:10m;

server {
    ...
    limit_conn perip 10;
    limit_conn perserver 100;
}
```

2.5.5 Модуль `ngx_http_limit_req_module`

Модуль `ngx_http_limit_req_module` позволяет ограничить скорость обработки запросов по заданному ключу или, как частный случай, скорость обработки запросов, поступающих с одного IP-адреса. Ограничение обеспечивается с помощью метода `leaky bucket`.

Далее будут рассмотрены основные директивы модуля `ngx_http_limit_req_module`.

`limit_req_zone`

Задаёт параметры зоны разделяемой памяти, которая хранит состояние для разных значений ключа. Состояние в частности хранит текущее число избыточных запросов. В качестве ключа можно использовать текст, переменные и их комбинации. Запросы с пустым значением ключа не учитываются.

Синтаксис:

```
limit_req_zone <ключ> zone=<название>:<размер> rate=<скорость>
[sync];
```

При переполнении зоны удаляется наименее востребованное состояние. Если и это не позволяет создать новое состояние, запрос завершается с ошибкой.

Скорость задаётся в запросах в секунду (`r/s`). Если же нужна скорость меньше одного запроса в секунду, то она задаётся в запросах в минуту (`r/m`), например, ползапроса в секунду — это `30r/m`.

Параметр sync разрешает синхронизацию данной зоны разделяемой памяти.

`limit_req`

Задаёт зону разделяемой памяти (`zone`) и максимальный размер всплеска запросов (`burst`). Если скорость поступления запросов превышает описанную в зоне, то их обработка задерживается так, чтобы запросы обрабатывались с заданной скоростью. Избыточные запросы задерживаются до тех пор, пока их число не превысит максимальный размер всплеска. При превышении запрос завершается с ошибкой. По умолчанию максимальный размер всплеска равен нулю.

Синтаксис:

```
   limit_req zone=<название> [burst=<число>] [nodelay | delay=
<число>];
```

Если избыточные запросы в пределах лимита всплесков задерживать не требуется, то следует использовать параметр `nodelay`.

Параметр `delay` задаёт лимит, по достижении которого избыточные запросы задерживаются. Значение по умолчанию равно нулю и означает, что задерживаются все избыточные запросы.

Директив `limit_req` может быть несколько.

Директивы наследуются с родительского уровня конфигурации при условии, что в дочерней конфигурации не описаны свои директивы `limit_req`.

Пример конфигурации, ограничивающей скорость обработки запросов, поступающих с одного IP-адреса, и в то же время ограничивающей скорость обработки запросов одним виртуальным сервером:

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

   server {
      ...
      limit_req zone=perip burst=5 nodelay;
      limit_req zone=perserver burst=10;
   }
```

2.5.6 Модуль `ngx_http_auth_request_module`

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

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

Далее будут рассмотрены основные директивы модуля `ngx_http_auth_request_module`.

`auth_request`

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

Синтаксис:

```
auth_request uri | off;
```

По умолчанию используется значение `off`.

`auth_request_set`

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

Синтаксис:

```
auth_request_set $<переменная> <значение>;
```

2.5.7 Модуль `ngx_http_rewrite_module`

Модуль `ngx_http_rewrite_module` позволяет изменять `URI` запроса с помощью регулярных выражений `PCRE`, совершать перенаправления и выбирать конфигурацию по условию.

Директивы `break`, `if`, `return`, `rewrite` и `set` обрабатываются в следующем порядке:

- последовательно выполняются директивы этого модуля, описанные на уровне `server`;
- в цикле:
  - ищется `location` по `URI` запроса;
  - последовательно выполняются директивы этого модуля, описанные в найденном `location`;
- цикл повторяется, если `URI` запроса изменялся, но не более 10 раз.

Далее будут рассмотрены основные директивы модуля `ngx_http_rewrite_module`.

`if`

Проверяется указанное условие. Если оно истинно, то выполняются указанные в фигурных скобках директивы этого модуля и запросу назначается конфигурация, указанная внутри директивы `if`. Конфигурации внутри директив `if` наследуются с предыдущего уровня конфигурации.

Синтаксис:

```
if (<условие>) {
...
}
```

В качестве `<условия>` могут быть заданы:

- имя переменной; ложными значениями переменной являются пустая строка или `0`;
- сравнение переменной со строкой с помощью операторов `=` и `!=`;
- соответствие переменной регулярному выражению с учётом регистра символов — `~` и без него — `~*`. В регулярных выражениях можно использовать выделения, которые затем доступны в виде переменных `$1..$9`. Также можно использовать отрицательные операторы `!~` и `!~*`. Если в регулярном выражении встречаются символы } или `;` , то всё выражение следует заключить в одинарные или двойные кавычки;
- проверка существования файла с помощью операторов `-f` и `!-f`;
- проверка существования каталога с помощью операторов `-d` и `!-d`;
- проверка существования файла, каталога или символической ссылки с помощью операторов `-e` и `!-e`;
- проверка исполняемости файла с помощью операторов `-x` и `!-x`.

`return`

Завершает обработку и возвращает клиенту указанный код. Нестандартный код `444` закрывает соединение без передачи заголовка ответа.

Синтаксис:

```
return <код> [<текст>];
return <код> URL;
return URL;
```

Можно задать либо `URL` перенаправления (для кодов `301`, `302`, `303`, `307` и `308`) либо текст тела ответа (для остальных кодов). В тексте тела ответа и `URL` перенаправления можно использовать переменные. Как частный случай, `URL` перенаправления может быть задан как `URI`, локальный для данного сервера, при этом полный `URL` перенаправления формируется согласно схеме запроса (`$scheme`) и директивам `server_name_in_redirect` и `port_in_redirect`.

Кроме того, в качестве единственного параметра можно указать `URL` для временного перенаправления с кодом `302`. Такой параметр должен начинаться со строк `http://`, `https://` или `$scheme`. В `URL` можно использовать переменные.

2.5.8 Модуль `ngx_http_access_module`

Модуль `ngx_http_access_module` позволяет ограничить доступ для определённых адресов клиентов.

`allow`

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

Синтаксис:

```
allow <адрес> | CIDR | unix: | all;
```

`deny`

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

Синтаксис:

```
deny <адрес> | CIDR | unix: | all;
```

Пример конфигурации:

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

Правила проверяются в порядке их записи до первого соответствия. В данном примере доступ разрешён только для IPv4-сетей 10.1.1.0/16 и 192.168.1.0/24, кроме адреса 192.168.1.1, и для IPv6-сети 2001:0db8::/32.

2.5.9 Модуль ngx_http_headers_module

Модуль ngx_http_headers_module позволяет выдавать поля заголовка Expires и Cache-Control, а также добавлять произвольные поля в заголовок ответа.

expires

Разрешает или запрещает добавлять или менять поля Expires и Cache-Control в заголовке ответа при условии, что код ответа равен 200, 201, 204, 206, 301, 302, 303, 304, 307 или 308.

В качестве параметра можно задать положительное или отрицательное время.

Синтаксис:

```
expires [modified] <время>;
expires epoch | max | off;
```

Время в поле Expires высчитывается как сумма текущего времени и времени, заданного в директиве. Если используется параметр modified, то время определяется как сумма времени модификации файла и времени, заданного в директиве.

Кроме того, с помощью префикса @ можно задать время суток:

```
expires @15h30m;
```

Содержимое поля Cache-Control зависит от знака заданного времени:

- отрицательное время — Cache-Control: no-cache.
- положительное или равное нулю время — Cache-Control: max-age=t, где t – это время в секундах, заданное в директиве.

Параметр epoch задаёт время «Thu, 01 Jan 1970 00:00:01 GMT» (1 января 1970 00:00:01 GMT) для поля Expires и «no-cache» для поля Cache-Control.

Параметр max задаёт время «Thu, 31 Dec 2037 23:55:55 GMT» (31 декабря 2037 23:55:55 GMT) для поля Expires и 10 лет для поля Cache-Control.

Параметр off запрещает добавлять или менять поля Expires и Cache-Control в заголовке ответа.

`add_header`

Добавляет указанное поле в заголовок ответа при условии, что код ответа равен 200, 201, 204, 206, 301, 302, 303, 304, 307 или 308. В значении параметра можно использовать переменные.

Директив `add_header` может быть несколько. Директивы наследуются от родительской конфигурации при условии, что на дочернем уровне не описаны свои директивы `add_header`.

Синтаксис:

```
add_header <имя> <значение> [always];
```

Если указан параметр `always`, то поле заголовка будет добавлено независимо от кода ответа.

2.5.10 Модуль `ngx_http_index_module`

Модуль `ngx_http_index_module` обслуживает запросы, оканчивающиеся слэшем (`/`). Такие запросы также могут обслуживаться модулями `ngx_http_auto- index_module` и `ngx_http_random_index_module`.

`index`

Определяет файлы, которые будут использоваться в качестве индекса. В имени файла можно использовать переменные. Наличие файлов проверяется в порядке их перечисления. В конце списка может стоять файл с абсолютным путём.

Синтаксис:

```
index <файл> ...;
```

Необходимо иметь в виду, что при использовании индексного файла делается внутреннее перенаправление и запрос может быть обработан уже в другом блоке `location`. Например, в конфигурации вида:

```
location = /
   index index.html;

location /
   ...
```

запрос `/` будет фактически обработан во втором блоке `location` как `/index. html`.

2.5.11 Модуль `ngx_http_autoindex_module`

Модуль `ngx_http_autoindex_module` обслуживает запросы, оканчивающиеся слэшем (`/`), и выдаёт листинг каталога. Обычно запрос попадает к модулю `ngx_http_auto-index_module`, когда модуль `ngx_http_index_module` не нашёл индексный файл.

`autoindex`

Разрешает или запрещает вывод листинга каталога.

Синтаксис:

```
autoindex on | off;
```

Значение по умолчанию `off`.

2.5.12 Модуль ngx_http_gzip_module

Модуль ngx_http_gzip_module — это фильтр, сжимающий ответ методом gzip, что позволяет уменьшить размер передаваемых данных в 2 и более раз.

Далее будут рассмотрены основные директивы модуля ngx_http_gzip_module.

gzip

Разрешает или запрещает сжатие ответа методом gzip.

Синтаксис:

```
gzip on | off;
```

Значение по умолчанию off.

gzip_types

Разрешает сжатие ответа методом gzip для указанных MIME-типов в дополнение к text/html. Специальное значение * соответствует любому MIME-типу. Ответы с типом text/html сжимаются всегда.

Синтаксис:

```
gzip_types <mime-тип> ...;
```

Значение по умолчанию:

```
gzip_types text/html;
```

gzip_min_length

Устанавливает минимальную длину ответа, который будет сжиматься методом gzip. Длина определяется только из поля Content-Length заголовка ответа.

Синтаксис:

```
gzip_min_length <длина>;
```

Значение по умолчанию:

```
gzip_min_length 20;
```

2.6 Настройка NGINX как обратного прокси-сервера

Обратный прокси-сервер — это тип прокси-сервера, который ретранслирует запросы клиентов из внешней сети на один или несколько серверов, логически расположенных во внутренней сети.

В отличие от обычного прокси-сервера, который работает от имени клиента и перенаправляет запросы к серверам, обратный прокси-сервер получает запросы от клиентов и пересылает их на один или несколько серверов.

Проксируемый сервер определяется как сервер, с которым NGINX устанавливает соединение для выполнения запроса клиента. Он может находиться на другой физической или виртуальной машине (но необязательно). Проксируемый сервер может быть демоном, прослушивающим сокет в домене UNIX на локальной машине, или одним из многих демонов, прослушивающих порты TCP на другой машине. Это также может быть сервер Apache с модулями для обработки запросов различных типов.

NGINX может проксировать запросы серверам различных типов.

Сначала клиент подключается к прокси-серверу и запрашивает какой-либо ресурс, расположенный на другом сервере. Затем прокси-сервер либо подключается к указанному серверу и получает ресурс у него, либо возвращает ресурс из собственного кеша (в случаях, если прокси имеет свой кеш).

В зависимости от протокола, который должен применяться для работы, в NGINX для настройки прокси-серверов доступно несколько модулей:

- `ngx_http_proxy_module` – позволяет передавать запросы `HTTP`-серверу;
- `ngx_http_uwsgi_module` – позволяет передавать запросы `uwsgi`-серверу;
- `ngx_http_fastcgi_module` – позволяет передавать запросы `FastCGI`-серверу;
- `ngx_http_scgi_module` – позволяет передавать запросы `SCGI`-серверу;
- `ngx_http_memcached_module` – позволяет получать ответ с сервера `memcached`.

2.6.1 Модуль `ngx_http_proxy_module`

Модуль `ngx_http_proxy_module` позволяет передавать запросы другому `HTTP`-серверу.

Далее будут рассмотрены основные директивы модуля `ngx_http_proxy_module`.

`proxy_pass`

Задаёт протокол и адрес проксируемого сервера, а также необязательный `URI`, на который должен отображаться `location`. В качестве протокола можно указать `http` или `https`.

Синтаксис:

```
proxy_pass <URL>;
```

Адрес может быть указан в виде доменного имени или `IP`-адреса и необязательного порта:

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

Также поддерживается указание адреса в виде пути `UNIX`-сокета, который задается после слова `unix` и заключается в двоеточия:

```
proxy_pass http://unix:/tmp/backend.socket:/uri/;
```

Если доменному имени соответствует несколько адресов, то все они будут использоваться по очереди (`round-robin`). Кроме того, в качестве адреса можно указать группу серверов.

В значении параметра можно использовать переменные. В этом случае, если адрес указан в виде доменного имени, поиск имени производится среди описанных групп серверов. В случае если имя не найдено, оно определяется с помощью директивы `resolver`.

`URI` запроса передаётся на сервер следующим образом:

- если директива `proxy_pass` указана с `URI`, то при передаче запроса серверу часть нормализованного `URI` запроса, соответствующая `location`, заменяется на `URI`, указанный в директиве. Например:

```
location /name/ {
```

```
proxy_pass http://127.0.0.1/remote/;
}
```

- если директива `proxy_pass` указана без `URI`, то при обработке первоначального запроса на сервер передаётся URI запроса в том же виде, в каком его прислал клиент, а при обработке изменённого `URI` – нормализованный `URI` запроса целиком:

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

В следующих случаях часть `URI` запроса, подлежащую замене, выделить невозможно:

- если `location` задан регулярным выражением, а также в именованных `location`. В этих случаях `proxy_pass` следует указывать без `URI`.
- если внутри проксируемого `location` с помощью директивы `rewrite` изменяется `URI`, и именно с этой конфигурацией будет обрабатываться запрос (`break`):

```
location /name/ {
rewrite    /name/([^/]+) /users?name=$1 break;
proxy_pass http://127.0.0.1;
}
```

  В этом случае `URI`, указанный в директиве, игнорируется, и на сервер передаётся изменённый `URI` запроса целиком.
- при использовании переменных в `proxy_pass`:

```
location /name/ {
proxy_pass http://127.0.0.1$request_uri;
}
```

  В этом случае, если в директиве указан `URI`, он передаётся на сервер как есть, заменяя `URI` первоначального запроса.

`proxy_set_header`

Позволяет переопределять или добавлять поля заголовка запроса, передаваемые проксируемому серверу. В качестве значения можно использовать текст, переменные и их комбинации. Директивы наследуются от родительской конфигурации при условии, что в дочерней не описаны свои директивы `proxy_set_header`.

Синтаксис:

```
proxy_set_header <поле> <значение>;
```

По умолчанию переопределяются только два поля:

```
proxy_set_header Host $proxy_host;
proxy_set_header Connection close;
```

Если включено кеширование, поля заголовка `If-Modified-Since`, `If-Unmodified-Since`, `If-None-Match`, `If-Match`, `Range` и `If-Range` исходного запроса не пе-

редаются на проксируемый сервер.

Неизменённое поле заголовка запроса `Host` можно передать следующим образом:

```
proxy_set_header Host $http_host;
```

Однако, если это поле отсутствует в заголовке запроса клиента, то ничего передаваться не будет. В этом случае лучше воспользоваться переменной `$host` – её значение равно имени сервера в поле `Host` заголовка запроса, или же основному имени сервера, если поля нет:

```
proxy_set_header Host $host;
```

Кроме того, можно передать имя сервера вместе с портом проксируемого сервера:

```
proxy_set_header Host $host:$proxy_port;
```

Если значение поля заголовка — пустая строка, данное поле не будет передаваться проксируемому серверу:

```
proxy_set_header Accept-Encoding "";
```

`proxy_buffering`

Разрешает или запрещает использовать буферизацию ответов проксируемого сервера.

Синтаксис:

```
proxy_buffering on | off;
```

Значение по умолчанию:

```
proxy_buffering on;
```

Если буферизация включена, то NGINX принимает ответ проксируемого сервера как можно быстрее, сохраняя его в буферы, заданные директивами `proxy_buffer_size` и `proxy_buffers`. Если ответ не вмещается целиком в память, то его часть может быть записана на диск во временный файл. Запись во временные файлы контролируется директивами `proxy_max_temp_file_size` и `proxy_temp_file_write_size`.

Если буферизация выключена, то ответ синхронно передаётся клиенту сразу же по мере его поступления. NGINX не пытается считать весь ответ проксируемого сервера. Максимальный размер данных, который NGINX может принять от сервера за один раз, задаётся директивой `proxy_buffer_size`.

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

`proxy_buffer_size`

Задает `<размер>` буфера, в который будет читаться первая часть ответа, получаемого от проксируемого сервера. В этой части ответа обычно находится небольшой заголовок ответа.

Синтаксис:

```
proxy_buffer_size <размер>;
```

По умолчанию размер одного буфера равен размеру страницы памяти. В зависимости от платформы – 4K или 8K, однако его можно сделать меньше.

`proxy_buffers`

Задаёт <число> и <размер> буферов для одного соединения, в которые будет читаться ответ, получаемый от проксируемого сервера.

Синтаксис:

```
proxy_buffers <число> <размер>;
```

По умолчанию размер одного буфера равен размеру страницы. В зависимости от платформы – 4K или 8K.

`proxy_max_temp_file_size`

Если включена буферизация ответов проксируемого сервера, и ответ не вмещается целиком в буферы, заданные директивами `proxy_buffer_size` и `proxy_buffers`, часть ответа может быть записана во временный файл. Директива задаёт максимальный <размер> временного файла. Размер данных, сбрасываемых во временный файл за один раз, задаётся директивой `proxy_temp_file_ write_size`.

Синтаксис:

```
proxy_max_temp_file_size <размер>;
```

значение по умолчанию:

```
proxy_max_temp_file_size 1024m;
```

Значение 0 отключает возможность буферизации ответов во временные файлы.

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

`proxy_temp_file_write_size`

Ограничивает <размер> данных, сбрасываемых во временный файл за один раз, при включённой буферизации ответов проксируемого сервера во временные файлы.

Синтаксис:

```
proxy_temp_file_write_size <размер>;
```

Значение по умолчанию:

```
proxy_temp_file_write_size 8k|16k;
```

По умолчанию <размер> ограничен двумя буферами, заданными директивами `proxy_buffer_size` и `proxy_buffers`. Максимальный размер временного файла задаётся директивой `proxy_max_temp_file_size`.

`proxy_ignore_headers`

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

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

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

`proxy_cache`

Задаёт зону разделяемой памяти, используемой для кеширования. Одна и та же зона может использоваться в нескольких местах. В значении параметра можно использовать переменные.

Синтаксис:

```
proxy_cache <зона> | off;
```

По умолчанию параметр установлен в значение `off`, которое запрещает кеширование, унаследованное от родительского уровня конфигурации.

`proxy_redirect`

Задаёт текст, который нужно изменить в полях заголовка `Location` и `Refresh` в ответе проксируемого сервера.

Синтаксис:

```
proxy_redirect default;
proxy_redirect off;
proxy_redirect <перенаправление> <замена>;
```

Например, проксируемый сервер вернул поле заголовка:

```
Location: http://localhost:8000/two/some/uri/
```

Директива:

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

перепишет эту строку в виде:

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

В заменяемой строке можно не указывать имя сервера:

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

тогда будут подставлены основное имя сервера и порт, если он отличен от `80`.

Стандартная замена, задаваемая параметром `default`, использует параметры

директив `location` и `proxy_pass`. Поэтому две нижеприведённые конфигурации одинаковы:

```
location /one/ {
   proxy_pass http://upstream:port/two/;
   proxy_redirect default;

location /one/ {
   proxy_pass http://upstream:port/two/;
   proxy_redirect http://upstream:port/two/ /one/;
```

Примечание. Параметр `default` недопустим, если в `proxy_pass` используются переменные.

В строке `<замена>` можно использовать переменные. Например:

```
proxy_redirect http://localhost:8000/ http://$host:$server_port/;
```

В строке `<перенаправление>` тоже можно использовать переменные. Например:

```
proxy_redirect http://$proxy_host:8000/ /;
```

Директиву также можно задать при помощи регулярных выражений. При этом `<перенаправление>` должно начинаться либо с символа `~`, если при сравнении следует учитывать регистр символов, либо с символов `~*`, если регистр символов учитывать не нужно. Регулярное выражение может содержать именованные и позиционные выделения, а `<замена>` ссылаться на них:

```
proxy_redirect ~^(http://[^:]+):\d+(/.+)$ $1$2;
proxy_redirect ~*/user/([^/]+)/(.+)$ http://$1.example.com/$2;
```

На одном уровне может быть указано несколько директив `proxy_redirect`:

```
proxy_redirect default;
proxy_redirect http://localhost:8000/ /;
proxy_redirect http://www.example.com/ /;
```

Если к полям заголовка в ответе проксируемого сервера могут быть применены несколько директив, будет выбрана первая из них. Параметр `off` отменяет действие унаследованных с родительского уровня конфигурации директив `proxy_redirect`.

С помощью этой директивы можно также добавлять имя хоста к относительным перенаправлениям, выдаваемым проксируемым сервером. Например:

```
proxy_redirect / /;
```

2.6.2 Модуль `ngx_http_uwsgi_module`

Модуль `ngx_http_uwsgi_module` позволяет передавать запросы `uWSGI`-серверу.

`uWSGI`-сервер — это сервер приложений, который реализует спецификацию `WSGI` (Web Server Gateway Interface) для Python-приложений. Он служит промежуточным звеном между веб-сервером (NGINX) и веб-приложением, позволяя обрабатывать

запросы и отправлять ответы.

Когда uWSGI используется вместе с NGINX, NGINX выступает в роли обратного прокси-сервера, который принимает HTTP-запросы от клиентов и передает их на uWSGI, который, в свою очередь, запускает Python-приложение. Это позволяет разделить обработку статического контента (который может обслуживаться NGINX) и динамического контента (который обрабатывается uWSGI и запущенным приложением).

Далее будут рассмотрены основные директивы модуля ngx_http_uwsgi_module.

uwsgi_pass_request_body

Позволяет запретить передачу исходного тела запроса на uwsgi-сервер.

Синтаксис:

```
uwsgi_pass_request_headers on | off;
```

Значение по умолчанию on.

uwsgi_pass_request_headers

Позволяет запретить передачу полей заголовка исходного запроса на uwsgi-сервер.

Синтаксис:

```
uwsgi_pass_request_headers on | off;
```

Значение по умолчанию on.

uwsgi_pass

Задаёт протокол и адрес uwsgi-сервера. В качестве протокола можно указать uwsgi или suwsgi (secured uwsgi, uwsgi через SSL).

Синтаксис:

```
uwsgi_pass [<протокол>://]<адрес>;
```

Адрес может быть указан в виде доменного имени или IP-адреса и порта, например:

```
uwsgi_pass localhost:9000;
uwsgi_pass uwsgi://localhost:9000;
uwsgi_pass suwsgi://[2001:db8::1]:9090;
```

или в виде пути UNIX-сокета:

```
uwsgi_pass unix:/tmp/uwsgi.socket;
```

uwsgi_param

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

Синтаксис:

```
uwsgi_param <параметр> <значение> [if_not_empty];
```

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

```
uwsgi_param HTTPS $https if_not_empty;
```

`uwsgi_read_timeout`

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

Синтаксис:

```
uwsgi_read_timeout <время>;
```

Значение по умолчанию `60s` (60 секунд).

2.6.3 Модуль `ngx_http_fastcgi_module`

Модуль `ngx_http_fastcgi_module` — это модуль для веб-сервера NGINX, который позволяет взаимодействовать с `FastCGI`-приложениями.

`FastCGI` — это протокол, который используется для связи веб-сервера с приложениями, написанными на языках программирования, таких как `PHP`, `Python`, `Ruby` и других.

Далее будут рассмотрены основные директивы модуля `ngx_http_fastcgi_module`.

`fastcgi_pass`

Задаёт адрес `FastCGI`-сервера.

Синтаксис:

```
fastcgi_pass <адрес>;
```

Адрес может быть указан в виде доменного имени или `IP`-адреса и порта, например:

```
fastcgi_pass localhost:9000;
```

или в виде пути `UNIX`-сокета, например:

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

Если доменному имени соответствует несколько адресов, то все они будут использоваться по очереди (`round-robin`). Кроме того, в качестве адреса можно указать группу серверов.

В значении параметра можно использовать переменные. В этом случае, если адрес указан в виде доменного имени, поиск имени производится среди описанных групп серверов. В случае если имя не найдено, оно определяется с помощью директивы `resolver`.

`fastcgi_pass_request_body`

Позволяет запретить передачу исходного тела запроса на `FastCGI`-сервер.

Синтаксис:

```
fastcgi_pass_request_body on | off;
```

Значение по умолчанию `on`.

`fastcgi_pass_request_headers`

Позволяет запретить передачу полей заголовка исходного запроса на `FastCGI`-сервер.

Синтаксис:

```
fastcgi_pass_request_headers on | off;
```

Значение по умолчанию `on`.

`fastcgi_param`

Задаёт `<параметр>`, который будет передаваться `FastCGI`-серверу. В качестве значения можно использовать текст, переменные и их комбинации. Директивы наследуются от родительского уровня конфигурации при условии, что на дочернем уровне не описаны свои директивы `fastcgi_param`.

Синтаксис:

```
fastcgi_param <параметр> <значение> [if_not_empty];
```

Далее приведён пример минимально необходимых параметров для `PHP`:

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

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

Если скрипты обрабатывают запросы `POST`, необходимо указать ещё три параметра:

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

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

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

`fastcgi_read_timeout`

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

Синтаксис:

```
fastcgi_read_timeout <время>;
```

Значение по умолчанию `60s` (60 секунд).

2.6.4 Модуль ngx_http_scgi_module

Модуль ngx_http_scgi_module позволяет передавать запросы SCGI-серверу.

Далее будут рассмотрены основные директивы модуля ngx_http_scgi_module.

scgi_pass

Задаёт адрес SCGI-сервера.

Синтаксис:

```
scgi_pass <адрес>;
```

Адрес может быть указан в виде доменного имени или IP-адреса и порта, например:

```
scgi_pass localhost:9000;
```

или в виде пути UNIX-сокета:

```
scgi_pass unix:/tmp/scgi.socket;
```

Если доменному имени соответствует несколько адресов, то все они будут использоваться по очереди (round-robin). Кроме того, в качестве адреса можно указать группу серверов.

В значении параметра можно использовать переменные. В этом случае, если адрес указан в виде доменного имени, поиск имени производится среди описанных групп серверов. В случае если имя не найдено, оно определяется с помощью директивы resolver.

scgi_pass_request_body

Позволяет запретить передачу исходного тела запроса на SCGI-сервер.

Синтаксис:

```
scgi_pass_request_body on | off;
```

Значение по умолчанию on.

scgi_pass_request_headers

Позволяет запретить передачу полей заголовка исходного запроса на SCGI-сервер.

Синтаксис:

```
scgi_pass_request_headers on | off;
```

Значение по умолчанию on.

scgi_param

Задаёт <параметр>, который будет передаваться SCGI-серверу. В качестве значения можно использовать текст, переменные и их комбинации. Директивы наследуются от родительского уровня конфигурации при условии, что на дочернем уровне не описаны свои директивы scgi_param.

Синтаксис:

```
scgi_param <параметр> <значение> [if_not_empty];
```

Стандартные переменные окружения CGI должны передаваться как заголовки SCGI:

```
location / {
   include scgi_params;
   ...
}
```

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

```
scgi_param HTTPS https if_not_empty;
```

scgi_read_timeout

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

Синтаксис:

```
scgi_read_timeout время;
```

Значение по умолчанию 60s (60 секунд).

2.6.5 Модуль ngx_http_memcached_module

Модуль ngx_http_memcached_module предоставляет возможность кеширования данных с использованием memcached. Модуль позволяет NGINX взаимодействовать с сервером memcached для хранения и извлечения данных, что может значительно улучшить производительность веб-приложений, уменьшая нагрузку на серверы приложений и базы данных.

Далее будут рассмотрены основные директивы модуля ngx_http_memcached_module.

memcached_pass

Задаёт адрес сервера memcached.

Синтаксис:

```
memcached_pass <адрес>;
```

Адрес может быть указан в виде доменного имени или IP-адреса и порта, например:

```
memcached_pass localhost:11211;
```

или в виде пути UNIX-сокета, например:

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

Если доменному имени соответствует несколько адресов, то все они будут использоваться по очереди (round-robin). Кроме того, в качестве адреса можно указать группу серверов.

`memcached_buffer_size`

Задаёт `<размер>` буфера, в который будет читаться ответ, получаемый от сервера `memcached`. Ответ синхронно передаётся клиенту сразу же по мере его поступления.

Синтаксис:

```
memcached_buffer_size <размер>;
```

По умолчанию размер одного буфера равен размеру страницы. В зависимости от платформы – `4K` или `8K`.

`memcached_read_timeout`

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

Синтаксис:

```
memcached_read_timeout <время>;
```

Значение по умолчанию `60s` (60 секунд).

Пример конфигурации

Пример конфигурации для настройки обращения к серверу `memcached`:

```
server {
   location / {
      set         $memcached_key "$uri?$args";
      memcached_pass host:11211;
      error_page   404 502 504 = @fallback;
   }

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

2.6.6 Проксирование TCP/UDP-трафика

Модуль `ngx_stream_proxy_module` позволяет проксировать `TCP`- и `UDP`-трафик. Он предназначен для работы в контексте стриминга, что означает, что он может обрабатывать потоки данных на уровне транспортного протокола (например, `TCP` и `UDP`), а не на уровне `HTTP`.

Далее будут рассмотрены основные директивы модуля `ngx_stream_proxy_ module`.

`proxy_pass`

Задаёт адрес проксируемого сервера.

Синтаксис:

```
proxy_pass <адрес>;
```

Адрес может быть указан в виде доменного имени или IP-адреса, и порта:

```
proxy_pass localhost:12345;
```

или в виде пути UNIX-сокета:

```
proxy_pass unix:/tmp/stream.socket;
```

Если доменному имени соответствует несколько адресов, то все они будут использоваться по очереди (round-robin). Кроме того, в качестве адреса можно указать группу серверов.

В значении параметра можно использовать переменные, например:

```
proxy_pass $upstream;
```

В этом случае, если адрес указан в виде доменного имени, поиск имени производится среди описанных групп серверов. В случае если имя не найдено, оно определяется с помощью директивы resolver.

`proxy_buffer_size`

Задаёт размер буфера, в который будут читаться данные, получаемые от проксируемого сервера. Также задаёт размер буфера, в который будут читаться данные, получаемые от клиента.

Синтаксис:

```
proxy_buffer_size <размер>;
```

По умолчанию размер буфера равен 16K.

`proxy_ssl`

Включает протоколы SSL/TLS для соединений с проксируемым сервером.

Синтаксис:

```
proxy_ssl on | off;
```

По умолчанию параметр имеет значение off.

`proxy_ssl_certificate`

Задаёт файл с сертификатом в формате PEM для аутентификации на проксируемом сервере.

Синтаксис:

```
proxy_ssl_certificate <файл>;
```

`proxy_ssl_certificate_key`

Задаёт файл с секретным ключом в формате PEM для аутентификации на проксируемом сервере.

Синтаксис:

```
proxy_ssl_certificate_key <файл>;
```

`proxy_timeout`

Задаёт таймаут между двумя идущими подряд операциями чтения или записи на клиентском соединении или соединении с проксируемым сервером. Если по истечении этого времени данные не передавались, соединение закрывается.

Синтаксис:

```
proxy_timeout <время>;
```

По умолчанию таймаут равен `10m` (10 минут).

2.7 Распределение нагрузки

С понятием проксирования тесно связан модуль `ngx_http_upstream_module`. Директива `upstream` начинает новый контекст, в котором определяется группа проксируемых серверов.

Далее будут рассмотрены основные директивы модуля `ngx_http_upstream_ module`.

2.7.1 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 хэширования `IP`-адреса, активируемый директивой `ip_hash`, основан на 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 распределние нагрузки между проксируемыми 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.

2.7.2 Основные директивы модуля распределения нагрузки

`upstream`

Описывает группу серверов. Серверы могут слушать на разных портах. Кроме того, можно одновременно использовать серверы, слушающие на `TCP`- и `UNIX`-сокетах.

Синтаксис:

```
upstream <название> {
...
}
```

Пример конфигурации:

```
upstream backend {
   server backend1.example.com weight=5;
   server 127.0.0.1:8080   max_fails=3 fail_timeout=30s;
   server unix:/tmp/backend3;

   server backup1.example.com backup;
}
```

По умолчанию запросы распределяются по серверам циклически (в режиме `round-robin`) с учётом весов серверов. В вышеприведённом примере каждые 7 запросов будут распределены следующим образом: 5 запросов на `backend1.example.com` и по одному запросу на второй и третий серверы. Если при попытке работы с сервером происходит ошибка, то запрос передаётся следующему серверу, и так далее до тех пор, пока не будут опробованы все работающие серверы. Если не удастся получить успешный ответ ни от одного из серверов, то клиенту будет возвращён результат работы с последним сервером.

`server`

Задаёт адрес и другие параметры сервера. Адрес может быть указан в виде доменного имени или IP-адреса и необязательного порта, а также в виде пути UNIX-сокета, который указывается после префикса `unix:`. Если порт не указан, по умолчанию используется порт 80. Доменное имя, которому соответствует несколько IP-адресов, задаёт сразу несколько серверов.

Синтаксис:

```
server <адрес> [<параметры>];
```

К необязательным параметрам проксируемого сервера относятся:

- `weight=<число>` – относительный вес сервера;
- `max_fails=<число>` – максимальное число неудачных попыток установления связи с сервером в течение времени fail_timeout, после которого сервер помечается как недоступный;
- `fail_timeout=<время>` – время, в течение которого сервер должен ответить на запрос, и время, в течение которого сервер считается недоступным;
- `backup` – помечает сервер как запасной сервер, на него будут передаваться запросы в случае, если не работают основные серверы;
- `down` – помечает сервер как непригодный для обработки запросов;
- `resolve` – отслеживает изменения IP-адресов, соответствующих доменному имени сервера, и автоматически изменяет конфигурацию группы без необходимости перезапуска службы `nginx`;
- `service=<имя>` – включает преобразование SRV-записей DNS и задаёт имя сервиса.

`least_conn`

Задаёт для группы метод балансировки нагрузки, при котором запрос передаётся серверу с наименьшим числом активных соединений, с учётом весов серверов. Если подходит сразу несколько серверов, они выбираются циклически (в режиме `round-robin`) с учётом их весов.

`ip_hash`

Задаёт для группы метод балансировки нагрузки, при котором запросы распределяются по серверам на основе `IP`-адресов клиентов. В качестве ключа для хэширования используются первые три октета `IPv4`-адреса клиента или `IPv6`-адрес клиента целиком. Метод гарантирует, что запросы одного и того же клиента будут всегда передаваться на один и тот же сервер. Если же данный сервер будет считаться недоступным, то запросы такого клиента будут передаваться на другой сервер. С большой долей вероятности это также будет один и тот же сервер.

Если один из серверов нужно убрать на некоторое время, то для сохранения текущего хэширования `IP`-адресов клиентов этот сервер нужно пометить параметром `down`.

Например:

```
upstream backend {
   ip_hash;

   server backend1.example.com;
   server backend2.example.com;
   server backend3.example.com down;
   server backend4.example.com;
}
```

`keepalive`

Задействует кеш соединений для группы серверов.

Синтаксис:

```
keepalive <соединения>;
```

Параметр соединения устанавливает максимальное число неактивных постоянных соединений с серверами группы, которые будут сохраняться в кеше каждого рабочего процесса. При превышении этого числа наиболее давно не используемые соединения закрываются.

Примечание. Директива `keepalive` не ограничивает общее число соединений с серверами группы, которые рабочие процессы NGINX могут открыть. Параметр `<соединения>` следует устанавливать так, чтобы серверы группы по-прежнему могли обрабатывать новые входящие соединения.

Примечание. При использовании методов балансировки нагрузки, отличных от стандартного `round-robin`, следует активировать их до директивы `keepalive`.

2.7.3 Кеширование соединений

Директива `keepalive` показывает, сколько соединений с проксируемым сервером NGINX должен держать открытыми в каждом рабочем процессе. Кеш соединений

полезен в случае, когда NGINX должен 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.

Для соединений по HTTP директиву proxy_http_version следует установить в значение 1.1, а поле заголовка Connection — очистить, например:

```
upstream http_backend {    server 127.0.0.1:8080;
   keepalive 16;
}

server {
   ...
   location /http/ {
      proxy_pass http://http_backend;
      proxy_http_version 1.1;
      proxy_set_header Connection "";
      ...
   }
}
```

В приведенной конфигурации указано, что с сервером http_backend, работающем на порту 8080 сервера localhost, может быть открыто 16 соединений,

NGINX должен будет выполнить процедуру квитирования TCP только для первых 16 соединений в каждом рабочем процессе, а затем может оставить эти соединения открытыми, не отправляя заголовок Connection со значением close.

В директиве proxy_http_version указывается, что работа с проксируемым сервером будет выполняться по протоколу HTTP/1.1. Значение заголовка Connec- tion очищается с помощью директивы proxy_set_header.

Если для обслуживания запросов потребуется больше 16 соединений, NGINX их откроет, но по достижении порогового значения NGINX будет закрывать редко используемые соединения, чтобы общее число открытых соединений оставалось равным 16, как указано в директиве keepalive.

Данный механизм применим и для проксирования соединений по протоколам, отличным от HTTP. В примере ниже говорится, что NGINX должен 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;
}
```

2.8 Обработка и настройка файлов журналов

За ведение журналов HTTP-запросов отвечает модуль ngx_http_log_module. Он позволяет настраивать формат и место хранения журналов, а также управлять их содержимым, что помогает в мониторинге, отладке и анализе работы веб-сервера. Журналы описываются в контексте location.

2.8.1 Настройка формата журналов

Для определения формата ведения журнала используется директива log_format. Синтаксис:

```
log_format <название> [escape=default|json|none] <строка> ...;
```

Параметр escape позволяет задать экранирование символов json или default в переменных. По умолчанию используется default. Значение none отключает экранирование символов.

При использовании default символы ", \, а также символы со значениями меньше 32 или больше 126 экранируются как \xXX. Если значение переменной не найдено, то в качестве значения в журнал будет записываться дефис (-).

При использовании json экранируются все символы, недопустимые в JSON-строках: символы " и \ экранируются как \" и \\, символы со значениями меньше 32 экранируются как \n, \r, \t, \b, \f или \u00XX.

Строки заголовка, переданные клиенту, начинаются с префикса sent_http_, например, $sent_http_content_range.

В конфигурации всегда существует предопределённый формат combined:

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

2.8.2 Директива error_log

Директива error_log конфигурирует запись в журнал. На одном уровне конфигурации может использоваться несколько журналов. Если на уровне конфигурации main запись события в файл явно не задана, то используется файл по умолчанию.

Синтаксис:

```
error_log <файл> [<критичность>];
```

Первый параметр задаёт `<файл>`, который будет хранить журнал. Специальное значение `stderr` выбирает стандартный файл ошибок.

Второй параметр определяет `<критичность>` события и может принимать одно из следующих значений: `debug`, `info`, `notice`, `warn`, `error`, `crit`, `alert` или `emerg`. Уровни критичности событий перечислены в порядке возрастания. При установке определённого уровня в журнал попадают все сообщения указанного уровня, а также сообщения более высоких уровней. Например, при стандартном уровне критичности `error` в журнал попадают сообщения уровней `error`, `crit`, `alert` и `emerg`. Если параметр не задан, по умолчанию используется `error`.

2.8.3 Буферизованная запись в журнал

Директива `access_log` задаёт путь, формат и настройки буферизованной записи в журнал. На одном уровне конфигурации может использоваться несколько файлов журналов. Запись в `syslog` настраивается указанием префикса `syslog:` в первом параметре. Специальное значение `off` отменяет все директивы `access_log` для текущего уровня. Если формат не указан, то используется предопределённый формат `combined`.

Если задан размер буфера с помощью параметра `buffer` или указан параметр `gzip`, то запись будет буферизованной.

Примечание. Размер буфера должен быть не больше размера атомарной записи в дисковый файл.

При включённой буферизации данные записываются в файл:

- если очередная строка журнала не помещается в буфер;
- если данные в буфере находятся дольше интервала времени, заданного параметром `flush`;
- при переоткрытии файла журнала или завершении рабочего процесса.

Если задан параметр `gzip`, то буфер будет сжиматься перед записью в файл. Степень сжатия может быть задана в диапазоне от 1 (быстрое сжатие) до 9 (качественное сжатие). По умолчанию используется буфер размером `64K6` и степень сжатия `1`. Данные сжимаются атомарными блоками, и в любой момент времени файл журнала может быть распакован или прочитан с помощью утилиты `zcat`.

Пример конфигурации:

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

2.8.4 Ротация журналов

Для настройки ротации файлы журналов необходимо сначала переименовать, а затем передать сигнал `USR1` главному процессу. Он откроет заново все текущие открытые файлы и назначит им в качестве владельца непривилегированного пользователя, с правами которого запущены рабочие процессы. После этого главный процесс закрывает все открытые файлы и посылает сообщение о переоткрытии файлов рабочим процессам. Они также открывают новые файлы и сразу же закрывают старые. В

результате старые файлы становятся готовы для дальнейшей обработки, например, для сжатия.

3. Сервер приложений Tomcat

3.1 Общие сведения

Программное обеспечение Apache Tomcat (далее Tomcat) представляет собой среду выполнения сервлетов, реализующую спецификации Java Servlet, JavaServer Pages (JSP) и Java WebSocket, предоставляя таким образом платформу для запуска веб-приложений, написанных на языке Java.

Java Servlet – это серверный компонент, который обрабатывает запросы и генерирует ответы в формате `HTML` или других форматах. Сервлеты могут взаимодействовать с клиентами через `HTTP`.

Среда выполнения сервлетов обеспечивает управление жизненным циклом сервлетов, обработку запросов и управление сессиями.

JavaServer Pages (JSP) – технология, позволяющая создавать динамические веб-страницы с использованием Java. JSP-файлы компилируются в сервлеты, что позволяет интегрировать Java-код в `HTML`-код.

Java WebSocket – протокол, который обеспечивает двунаправленную полнодуплексную связь клиент-сервер в реальном времени. Он основан на протоколе TCP и предназначен для обмена сообщениями между браузером и веб-сервером.

В отличие от традиционного `HTTP`, который следует модели «запрос – ответ», WebSockets позволяют свободно обмениваться данными в обоих направлениях. Такой подход делает WebSockets наиболее подходящими для приложений, которые требуют мгновенных обновлений, например, чатов, онлайн-игр и уведомлений в реальном времени.

Tomcat упрощает разработку и развертывание веб-приложений. Он проверяет каталоги проекта для поиска и включения в работу новых сервлетов. Если во время выполнения приложения сервлет обновляется, Tomcat обновляет все зависимые компоненты. Если сервлет удаляется, сервер останавливает обработку запросов и удаляет элемент. В качестве среды выполнения сервлетов Tomcat выполняет несколько важных функций:

- создание среды для запуска сервлетов;
- указание параметров сессий;
- обеспечение обмена данными между сервлетами и клиентами;
- идентификация клиентских подключений;
- управление метаданными.

Tomcat позволяет веб-приложениям использовать Java для создания динамичных веб-страниц. Tomcat также может быть настроен и использован как самостоятельный веб-сервер, обрабатывающий как статические страницы, так и динамические запросы

через Servlets и JSP.

Часто Tomcat используется в сочетании с традиционными веб-серверами (такими как Apache HTTP Server или NGINX) для обработки статического контента, в то время как динамический контент обрабатывается через Tomcat.

3.1.1 Компоненты Tomcat

Tomcat состоит из нескольких ключевых компонентов:

- Catalina – основной компонент Tomcat, среда выполнения сервлетов Tomcat, которая реализует спецификацию Servlet API. Servlet API является основой для всех остальных технологий Java, касающихся `Web`, и предоставляет возможность динамически генерировать любой веб-контент, используя любые библиотеки, доступные для `java`.
- Coyote – компонент стека `HTTP` Tomcat, который поддерживает протокол `HTTP` для веб-серверов или среды выполнения приложений. Coyote прослушивает входящие соединения на определённом `TCP`-порту сервера, пересылает запросы в механизм Tomcat для обработки запросов и возвращает ответ запрашивающему клиенту.
- Jasper – механизм JSP Tomcat, который является реализацией спецификации JavaServer Pages 2.0 Sun Microsystems. Jasper анализирует JSP-файлы, чтобы компилировать их в Java-код как сервлеты (которые могут быть обработаны с помощью Catalina). Во время выполнения Jasper может автоматически обнаруживать изменения JSP-файла и перекомпилировать его.
- Context – представляет собой отдельное веб-приложение внутри Tomcat. Каждый Context содержит настройки конкретного приложения. Включает параметры конфигурации, такие как пути к ресурсам, параметры безопасности и настройки сессий. Контексты могут быть определены в файле `context.xml` или в файле `server.xml`.
- Realm – компонент безопасности, который управляет аутентификацией и авторизацией пользователей. Realm определяет, как Tomcat проверяет учетные данные пользователей, а также определяет права доступа пользователей к различным ресурсам приложения.
- Logger – система ведения журналов в Tomcat. Она отвечает за запись различных событий, таких как ошибки, предупреждения и информация о работе сервера. Ведение журналов настраивается через файл `logging.properties` или с использованием сторонних библиотек журналирования.

3.1.2 Переменные окружения

В Tomcat используются две важные переменные окружения – `CATALINA_HOME` и `CATALINA_BASE`.

Переменная `CATALINA_HOME`

Переменная `CATALINA_HOME` указывает на корневую директорию установки Tomcat – директория, где находятся все файлы сервера, включая библиотеки, конфигурационные файлы и исполняемые файлы.

К основным файлам сервера Tomcat относятся:

- `/bin` — скрипты для запуска и остановки сервера;
- `/lib` — библиотеки, необходимые для работы Tomcat;
- `/conf` — конфигурационные файлы (например, `server.xml`, `web.xml` и другие).

Пример установки переменной `CATALINA_HOME`:

```
echo "export CATALINA_HOME='/usr/share/tomcat/'" >> ~/.bashrc
source ~/.bashrc
```

Переменная `CATALINA_BASE`

Переменная `CATALINA_BASE` указывает на базовую директорию для конкретного экземпляра Tomcat, тем самым позволяя запускать несколько экземпляров Tomcat с одной и той же установкой, используя разные конфигурации и приложения.

Переменную `CATALINA_BASE` необходимо установить до запуска Tomcat.

Переменная используется для определения местоположения следующих директорий:

- `/webapps` — директория, где размещаются веб-приложения;
- `/logs` — директория для журналов;
- `/conf` — конфигурационные файлы, специфичные для этого экземпляра;
- `/temp` и `/work` — временные директории.

Если переменная `CATALINA_BASE` не установлена, Tomcat будет использовать значение переменной `CATALINA_HOME` как значение по умолчанию.

Пример установки переменной `CATALINA_BASE`:

```
echo "export CATALINA_BASE='/usr/share/tomcat/'" >> ~/.bashrc
source ~/.bashrc
```

При использовании в системе нескольких экземпляров Tomcat переменная `CATALINA_HOME` содержит статические данные, такие как файлы `.jar` или бинарные файлы, а переменная `CATALINA_BASE` содержит файлы конфигурации, журналов, развернутые приложения и другие необходимые ресурсы (то, что необходимо сохранить при обновлении приложения).

Преимущества использования `CATALINA_BASE`

По умолчанию переменные `CATALINA_HOME` и `CATALINA_BASE` указывают на один и тот же каталог.

В случае необходимости запуска нескольких экземпляров Tomcat на одной машине, следует вручную указать переменной `CATALINA_BASE` путь к экземпляру конфигурации Tomcat.

Такой подход обеспечивает следующие преимущества:

- простота обновления – поскольку все экземпляры Tomcat с одним расположением `CATALINA_HOME` совместно используют один набор файлов `.jar` и двоичных файлов, обновление файлов до новой версии и применение изменений ко всем экземплярам Tomcat может быть произведено с использованием одного и того же каталога `CATALINA_HOME`;
- отсутствие дублирования одних и тех же статических файлов `.jar`;
- возможность совместного использования определенных настроек, например, `setenv`.

Рекомендации по использованию `CATALINA_BASE`

Перед началом использования `CATALINA_BASE` рекомендуется создать необходимое для дальнейшей работы дерево каталогов.

> **Примечание**. Дерево каталогов может быть создано автоматически в процессе установки Tomcat. Однако в случае, если по каким-либо причинам дерево каталогов не будет создано автоматически (например, по причине отсутствия необходимых разрешений), установленный экземпляр Tomcat может не запуститься или будет работать неправильно.

Рекомендации к созданию дерева каталогов Tomcat:

- каталог `/lib` с ресурсами для добавления в `classpath` – рекомендуется, если приложение зависит от внешних библиотек. Сначала проверяется значение переменной `CATALINA_BASE`, затем загружается `CATALINA_HOME`;
- каталог `/logs` для файлов журнала – рекомендуется;
- каталог `/webapps` для автоматически загружаемых веб-приложений – рекомендуется, если планируется развертывание веб-приложений. Проверяется только значение переменной `CATALINA_BASE`;
- каталог `/work`, содержащий временные рабочие каталоги для развернутых веб-приложений – рекомендуется;
- каталог `/temp`, используемый JVM (Java Virtual Machine) для временных файлов – рекомендуется.

В Tomcat за ведение внутреннего журналирования отвечает файл `tomcat-juli.jar`. Оригинальный файл `tomcat-juli.jar` изменять не рекомендуется. Если необходимо самостоятельно настроить ведение журналов, следует внести правки в файлах, указанных в переменной `CATALINA_BASE` для конкретного экземпляра Tomcat.

Также рекомендуется скопировать все файлы конфигурации из каталога `$CATALINA_HOME/conf/` в каталог `$CATALINA_BASE/conf/`. Обратите внимание, что если в каталоге `$CATALINA_BASE` отсутствует какой-либо файл конфигурации, а резервного варианта в каталоге `$CATALINA_HOME` нет, – это может привести к сбою в работе Tomcat.

Для корректного функционирования Tomcat каталог `$CATALINA_BASE` должен содержать следующие файлы:

- `/conf/server.xml` – главный файл конфигурации, позволяющий настраивать различные компоненты среды выполнения Tomcat;
- `/conf/web.xml` – файл конфигурации, отвечающий за описание классов, ресурсов и настройки приложения, а также за то, как сервер будет использовать их для выполнения веб-запросов.

3.2 Установка и запуск Tomcat

Установка веб-сервера Tomcat производится командой:

```
dnf install tomcat java-1.8.0-openjdk
```

Перед запуском службы Tomcat рекомендуется настроить переменную среды `CATALINA_HOME` или `CATALINA_BASE`. Подробнее о настройке переменной среды см. в п. 3.1.2 «Переменные окружения».

После установки запустите и добавьте в автозагрузку службу `tomcat`:

```
systemctl enable --now tomcat
```

Проверьте статус службы:

```
systemctl status tomcat
```

В статусе должно отображаться `active (running)`.

Для обеспечения безопасности и изоляции процессов работы во время установки сервера приложений Tomcat в системе создается пользователь `tomcat`, с правами которого будет производиться запуск необходимых служб сервера приложений. Информацию о пользователе можно просмотреть, выполнив команду:

```
id tomcat
```

```
uid=53(tomcat) gid=53(tomcat) группы=53(tomcat)
```

По умолчанию доступ к веб-интерфейсу Tomcat будет осуществляться по адресу `http://localhost:8080/`.

3.3 Файлы конфигурации

После запуска Tomcat необходимо настроить его файлы конфигурации, исходя из целей, для которых будет использоваться Tomcat. Далее приведены основные конфигурационные файлы Tomcat, расположенные в каталоге `$CATALINA_BASE/conf`.

3.3.1 Файл `server.xml`

Файл `server.xml` является главным файлом конфигурации Tomcat и содержит описание основных компонентов сервера. Настройка атрибутов файла позволяет точно определить функциональные возможности сервера.

Далее будут рассмотрены основные секции, необходимые для корректного функционирования сервера.

Connectors

Секция `<Connectors>` отвечает за настройку точек подключения, которые обрабатывают входящие запросы от клиентов. Каждый коннектор может использовать разные протоколы (например, `HTTP`, `HTTPS`, `AJP`) и может быть настроен с различными параметрами для оптимизации работы сервера.

Структура секции <Connectors> имеет следующий вид:

```
<Connector port="8080" protocol="HTTP/1.1"
  connectionTimeout="20000"
  maxThreads="200"
  redirectPort="8443"
  />
```

где:

- port – порт, на котором коннектор будет слушать входящие запросы. В примере используется стандартный порт для HTTP – 8080;
- protocol – протокол, который будет использоваться для обработки запросов:
  - HTTP/1.1 – для HTTP-запросов;
  - org.apache.coyote.http11.Http11NioProtocol – для использования NIO (неблокирующий ввод-вывод);
  - AJP/1.3 – для протокола AJP (Apache JServ Protocol).
- connectionTimeout – время ожидания (в миллисекундах) для установления соединения. В примере – 20000 мс или 20 сек;
- maxThreads – максимальное количество потоков, используемых для обработки входящих запросов. В примере – 200, что позволяет одновременно обрабатывать до 200 запросов;
- redirectPort – порт, на который будет перенаправлен трафик, если запрос поступает по незащищённому протоколу (например, HTTP), но требуется переключение на защищённый (например, HTTPS). В примере перенаправление производится на порт 8443.

Listener

Секция <Listener> используется для регистрации объектов, которые могут реагировать на события жизненного цикла различных компонентов сервера. Каждая секция <Listener> позволяет выполнять определенные действия при возникновении событий, таких как инициализация сервера, развертывание веб-приложений, завершение работы и другие.

Структура секции <Listener> имеет следующий вид:

```
  Listener className="org.apache.catalina.core.StandardServer$Stan-
dardServerLifecycleListener" />
```

Каждая секция <Listener> имеет свои специфические атрибуты и классы. К самым распространенным классам Listener относятся:

- LifecycleListener – позволяет выполнять действия при изменении состояния жизненного цикла компонента. Например, можно использовать его для выполнения инициализации или освобождения ресурсов;
- ContextConfig – отвечает за конфигурацию контекста приложения при его загрузке;
- JmxRemoteLifecycleListener – позволяет включить поддержку JMX (Java Management Extensions) для удаленного мониторинга и управления Tomcat;
- MemoryLeakPreventionListener – помогает предотвратить утечки памяти, освобождая ресурсы при остановке сервера;
- SessionIdGeneratorBase – может использоваться для генерации уникальных

идентификаторов сессий.

Context

Секция `<Context>` определяет контекст для конкретного веб-приложения. Контекст — это рабочая область приложения, он служит для настройки параметров, специфичных для этого приложения. Каждый контекст может иметь свои собственные настройки, такие как параметры конфигурации, ресурсы, доступные для приложения, и поведение сервера.

Структура секции `<Context>` имеет следующий вид:

```
   <Context path="/myapp" docBase="myapp.war" reloadable="true"
crossContext="false">
       <Resource ...>
       <Valve ...>
   </Context>
```

где:

- `path="/myapp"` – определяет путь, по которому будет доступно приложение;
- `docBase="myapp.war"` – указывает на `WAR`-файл приложения, который будет развернут, также можно указать каталог с развернутым приложением;
- `reloadable="true"` – указывает на необходимость перезагрузки контекста при изменении классов в приложении;
- `crossContext="false"` – указывает на запрет доступа к другим контекстам;
- `<Resource>` – вложенный элемент, который определяет ресурс (например, `DataSource`), доступный для приложения;
- `<Valve>` – вложенный элемент, который добавляет валидацию для журналирования доступа к этому контексту.

Valve

Секция `<Valve>` используется для добавления компонентов, которые могут обрабатывать запросы и ответы на уровне сервера или конкретного контекста. Валидация (`Valve`) предоставляет возможности для журналирования, аутентификации, контроля доступа и других аспектов обработки `HTTP`-запросов.

Структура секции `<Valve>` имеет следующий вид:

```
   <Valve className="org.apache.catalina.valves.AccessLogValve"
          directory="logs" prefix="localhost_access_log."
          suffix=".txt"
          pattern="%h %l %u %t &amp;quot;%r&amp;quot; %s %b" />
```

где:

- `directory` – директория, в которой будут храниться файлы журнала;
- `prefix` – префикс имени файла журнала;
- `suffix` – суффикс имени файла журнала (например, `.log`);
- `pattern` – шаблон форматирования записей журнала. Можно использовать различные переменные, например:
  - `%h` – `IP`-адрес клиента,
  - `%l` – логин клиента,
  - `%u` – имя пользователя,
  - `%t` – время запроса,

– %r – запрос,
– %s – код состояния ответа,
– %b – размер ответа.

Host

Секция <Host> определяет виртуальный хост, который может обслуживать один или несколько веб-приложений. Виртуальные хосты позволяют одному экземпляру сервера Tomcat обрабатывать запросы для нескольких доменных имен или приложений.

Структура секции <Host> имеет следующий вид:

```
<Host name="example.com" appBase="webapps"
      unpackWARs="true" autoDeploy="true"
      deployOnStartup="true">
    <Context ... />
    <Valve ... />
</Host>
```

где:

- name – уникальное имя виртуального хоста, которое используется для сопоставления с запросами, поступающими на сервер;
- appBase – указывает базовый каталог, в котором находятся приложения, развернутые для данного хоста. По умолчанию каталог /webapps, однако его можно изменить на любой другой;
- unpackWARs – указывает, следует ли распаковывать WAR-файлы в директорию временных файлов при развертывании. Если установлено значение true, WAR-файлы будут распакованы;
- autoDeploy – определяет, будет ли Tomcat автоматически развертывать новые приложения, найденные в каталоге /webapps, при старте сервера или добавлении новых WAR-файлов;
- deployOnStartup – определяет, будет ли Tomcat автоматически развертывать приложения, находящиеся в каталоге /webapps, при запуске сервера;
- backgroundProcessorDelay – задержка (в миллисекундах) между циклами фоновой обработки, может быть полезно для управления производительностью сервера;
- <Context> – вложенный элемент, который определяет контекст приложения;
- <Valve> – вложенный элемент, который добавляет валв для журналирования доступа к этому хосту.

Realm

Секция <Realm> используется для настройки механизмов аутентификации и авторизации пользователей. Секции <Realm> позволяют Tomcat взаимодействовать с различными источниками данных для проверки учетных данных пользователей, такими как базы данных, LDAP-серверы и другие.

Секции <Realm> могут быть добавлены в различные уровни конфигурации Tomcat:

- глобальная конфигурация (<Engine>) – секции <Realm>, добавленные на этом уровне, будут применяться ко всем приложениям на сервере;
- конфигурация конкретного хоста (<Host>) – секции <Realm>, добавленные в секцию <Host>, будут применяться только к приложениям, развернутым на

этом хосте;
- конфигурация конкретного контекста (`<Context>`) – секции `<Realm>`, добавленные в секцию `<Context>`, будут применяться только к конкретному веб-приложению.

Для секции `<Realm>` могут быть использованы следующие механизмы аутентификации и авторизации:
- `MemoryRealm` – используется для аутентификации с помощью встраиваемых учетных записей пользователей и ролей в памяти;
- `UserDatabaseRealm` – предназначен для аутентификации и авторизации пользователей и использует ресурс `JNDI` для хранения информации о пользователях;
- `JNDIRealm` – позволяет аутентифицировать пользователей через `LDAP` или другие `JNDI`-совместимые источники;
- `DataSourceRealm` – позволяет аутентифицировать пользователей с использованием пула соединений (`DataSource`) для подключения к базе данных.

Подробную информацию о доступных механизмах аутентификации и авторизации пользователей см. в подразделе 3.7 «Механизмы аутентификации и авторизации пользователей»

Engine

Компонент `<Engine>` отвечает за обработку запросов и управление контекстами приложений. Он является частью более общей структуры, состоящей из `<Server>` и `<Host>`, и служит для настройки поведения всего сервера приложений.

Структура компонента `<Engine>` имеет следующий вид:

```
<Engine name="Catalina" defaultHost="localhost" jvmRoute="jvm1">
    <Host>
      ...
    </Host>
</Engine>
```

где:
- `name` – уникальное имя для конкретного экземпляра `<Engine>`. Обычно используется для идентификации в файлах журналов и при отладке;
- `defaultHost` – указывает имя хоста, который будет использоваться по умолчанию, если запрашиваемый хост не найден. Может быть полезен, если настроено несколько виртуальных хостов;
- `jvmRoute` – параметр используется в кластерах для обеспечения "sticky sessions", позволяет серверу идентифицировать, к какому экземпляру приложения принадлежит сессия пользователя;
- `<Host>` – определяет виртуальный хост, который может обслуживать один или несколько веб-приложений.

Service

Секция `<Service>` определяет группу коннекторов, которые управляют входящими запросами, и одну или несколько сред выполнения, которые обрабатывают запросы. В данной секции доступно также конфигурирование виртуальных хостов и контекстов приложений.

Структура секции `<Service>` имеет следующий вид:

```
<Service name="Catalina">
  <Connector_1... />
  <Connector_2... />
  <Engine ...>
      <Host...>
          <Context... />
      </Host>
  </Engine>
</Service>
```

где:

- `name` – уникальное имя сервиса; в примере Catalina, но доступно указание любого имени;
- `<Connector>` – определяет точки подключения, которые обрабатывают входящие запросы. Каждый коннектор может использовать разные протоколы (например, `HTTP`, `HTTPS`, `AJP`) и настраиваться с различными параметрами;
- `<Engine>` – элемент, который управляет обработкой запросов для всех коннекторов в рамках данного сервиса;
- `<Host>` – определяет виртуальный хост, который будет обрабатывать запросы для данного сервиса;
- `<Context>` – определяет контекст приложения (веб-приложения), которое будет развернуто в данном виртуальном хосте.

Server

Секция `<Server>` является корневым элементом конфигурации сервера. Она определяет основные параметры и настройки для всего экземпляра Tomcat. Внутри данной секции можно настраивать различные компоненты, такие как `port`, `listener`, `host` и пр.

Структура секции `<Server>` имеет следующий вид:

```
<Server port="8005" shutdown="SHUTDOWN">
    <!-- Слушатели (Listeners) -->
    <Listener className="..."/>

    <!-- Хосты (Hosts) -->
    <Host name="localhost" appBase="webapps" unpackWARs="true"
          autoDeploy="true">

        <!-- Контексты (Contexts) -->
        <Context path="/myapp" docBase="myapp.war" />
    </Host>

    <!-- Другие элементы -->
</Server>
```

где:

- `port` – указывает номер порта, на котором сервер будет ожидать команды завершения работы. По умолчанию – `8005`. Команда завершения работы отправляется

с помощью специального сообщения, и если сервер получает его, он начинает процесс остановки;
- `shutdown` – определяет строку, которую сервер будет ожидать для завершения работы. По умолчанию – `"SHUTDOWN"`. Если сервер получает эту строку на порту, указанном в атрибуте `port`, он инициирует процесс остановки;
- `<Listener>` – позволяет регистрировать объекты, которые могут реагировать на события жизненного цикла сервера, такие как инициализация, остановка и развертывание приложений;
- `<GlobalNamingResources>` – может использоваться для определения глобальных ресурсов, таких как JDBC DataSource, которые могут быть доступны всем приложениям в среде выполнения;
- `<Host>` – определяет виртуальный хост, который может содержать несколько веб-приложений. Каждый хост может иметь свои настройки, такие как `appBase` (папка с приложениями), `unpackWARs` (разворачивать ли `WAR`-файлы) и `autoDeploy` (автоматически развертывать приложения);
- `<Engine>` – определяет механизм обработки запросов для данного хоста. Обычно используется по умолчанию и может содержать настройки для управления сессиями и обработкой запросов;
- `<Realm>` – определяет механизм аутентификации и авторизации пользователей для приложения;
- `<Valve>` – позволяет добавлять дополнительные фильтры или обработчики для обработки запросов и ответов.

3.3.2 Файл `web.xml`

Файл `web.xml` является основным файлом конфигурации для веб-приложений, развернутых на сервере приложений Apache Tomcat. Данный файл описывает настройки приложения, такие как сервлеты, фильтры, параметры конфигурации и маппинг `URL`. Он находится в каталоге `/WEB-INF` веб-приложения.

Далее приведено описание основных элементов и атрибутов, используемых в файле конфигурации `web.xml`:
- `<web-app>` – корневой элемент файла `web.xml`, который содержит следующие атрибуты:
  - `xmlns` – пространство имен `XML` для спецификации;
  - `version` – версия спецификации Servlet API (например, 3.1).
- `<display-name>` – имя веб-приложения;
- `<servlet>` – сервлет, который будет обрабатывать запросы:
  - `<servlet-name>` – уникальное имя сервлета;
  - `<servlet-class>` – полное имя класса сервлета (например, `com.example.MyServlet`);
  - `<init-param>` – параметры инициализации для сервлета;
  - `<param-name>` – имя параметра;
  - `<param-value>` – значение параметра;
  - `<load-on-startup>` – определяет порядок загрузки сервлета при старте приложения (значение `1` или больше означает, что сервлет будет загружен при старте).
- `<servlet-mapping>` – определяет, какие `URL` будут обрабатываться данным сервлетом:
  - `<url-pattern>` – шаблон `URL`, который будет направлять запросы к соответ-

ствующему сервлету.

- `<filter>` – определяет фильтр, который может изменять запросы и ответы:
  - `<filter-name>` – уникальное имя фильтра;
  - `<filter-class>` – полное имя класса фильтра.
- `<filter-mapping>` – определяет, какие `URL` будут обрабатываться данным фильтром:
  - `<url-pattern>` – шаблон `URL` для фильтра (например, `/*` для всех запросов).
- `<welcome-file-list>` – определяет список файлов приветствия, которые сервер будет искать, если запрашивается корневой `URL` приложения:
  - `<welcome-file>` – название файла приветствия (например, `index.html`, `index.jsp`).
- `<error-page>` – определяет страницы ошибок для обработки различных кодов состояния `HTTP`:
  - `<error-code>` – код ошибки (например, `404`).
  - `<location>` – `URL` страницы для отображения при возникновении указанной ошибки.
- `<session-config>` – конфигурация сеансов для вашего веб-приложения:
  - `<session-timeout>` – время ожидания сеанса в минутах.
- `<context-param>` – определяет параметры контекста, доступные всему приложению.
- `<param-name>` и `<param-value>` – имя и значение параметра контекста.

3.3.3 Файл `tomcat-users.xml`

Файл `tomcat-users.xml` используется для управления пользователями и ролями, которые могут аутентифицироваться на сервере. Файл `tomcat-users.xml` позволяет задавать учетные записи пользователей, а также определять их роли, такие как администраторы или пользователи, что позволяет контролировать доступ к различным функциям и ресурсам сервера.

Файл `tomcat-users.xml` по умолчанию расположен в каталоге `$CATALINA_BASE/conf`.

Структура файла `tomcat-users.xml` имеет примерно следующий вид:

```
<?xml version="1.0" encoding="UTF-8"?>
<tomcat-users>
    <role rolename="manager-gui"/>
    <role rolename="manager-script"/>
    <role rolename="admin-gui"/>
    <role rolename="admin-script"/>

    <user username="admin" password="admin123"
      roles="manager-gui,admin-gui"/>
    <user username="deployer" password="deploy456"
      roles="manager-script"/>
</tomcat-users>
```

где:

- `<tomcat-users>` – корневой элемент файла, который содержит все определения пользователей и ролей;
- `<role rolename>` – определяет роль, которую может иметь пользователь, используется для определения прав доступа пользователя;

- `<user>` – определяет пользователя, который может аутентифицироваться на сервере:
  - `username` – имя пользователя;
  - `password` – пароль пользователя. Обратите внимание, что хранение паролей в открытом виде не рекомендуется. Рекомендуется использование более безопасных методов хранения паролей.
  - `roles` – роли, присвоенные пользователю (несколько значений указываются через запятую).

3.3.4 Файл `catalina.policy`

Файл `catalina.policy` используется для настройки политик безопасности, которые определяют, какие действия могут выполняться приложениями, развернутыми на сервере. Файл `catalina.policy` позволяет управлять доступом к различным ресурсам и операциям, таким как доступ к файловой системе, сетевым ресурсам и прочим системным ресурсам.

Файл `catalina.policy` по умолчанию расположен в каталоге `$CATALINA_BASE/conf`.

Структура файла `catalina.policy` имеет примерно следующий вид:

```
// Разрешения для всех приложений
grant {

   // Разрешение на чтение и запись в домашний каталог пользова-
   теля
   permission java.io.FilePermission "${user.home}/-", "read,write";

   // Разрешение на выполнение сетевых операций
   permission java.net.SocketPermission "*", "connect,resolve";

   // Разрешение на доступ к системным свойствам
   permission java.util.PropertyPermission "*", "read,write";
};

// Разрешения для конкретного приложения
grant codeBase "file:${catalina.base}/webapps/myapp/-" {

   // Разрешение на чтение файлов в каталоге приложения
   permission java.io.FilePermission "${catalina.base}/webapps/
myapp/-", "read";

   // Разрешение на выполнение операций с сокетами
   permission java.net.SocketPermission "localhost:8080",
"connect";
};
```

где:

- `grant` – блок, определяющий набор разрешений. Внутри этого блока указываются разрешения, которые будут применяться к указанной кодовой базе;
- `codeBase` – указывает путь к кодовой базе (например, директории приложения

или JAR-файла), для которой применяются разрешения. Путь может содержать шаблоны (например, * для обозначения всех файлов);

- `permission` – команда, определяющая конкретное разрешение. Синтаксис команды имеет вид:

```
permission <permission-class> "<target>", "<actions>";
```

где:

  - `<permission-class>` – класс разрешения (например, `java.io.FilePermission`, `java.net.SocketPermission`, `java.util.PropertyPemission` и т. д.);
  - `<target>` – цель разрешения (например, путь к файлу или адрес сокета);
  - `<actions>` – действия, разрешенные для данной цели (например, `read`, `write`, `connect`, `delete` и т.д.).

3.3.5 Файл `catalina.properties`

Файл `catalina.properties` содержит различные настройки, которые влияют на поведение сервера. Файл по умолчанию расположен в каталоге `$CATALINA_BASE/conf`.

Структура файла `catalina.properties` имеет примерно следующий вид:

```
# Каталоги
catalina.base=${catalina.home}
catalina.home=/path/to/tomcat

# Журналирование
java.util.logging.manager=org.apache.juli.ClassLoaderLogManager
java.util.logging.config.file=${catalina.base}/conf/logging.pro-
perties

# Загрузчики классов
shared.loader=${catalina.base}/lib/*.jar

# Другие настройки
...
```

К основным параметрам файла catalina.properties относятся:

- `catalina.base` – базовый каталог для текущего экземпляра Tomcat;
- `catalina.home` – корневой каталог установки Tomcat;
- `java.util.logging.manager` – класс, который будет использоваться для управления журналированием, по умолчанию – `org.apache.juli.ClassLoaderLogManager`;
- `java.util.logging.config.file` – путь к файлу конфигурации журналирования, по умолчанию `/conf/logging.properties`;
- `shared.loader` – дополнительные JAR-файлы или классы, которые будут загружены в общий класс-путь (`shared classpath`);
- `loader.repos` – репозитории загрузчиков классов;
- `webapp.loader` – загрузчик классов для веб-приложений.

3.3.6 Файл `context.xml`

Файл `context.xml` определяет контексты для веб-приложений. Он может использоваться для настройки различных параметров, таких как источники данных (`DataSource`), параметры безопасности, настройки кеширования и прочие специфичные для приложения параметры.

Файл `context.xml` может располагаться в нескольких местах:

- в каталоге `/META-INF` веб-приложения – для применения настроек к конкретному приложению, вы можете разместить файл `context.xml` в папке `/META-INF` вашего `WAR`-файла;
- в каталоге `$CATALINA_BASE/conf` установленного экземпляра Tomcat – для применения настроек ко всем приложениям на сервере.

Структура файла `context.xml` имеет примерно следующий вид:

```
<Context>

    <!-- Настройка источника данных -->
    <Resource name="jdbc/MyDB"
              auth="Container"
              type="javax.sql.DataSource"
              maxTotal="100"
              maxIdle="30"
              maxWaitMillis="10000"
              username="dbuser"
              password="dbpassword"
              driverClassName="com.mysql.cj.jdbc.Driver"
              url="jdbc:mysql://localhost:3306/mydb"/>

    <!-- Параметры приложения -->
    <Parameter name="configParam" value="configValue"
               override="false"/>

    <!-- Ограничения безопасности -->
    <SecurityConstraint>
        <WebResourceCollection>
            <ResourceDescription>Protected Area</ResourceDescrip-
            tion>
            <UrlPattern>/secure/*</UrlPattern>
        </WebResourceCollection>
        <AuthConstraint>
            <RoleName>admin</RoleName>
        </AuthConstraint>
    </SecurityConstraint>

    <!-- Настройки кеширования -->
    <Resources cachingAllowed="true" cacheMaxSize="100000" />

</Context>
```

где:

- `<Context>` – корневой элемент файла `context.xml`. Все настройки должны быть помещены внутри этого элемента.
- `<Resource>` – элемент, необходимый для подключения к базе данных и позволяющий Tomcat управлять соединениями с базой данных. Может включать в себя следующие атрибуты:
  - `name` – имя ресурса, которое будет использоваться в `JNDI`;
  - `auth` – определяет способ аутентификации (по умолчанию `"Container"`);
  - `type` – тип ресурса (например, `javax.sql.DataSource`);
  - `maxTotal` – максимальное количество соединений;
  - `maxIdle` – максимальное количество бездействующих соединений;
  - `maxWaitMillis` – максимальное время ожидания соединения;
  - `username`, `password` – учетные данные для подключения к базе данных;
  - `driverClassName` – класс драйвера `JDBC`;
  - `url` – `URL` для подключения к базе данных.
- `<Parameter>` – параметры конфигурации веб-приложения. Может включать в себя следующие атрибуты:
  - `name` – имя параметра;
  - `value` – значение параметра;
  - `override` – указывает, может ли значение быть переопределено в других конфигурациях.
- `<SecurityConstraint>` – параметры безопасности веб-приложения. Может включать в себя следующие элементы:
  - `<WebResourceCollection>` – определяет ресурсы, которые будут защищены;
  - `<AuthConstraint>` – определяет роли, необходимые для доступа к защищенным ресурсам.
- `<Resources>` – параметры кеширования. Может включать в себя следующие атрибуты:
  - `cachingAllowed` – разрешает или запрещает кеширование;
  - `cacheMaxSize` – максимальный размер кеша в байтах.

3.4 Настройка портов для приложений

3.4.1 Смена стандартного порта

По умолчанию Tomcat прослушивает соединения на порту `8080`. Однако в случае возникновения конфликтов с другими сервисами, требованиями безопасности или для запуска нескольких экземпляров Tomcat на одном сервере, порт можно изменить.

Для этого необходимо открыть файл `$CATALINA_BASE/conf/server.xml`:

```
nano $CATALINA_BASE/conf/server.xml
```

и в секции `<Connector>` изменить номер порта `8080` на нужный:

```
<Connector port="8080" protocol="HTTP/1.1"
    connectionTimeout="20000"
    redirectPort="8443"
    maxParameterCount="1000"
/>
```

Также необходимо открыть указанный порт для соединений и перезапустить конфигурацию `firewalld`:

```
firewall-cmd --permanent --add-port=8080/tcp
firewall-cmd --reload
```

Подробную информацию о настройке `firewalld` см. в п. «Брандмауэр Firewall» Руководства администратора. Часть 1.

Затем сохранить внесенные изменения и перезапустить службу `tomcat`:

```
systemctl restart tomcat
```

После выполненной настройки доступ к веб-интерфейсу Tomcat будет осуществляться по адресу `http://localhost:<номер_порта>`.

3.4.2 Настройка доступа к приложениям на нескольких портах

Настройка доступа к приложениям на нескольких портах может потребоваться в следующих случаях:

- для разделения трафика – доступно использование разных портов для разных типов трафика (например, `HTTP` и `HTTPS`) или для различных приложений, работающих на одном сервере;
- для обеспечения безопасности – разные порты могут использоваться для изоляции приложений или сервисов, что позволит повысить безопасность;
- для проверки работы приложения – в рамках разработки и тестирования работы приложения может потребоваться запуск нескольких экземпляров приложения на разных портах;
- для обслуживания – можно использовать один порт для обслуживания, а другой – для разработки или тестирования.

Для выполнения настройки доступности приложений на нескольких портах необходимо открыть файл `$CATALINA_BASE/conf/server.xml`:

```
nano $CATALINA_BASE/conf/server.xml
```

и в секции `<Service>` добавить дополнительную секцию `<Connector>` с нужным номером порта, например:

```
<Connector port="8081" protocol="HTTP/1.1"
               connectionTimeout="20000"
               redirectPort="8443" />
```

где:

- `port="8081"` – новый порт, на котором будет слушать сервер;
- `protocol="HTTP/1.1"` – протокол, который будет использоваться;
- `connectionTimeout` – время ожидания соединения (в секундах);
- `redirectPort` – порт для перенаправления соединения при использовании `HTTPS`.

Также необходимо открыть указанный порт для соединений и перезапустить конфигурацию `firewalld`:

```
firewall-cmd --permanent --add-port=8081/tcp
firewall-cmd --reload
```

Подробную информацию о настройке `firewalld` см. в п. «Брандмауэр Firewall» Руководства администратора. Часть 1.

Для применения внесенных изменений требуется перезапустить службу `tomcat`:

```
systemctl restart tomcat
```

После выполненной настройки доступ к веб-приложению будет осуществляться по одному из указанных портов по адресу `http://localhost:<номер_порта_веб-приложения>/<имя_веб-приложения>`.

3.5 Развертывание веб-приложений

Веб-приложение определяется как иерархия каталогов и файлов в стандартном формате. К такой иерархии можно получить доступ в ее "распакованном" виде, где каждый каталог и файл существуют в файловой системе отдельно, или в "упакованном" виде, известном как веб-архив или `WAR`-файл. Первый формат полезен при разработке, в то время как второй используется при распространении приложения для установки.

Каталог верхнего уровня иерархии веб-приложения также является корневым каталогом, хранящим файлы самого приложения. В данном каталоге размещаются `HTML`-файлы, JSP-страницы и прочие ресурсы (таблицы стилей (`CSS`), изображения, клиентские скрипты (`javascript`) и т.п.), которые составляют пользовательский интерфейс веб-приложения.

Когда системный администратор развертывает веб-приложение на определенном сервере, он назначает контекстный путь к приложению. Таким образом, если системный администратор назначит приложению контекстный путь `/catalog`, то `URI` запроса, ссылающийся на `/catalog/index.html`, извлечет файл `index.html` из корневого каталога, хранящего файлы приложения.

3.5.1 Структура каталогов веб-приложений

Для упрощения создания архива веб-приложения в требуемом формате рекомендуется расположить исполняемые файлы веб-приложения (то есть файлы, которые Tomcat фактически будет использовать при запуске приложения) в той же структуре, которая необходима для формата `WAR`. Таким образом, в корневом каталоге приложения будет организована следующая структура:

- `*.html`, `*.jsp` и т.д. – страницы `HTML` и JSP, а также другие файлы, которые должны быть доступны клиентскому браузеру (например, `JavaScript`, таблицы стилей, изображения и т.д.) для приложения. В более крупных приложениях допускается разделение данных файлов по иерархии подкаталогов, но для небольших приложений, как правило, проще поддерживать только один каталог для указанных файлов;
- `/WEB-INF/web.xml` – дескриптор развертывания веб-приложения – `XML`-файл, описывающий сервлеты и другие компоненты, из которых состоит приложе-

ние, а также любые параметры инициализации и ограничения безопасности, управляемые средой выполнения приложения, которые необходимо применить. Подробнее см. в п. 3.5.3 «Дескриптор развертывания веб-приложения»;

- `/WEB-INF/classes/` – каталог содержит все файлы Java-классов (и связанные с ними ресурсы), необходимые для приложения, включая как сервлетные, так и несервлетные классы, которые не объединены в JAR-файлы. Если используемые классы организованы в пакеты Java, это необходимо отразить в иерархии каталогов в разделе `/WEB-INF/classes/`. Например, класс Java с именем `com.mycompany.mypackage.MyServlet` должен храниться в файле с именем `/WEB-INF/classes/com/mycompany/mypackage/MyServlet. class`;
- `/WEB-INF/lib/` – каталог содержит файлы JAR (файлы классов Java и связанные с ними ресурсы), необходимые для приложения, такие как библиотеки классов сторонних производителей или драйверы `JDBC`.

Когда приложение устанавливается в Tomcat, классы в каталоге `/WEB-INF/classes/`, а также все классы в файлах JAR из каталога `/WEB-INF/lib/`, становятся видимыми для других классов в конкретном веб-приложении. Таким образом, для упрощения установки веб-приложения рекомендуется размещать все необходимые библиотечные классы в одном из этих каталогов.

3.5.2 Общие файлы библиотек

Tomcat поддерживает механизмы однократной установки библиотечных JAR-файлов (или распакованных классов) и делает их видимыми для всех установленных веб-приложений (без необходимости их включения в само веб-приложение). Стандартное расположение, используемое при установке Tomcat для общего кода, – каталог `$CATALINA_BASE/lib`. Размещенные в данном каталоге JAR-файлы видны как веб-приложениям, так и внутреннему коду Tomcat. Данный каталог рекомендуется использовать для установки драйверов `JDBC`, необходимых как для веб-приложения, так и для внутреннего использования Tomcat (например, для `DataSourceRealm`).

Стандартная установка Tomcat включает в себя большое количество предустановленных общих файлов библиотек, включая API-интерфейсы `Servlet 4.0` и `JSP 2.3`, которые являются основополагающими для написания сервлетов и JavaServer Pages.

3.5.3 Дескриптор развертывания веб-приложения

Файл `/WEB-INF/web.xml` содержит дескриптор развертывания веб-приложения. Файл представляет собой `XML`-документ и определяет все, что необходимо знать серверу о приложении (за исключением контекстного пути, который назначается системным администратором при развертывании приложения).

Файл `web.xml` содержит комментарии, описывающие назначение каждого включенного элемента, и доступен для редактирования дескриптора.

> **Примечание**. Спецификация сервлета включает в себя дескриптор типа документа (`DTD`) для дескриптора развертывания веб-приложения, и Tomcat применяет правила, определенные в нем, при обработке файла `/WEB-INF/web.xml` для приложения. Поэтому необходимо указать элементы конкретного дескриптора (такие как `<filter>`, `<servlet>` и `<servlet-mapping>`) в порядке, определенном в `DTD`.

3.5.4 Дескриптор контекста

Файл `/META-INF/context.xml` может использоваться для определения конкретных параметров конфигурации Tomcat, таких как журнал доступа, источники данных, конфигурация менеджера сеансов и пр. Данный `XML`-файл должен содержать один элемент `<Context>`, который является дочерним для элемента `<Host>`, соответствующего хосту, на котором развертывается веб-приложение.

3.5.5 Способы развертывания приложений

В приведенном ниже описании используется имя переменной `$CATALINA_BASE` для указания базового каталога. Если Tomcat для нескольких экземпляров не был настроен, то для `$CATALINA_BASE` будет установлено значение `$CATALINA_HOME`, в который был установлен Tomcat.

Веб-приложение должно быть развернуто в среде выполнения сервлетов. Веб-приложение может быть развернуто в Tomcat одним из следующих способов:

- путем копирования иерархии каталогов распакованного архива в каталог `$CATALINA_BASE/webapps/`. Tomcat назначит контекстный путь приложения на основе выбранного имени подкаталога. После установки или обновления приложения необходимо перезапустить Tomcat.
- путем копирования файла архива веб-приложения (`WAR`) в каталог `$CATALINA_BASE/webapps/`. При запуске Tomcat автоматически распакует архив веб-приложения и запустит приложение. Такой подход обычно используется для установки дополнительного приложения, предоставляемого сторонним поставщиком или внутренними разработчиками, в существующую установку Tomcat.

  > **Примечание**. При использовании такого подхода для дальнейшего обновления приложения необходимо заменить файл архива веб-приложения и удалить ранее распакованный каталог этого приложения. После этого следует перезапустить Tomcat для применения внесенных изменений.

- путем использования веб-приложения Tomcat Manager для развертывания и отмены развертывания веб-приложений на работающем сервере Tomcat без его перезапуска.

3.5.6 Пример развертывания веб-приложения

Далее будет рассмотрен пример развертывания веб-приложения с использованием файла архива `WAR`. Архив веб-приложения должен быть предварительно загружен на сервер.

Для развертывания веб-приложения необходимо переместить файл архива `WAR` в каталог `$CATALINA_BASE/webapps`. Например:

```
mv ./sample.war $CATALINA_BASE/webapps/
```

Tomcat автоматически распакует архив и развернет приложение.

Для запуска приложения в адресной строке браузера необходимо указать следующий `URL`-адрес (при условии, что Tomcat используется на стандартном порту `8080`) – `http://localhost:8080/sample`.

В случае если необходимо просто просмотреть содержимое архива, распаковать его можно с помощью команды:

```
jar -xvf ./sample.war
```

3.6 Настройка безопасного подключения

HTTPS – протокол для безопасной связи по компьютерной сети. Данные в протоколе HTTPS передаются поверх криптографических протоколов TLS. Основной задачей HTTPS является аутентификация посещаемого веб-сайта и защита конфиденциальности и целостности передаваемых данных.

Все браузеры имеют возможность взаимодействовать с защищенными веб-серверами с использованием протокола SSL. Однако браузеру и серверу требуется SSL-сертификат, чтобы иметь возможность установить безопасное соединение.

SSL-сертификат имеет пару ключей: открытый и закрытый ключ, которые работают в связке для обеспечения зашифрованного соединения.

3.6.1 Виды SSL-сертификатов

Глобально SSL-сертификаты можно разделить на два типа:

- *Самоподписанные* – такие сертификаты подписываются на сервере компании и создать его может любой пользователь, поэтому каждый раз при посещении сайта браузеры будут выдавать предупреждение о незащищённом соединении.
- *Подписанные в удостоверяющих центрах* – такие сертификаты получают в удостоверяющих центрах (УЦ, ЦС), их подлинность будет подтверждена во всех браузерах.

Подписанные в УЦ сертификаты разделяются ещё на три вида – по типу проверки данных:

- DV (domain validation) – шифрует данные, но не имеет сведений о компании. Это базовый сертификат, который гораздо дешевле, чем остальные, и подходит любым компаниям.
- OV (organization validation) – шифрует данные и знает о существовании компании, подойдет юридическим лицам.
- EV (extended validation) – необходим для обеспечения высокого уровня защиты. Для получения такого сертификата, компания должна пройти сложную проверку – подтвердить законность и предоставить права на домен.

Все перечисленные сертификаты обеспечивают надежность в шифровании при передаче данных от браузера к серверу. Однако существуют и другие, более специфичные сертификаты:

- Wildcard – объединяет домены с поддоменами, а также защищает соединение между ними.
- SAN – защищает только те домены, которые указаны в сертификате.

В рамках данного руководства для проверки настройки HTTPS-соединения с веб-приложением будет использоваться самоподписанный сертификат. Однако для корпоративного использования необходимо приобрести один из подтвержденных УЦ сертификатов.

3.6.2 Алгоритм создания безопасных соединений

Когда браузер пытается получить доступ к веб-сайту, защищенному SSL, также генерируется ключ сеанса – это временный ключ, который используется один раз в определённый период времени для шифрования и дешифрования данных, переданных между двумя сторонами. Таким образом, для настройки SSL-соединения используются три ключа: открытый, закрытый и сеансовый ключи.

Обеспечение безопасного подключения производится согласно следующему алгоритму:

1. Браузер подключается к веб-серверу (веб-сайту), защищенному SSL (https) и посылает запрос на идентификацию сервера.
2. Сервер отправляет копию своего SSL-сертификата, включая открытый ключ.
3. Браузер проверяет корневой сертификат в списке доверенных ЦС, а также информацию о том, что сертификат не продлевается, не возвращен и его общее имя действительно для веб-сайта, к которому он подключается. Если браузер доверяет сертификату, он создает, шифрует и отправляет обратно симметричный ключ сеанса с использованием открытого ключа сервера.
4. Сервер расшифровывает симметричный ключ сеанса с помощью своего закрытого ключа и отправляет обратно подтверждение, зашифрованное ключом сеанса, для запуска зашифрованного сеанса.
5. После этого сервер и браузер шифруют все переданные данные с помощью ключа сеанса.

3.6.3 Организация защищенных соединений в Tomcat

Важно отметить, что настройка Tomcat для использования защищенных сокетов обычно необходима только при его запуске в качестве самостоятельного веб-сервера.

Если запустить Tomcat в первую очередь в качестве среды выполнения Servlet/JSP за другим веб-сервером, таким как Apache или NGINX, обычно необходимо настроить основной веб-сервер для обработки SSL-соединений от пользователей.

Для настройки SSL веб-сервер должен иметь соответствующий сертификат для каждого внешнего интерфейса (IP-адреса), который принимает защищенные соединения. Такой сертификат должен быть криптографически подписан доверенной третьей стороной – центром сертификации (ЦС, CA, УЦ).

Java предоставляет относительно простой инструмент командной строки, называемый keytool, который позволяет создать самоподписанный сертификат. Самоподписанные сертификаты – это сертификаты, созданные пользователем и не подписанные доверенным центром сертификации, следовательно, такие сертификаты не гарантируют подлинность.

Важно! Обратите внимание, что самоподписанные сертификаты могут быть полезны для некоторых сценариев тестирования веб-приложений, однако они **не подходят** для корпоративного использования.

В рамках данного руководства для проверки настройки HTTPS-соединения с веб-приложением будет использоваться самоподписанный сертификат. Однако для корпоративного использования необходимо приобрести один из подтвержденных УЦ сертификатов. ■

3.6.4 Подготовка хранилища ключей сертификата

Tomcat работает только с хранилищами ключей формата JKS, PKCS11 или PKCS12.

Формат JKS – это стандартный формат Java KeyStore, созданный с помощью утилиты командной строки keytool. Утилита входит в состав JDK.

Формат PKCS12 является интернет-стандартом и может управляться с помощью OpenSSL.

Для создания нового хранилища JKS, содержащего один самоподписанный сертификат, выполните команду:

```
keytool -genkey -alias tomcat -keyalg RSA
```

Данная команда создаст новый файл .keystore в домашнем каталоге пользователя, с правами которого команда была запущена.

Для указания другого местоположения или имени файла, следует добавить параметр -keystore, а затем полный путь к файлу хранилища ключей:

```
keytool -genkey -alias tomcat -keyalg RSA \
        -keystore /<путь_к_файлу_keystore>
```

После выполнения команды будет предложено задать пароль к хранилищу ключей и указать общую информацию о сертификате (имя пользователя, информацию о компании и местоположении и т.д.).

Если настройка произведена верно, в каталоге появится файл хранилища ключей с сертификатом, который может использоваться сервером.

Для использования созданного хранилища ключей при настройке защищенных соединений необходимо назначить файлу соответствующие права доступа – назначить владельцем файла пользователя tomcat:

```
chown tomcat:tomcat /<путь_к_файлу_keystore>
```

После этого можно переходить к настройке файла конфигурации сервера веб-приложений Tomcat, подробную информацию см. в п. 3.6.8 «Настройка файла конфигурации».

3.6.5 Создание запроса на подпись сертификата

Для получения сертификата от выбранного центра сертификации необходимо создать запрос на подпись сертификата (CSR). Данный запрос будет использоваться ЦС для создания сертификата, который в дальнейшем будет идентифицировать веб-сайт как «безопасный».

Для создания запроса на подпись необходимо выполнить следующий алгоритм действий:

1. Создать локальный самоподписанный сертификат согласно п. 3.6.4 «Подготовка хранилища ключей сертификата»:

   ```
   keytool -genkey -alias tomcat -keyalg RSA \
           -keystore /<путь_к_файлу_keystore>
   ```

Примечание. В некоторых случаях для создания корпоративного сертификата потребуется указать домен веб-сайта (например, `www.example.com`) при запросе имени и фамилии.

2. Создать файл запроса `.csr`:

```
keytool -certreq -keyalg RSA -alias tomcat \
        -file certreq.csr \
        -keystore /<путь_к_файлу_keystore>
```

3. Сформированный файл запроса (в примере `certreq.csr`) необходимо отправить в ЦС (подробную информацию см. в официальной документации ЦС).
4. Получить сертификат.

3.6.6 Импорт сертификата в хранилище `keystore`

Полученный сертификат, подписанный ЦС, можно импортировать в локальное хранилище ключей `JKS`.

Для этого необходимо выполнить следующий алгоритм действий:

1. Импортировать в хранилище ключей корневой сертификат:

```
keytool -import -alias root \
        -keystore /<путь_к_файлу_keystore> \
        -trustcacerts -file /<путь_к_корневому_сертификату>
```

2. Импортировать в хранилище ключей полученный сертификат:

```
keytool -import -alias <имя_сертификата_в_хранилище> \
        -keystore /<путь_к_файлу_keystore> \
        -file <имя_сертификата>.crt
```

После подготовки хранилища ключей сертификатов необходимо внести соответствующие изменения в файл конфигурации `$CATALINA_BASE/conf/server.xml` для настройки соединения по защищенному протоколу. Подробную информацию см. в п. 3.6.8 «Настройка файла конфигурации».

3.6.7 Импорт сертификата в хранилище `PKCS12`

Для импорта существующего сертификата, подписанного собственным центром сертификации, в хранилище ключей `PKCS12` с помощью `OpenSSL`, выполните команду вида:

```
openssl pkcs12 -export -in mycert.crt -inkey mykey.key \
               -out mycert.p12 -name tomcat -CAfile myCA.crt \
               -caname root -chain
```

где:

- `-export` – экспорт хранилища ключей в файл;
- `-in mycert.crt` – существующий файл сертификата;
- `-inkey mykey.key` – существующий файл ключа;
- `-out mycert.p12` – имя вновь созданного хранилища ключей `PKCS12`;
- `-name tomcat` – имя для импортируемого сертификата;

- `-CAfile myCA.crt` – файл сертификата ЦС, выдавшего сертификат;
- `-caname root` – имя корневого сертификата ЦС.

Более подробную информацию по использованию `openssl` см. в справочной странице:

```
openssl --help
```

После подготовки хранилища ключей сертификатов необходимо внести соответствующие изменения в файл конфигурации `$CATALINA_BASE/conf/server.xml` для настройки соединения по защищенному протоколу. Подробную информацию см. в п. 3.6.8 «Настройка файла конфигурации».

3.6.8 Настройка файла конфигурации

Tomcat может использовать три различные реализации `SSL`:

- реализация `JSSE` предоставляется как часть среды выполнения Java – стандартная библиотека Java для работы с `SSL` и `TLS`, входит в состав JDK и предоставляет API для создания защищенных соединений;
- реализация `JSSE`, использующая `OpenSSL` – позволяет использовать `OpenSSL` в качестве движка для обработки `SSL/TLS` в приложениях Java;
- реализация `APR`, которая по умолчанию использует движок `OpenSSL` – библиотека, которая обеспечивает абстракцию платформы для приложений Apache, включая Tomcat, может использовать `OpenSSL` для реализации `SSL/TLS`.

Параметры, используемые для организации защищенных соединений, зависят от используемой реализации. Если настройка параметра `<Connector>` выполнена с указанием `generic protocol="HTTP/1.1"`, то реализация, используемая Tomcat, выбирается автоматически.

По умолчанию используется реализация Java `JSSE`. Далее будет рассмотрен пример настройки стандартной реализации `SSL` – Java `JSSE`.

После подготовки хранилища ключей сертификатов необходимо внести соответствующие изменения в файл конфигурации `$CATALINA_BASE/conf/server.xml` для настройки соединения по защищенному протоколу.

Для этого необходимо открыть файл `$CATALINA_BASE/conf/server.xml`:

```
nano $CATALINA_BASE/conf/server.xml
```

и внести в него следующие изменения:

- включить поддержку OpenSSL:

```
   <Listener className="org.apache.catalina.core.AprLifecycle-
Listener"
       SSLEngine="on" SSLRandomSeed="builtin" />
```

- настроить параметр <Connector> на обработку соединений по HTTPS:
  - секция <Connector> при использовании сертификата из хранилища ключей JKS будет иметь примерно следующий вид:

```
   <!-- Define an SSL Coyote HTTP/1.1 Connector on port
8443 -->
   <Connector
       protocol="org.apache.coyote.http11.Http11NioProtocol"
       port="8443" maxThreads="200"
       maxParameterCount="1000"
       scheme="https" secure="true" SSLEnabled="true"
       keystoreFile="/etc/tomcat/keystore.keystore"
       keystorePass="changeit"
       clientAuth="false" sslProtocol="TLS"/>
```

    где:
    * protocol="org.apache.coyote.http11.Http11NioProtocol" – протокол для обработки входящего трафика, в примере org.apache.coyote.http11.Http-11NioProtocol – неблокируемый коннектор Java NIO;
    * port="8443" – номер порта TCP/IP, на котором Tomcat будет прослушивать защищенные соединения. Значение можно заменить на любой номер порта;
    * maxThreads="200" – максимальное количество одновременных запросов, которые могут быть обработаны. Если параметр не указан, этот атрибут устанавливается равным 200;
    * maxParameterCount="1000" – максимальное общее количество параметров запроса (включая загруженные файлы), полученных из строки запроса. Значение меньше 0 означает отсутствие лимита. Если параметр не указан, используется значение по умолчанию 10000;
    * scheme="https" – имя протокола (для SSL – https, значение по умолчанию – http);
    * SSLEnabled="true" – используется для включения SSL-трафика на коннекторе;
    * keystoreFile="/etc/tomcat/keystore.keystore" – путь и имя ранее сформированного хранилища ключей, в котором хранится сертификат сервера и ключ для загрузки. По умолчанию путь к файлу – домашний каталог пользователя, запускающего Tomcat. Допускается использование относительных путей, они будут разрешены от переменной $CATALINA_BASE;
    * certificateKeystorePassword="changeit" – пароль для доступа к хранилищу ключей, содержащему закрытый ключ и сертификат сервера.
  - секция <Connector> при использовании сертификата из хранилища PKCS12 будет иметь примерно следующий вид:

```
   <!-- Define an SSL Coyote HTTP/1.1 Connector on port
8443 -->
   <Connector
       protocol="org.apache.coyote.http11.Http11NioProtocol"
       port="8443" maxThreads="200"
       maxParameterCount="1000"
```

```
        scheme="https" secure="true" SSLEnabled="true"
        SSLCertificateFile="/usr/local/ssl/server.crt"
        SSLCertificateKeyFile="/usr/local/ssl/server.pem"
        SSLVerifyClient="optional"
        SSLProtocol="TLSv1+TLSv1.1+TLSv1.2"/>
```

где:

* `protocol="org.apache.coyote.http11.Http11NioProtocol"` – протокол для обработки входящего трафика, в примере `org.apache.coyote.http11.Http11NioProtocol` – неблокируемый коннектор `Java NIO`;
* `port="8443"` – номер порта `TCP/IP`, на котором Tomcat будет прослушивать защищенные соединения. Значение можно заменить на любой номер порта;
* `maxThreads="200"` – максимальное количество одновременных запросов, которые могут быть обработаны. Если параметр не указан, этот атрибут устанавливается равным 200;
* `maxParameterCount="1000"` – максимальное общее количество параметров запроса (включая загруженные файлы), полученных из строки запроса. Значение меньше `0` означает отсутствие лимита. Если параметр не указан, используется значение по умолчанию `10000`;
* `scheme="https"` – имя протокола (для `SSL` – `https`, значение по умолчанию – `http`);
* `SSLEnabled="true"` – используется для включения `SSL`-трафика на коннекторе;
* `SSLCertificateFile="/usr/local/ssl/server.crt"` – путь и имя сертификата сервера. Допускается использование относительных путей, они будут разрешены от переменной `$CATALINA_BASE`;
* `SSLCertificateKeyFile="/usr/local/ssl/server.pem"` – путь и имя файла, содержащего закрытый ключ.

Для применения внесенных в файл конфигурации изменений необходимо перезапустить службу `tomcat`:

```
systemctl restart tomcat
```

Если настройка выполнена верно, доступ к веб-интерфейсу будет осуществляться по адресу `https://localhost:8443/`.

3.7 Механизмы аутентификации и авторизации пользователей

Apache Tomcat предоставляет несколько механизмов аутентификации и авторизации пользователей, которые позволяют безопасно управлять доступом к веб-приложениям. Основным компонентом, отвечающим за аутентификацию пользователей и определение их ролей, является `<Realm>`. Общая информация о компоненте `<Realm>` приведена в пп. «Realm».

3.7.1 Методы аутентификации

Аутентификация — это процесс проверки подлинности пользователя. Tomcat поддерживает следующие методы аутентификации:

- Basic Authentication (`BASIC`) — стандартный метод `HTTP`-аутентификации, при

котором имя пользователя и пароль передаются в заголовке HTTP. Данный метод рекомендуется использовать его только через HTTPS.

- Digest Authentication (DIGEST) — более безопасный метод, чем Basic Authentication, так как передает хеш пароля вместо самого пароля. Данный метод требует поддержки со стороны клиента.
- Form-based Authentication (FORM) — позволяет создавать пользовательские формы входа. Пользователи перенаправляются на страницу входа, где они должны ввести свои учетные данные. После успешной аутентификации пользователи перенаправляются на запрашиваемый ресурс.
- Client Certificate Authentication (CLIENT-CERT) — использует сертификаты X.509 для аутентификации клиентов. Данный метод обеспечивает наивысший уровень безопасности.

3.7.2 Способы авторизации

Авторизация — это процесс определения прав доступа пользователя к ресурсам веб-приложения после успешной аутентификации. Tomcat использует роли для управления доступом.

Роли могут быть назначены каждому пользователю. Роли определяются в файле конфигурации tomcat-users.xml и могут быть использованы для ограничения доступа к определенным ресурсам в веб-приложении. Подробную информацию о доступных ролях см в п. 3.3.3 «Файл tomcat-users.xml»

Также в файле web.xml можно определить ограничения безопасности (security constraints), которые указывают, какие роли имеют доступ к определенным URL-шаблонам. Например, можно ограничить доступ к определенному ресурсу только для пользователей с ролью admin.

3.7.3 Механизм MemoryRealm

Механизм MemoryRealm используется для аутентификации с помощью встраиваемых учетных записей пользователей и ролей в памяти. Может применяться для простых приложений или для разработки и тестирования.

Во время запуска механизм MemoryRealm загружает информацию обо всех пользователях и соответствующих им ролях из файла tomcat-users.xml.

Примечание. Обратите внимание, что внесенные в файл tomcat-users.xml изменения будут применены только после перезапуска службы tomcat.

Настройка работы механизма MemoryRealm осуществляется согласно следующему алгоритму действий:

1. Определение механизма авторизации и аутентификации пользователей в файле server.xml и области его действия. Подробную информацию о структуре файла server.xml см. в п. 3.3.1 «Файл server.xml».

Укажите название механизма и определите область его действия. В приведенном далее примере область действия указанного механизма распространяется на все веб-приложения:

```
<Engine name="Catalina" defaultHost="localhost">
    ...
    <Realm className="org.apache.catalina.realm.MemoryRealm" />
    ...
```

```
    </Engine>
```

2. Создание пользователей и назначение им необходимых ролей в файле `tomcat-users.xml`. Укажите учетные данные для пользователей и назначьте необходимые роли. Подробную информацию о структуре файла `tomcat-users.xml` см. в п. 3.3.3 «Файл tomcat-users.xml».

В приведенном далее примере настроены роли для пользователей `admin` и `user`:

```
<tomcat-users>
    <role rolename="admin"/>
    <role rolename="user"/>

    <user username="admin" password="adminpass" roles="admin"/>
    <user username="user" password="userpass" roles="user"/>
</tomcat-users>
```

3. Настройка ограничения доступа к определенным URL-адресам веб-прило- жения в файле `$CATALINA_BASE/webapps/<название_веб-приложения>/WEB-INF/web.xml`. Подробную информацию о структуре файла `web.xml` см. в п. 3.3.2 «Файл web.xml».

В приведенном далее примере конфигурации настроено ограничение доступа к ресурсам каталога `/protected/*` для всех пользователей, кроме пользователя с назначенной ролью `admin`. При этом при попытке получения доступа к защищенному ресурсу с учетными данными какого-либо другого пользователя будет выводиться заданная страница с сообщением об ошибке:

```
<web-app xmlns="http://xmlns.jcp.org/xml/ns/javaee"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/javaee
      http://xmlns.jcp.org/xml/ns/javaee/web-app_3_1.xsd"
      version="3.1">

  <security-constraint>
      <display-name>Protected Area</display-name>
      <web-resource-collection>
          <web-resource-name>Protected Area</web-resource-name>
          <url-pattern>/protected/*</url-pattern>
      </web-resource-collection>
      <auth-constraint>
          <role-name>admin</role-name>
      </auth-constraint>
  </security-constraint>

  <login-config>
      <auth-method>BASIC</auth-method>
      <realm-name>My Application</realm-name>
  </login-config>

  <error-page>
      <error-code>403</error-code>
```

```
        <location>/error.jsp</location>
    </error-page>

</web-app>
```

4. Для применения внесенных изменений необходимо перезапустить службу `tomcat`:

```
systemctl restart tomcat
```

5. Для проверки контроля доступа к защищенному ресурсу веб-приложения перейдите по указанному в файле конфигурации адресу (в примере `http://localhost:8080/sample/protected`). Если настройка выполнена верно, будет выведено окно авторизации, в котором необходимо указать данные (имя и пароль) пользователя, имеющего доступ к защищенному контенту.

При этом, если указать данные любого другого пользователя, не имеющего доступ к защищенному ресурсу, должна быть выведена страница с соответствующей ошибкой.

3.7.4 Механизм `UserDatabaseRealm`

Механизм `UserDatabaseRealm` используется для аутентификации с помощью встраиваемых учетных записей пользователей и ролей в памяти. Может применяться для простых приложений или для разработки и тестирования.

Во время запуска механизм `UserDatabaseRealm` загружает информацию обо всех пользователях и соответствующих им ролях из файла `tomcat-users.xml`.

Настройка работы механизма `UserDatabaseRealm` осуществляется согласно следующему алгоритму действий:

1. Определение механизма авторизации и аутентификации пользователей в файле `context.xml` и области его действия.

Укажите механизм и ресурс, который будет использоваться для хранения данных пользователей:

```
<Context>
  ...
  <Realm className="org.apache.catalina.realm.UserDatabaseRealm"
      resourceName="UserDatabase"/>
  ...
</Context>
```

2. Создание пользователей и назначение им необходимых ролей в файле `tomcat-users.xml`. Укажите учетные данные для пользователей и назначьте необходимые роли. Подробную информацию о структуре файла `tomcat-users.xml` см. в п. 3.3.3 «Файл tomcat-users.xml».

В приведенном далее примере настроены роли для пользователей `admin` и `user`:

```
<tomcat-users>
  <role rolename="manager-gui"/>
  <role rolename="admin-gui"/>

  <user username="admin" password="admin123"
        roles="manager-gui,admin-gui"/>
```

```
    <user username="user" password="user123"
          roles="user"/>
  </tomcat-users>
```

3. Настройка ограничения доступа к определенным URL-адресам веб-прило- жения в файле `$CATALINA_BASE/webapps/<название_веб-приложения>/WEB-INF/web.xml`. Подробную информацию о структуре файла `web.xml` см. в п. 3.3.2 «Файл web.xml».

В приведенном далее примере конфигурации настроено ограничение доступа к ресурсам каталога `/protected/*`. Доступ к защищенному ресурсу имеют пользователи с назначенной ролью `admin-gui`:

```
  <web-app xmlns="http://xmlns.jcp.org/xml/ns/javaee"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/javaee
    http://xmlns.jcp.org/xml/ns/javaee/web-app_3_1.xsd"
    version="3.1">

  <security-constraint>
    <web-resource-collection>
      <web-resource-name>Protected Content</web-resource-name>
      <url-pattern>/protected/*</url-pattern>
    </web-resource-collection>
    <auth-constraint>
      <role-name>admin-gui</role-name>
    </auth-constraint>
  </security-constraint>

  <login-config>
    <auth-method>BASIC</auth-method>
    <realm-name>My Application</realm-name>
  </login-config>

  <error-page>
    <error-code>403</error-code>
    <location>/error.jsp</location>
  </error-page>

  </web-app>
```

4. Для применения внесенных изменений необходимо перезапустить службу `tomcat`:

```
  systemctl restart tomcat
```

5. Для проверки контроля доступа к защищенному ресурсу веб-приложения перейдите по указанному в файле конфигурации адресу (в примере `http://localhost:8080/sample/protected`). Если настройка выполнена верно, будет выведено окно авторизации, в котором необходимо указать данные (имя и пароль) пользователя, имеющего доступ к защищенному контенту.

При этом, если указать данные любого другого пользователя, не имеющего доступ к защищенному ресурсу, должна быть выведена страница с соответствующей ошибкой.

3.7.5 Механизм JNDIRealm

Механизм JNDIRealm используется для аутентификации пользователей и управления назначенными им ролями с помощью Java Naming and Directory Interface (JNDI). Данный механизм предоставляет возможность интеграции с различными системами управления пользователями, такими как LDAP.

Настройка работы механизма JNDIRealm осуществляется согласно следующему алгоритму действий:

1. Определение механизма авторизации и аутентификации пользователей в файле server.xml и области его действия.

Укажите название механизма и определите область его действия. В приведенном далее примере область действия указанного механизма распространяется на все веб-приложения:

```
<Engine name="Catalina" defaultHost="localhost">
...
  <Realm className="org.apache.catalina.realm.JNDIRealm"
    connectionURL="ldap://localhost:389"
    connectionName="cn=admin,dc=example,dc=com"
    connectionPassword="password_LDAP"
    userBase="ou=users,dc=example,dc=com"
    userSearch="(uid=0)"
    userSubtree="true"
    roleBase="ou=groups,dc=example,dc=com"
    roleName="cn"
    roleSearch="(member=0)"
    roleSubtree="true" />
...
</Engine>
```

где значения параметров connectionName=, connectionPassword=, userBase=, roleBase= необходимо заменить на значения, фактически используемые на LDAP-сервере.

2. Настройка ограничения доступа к определенным URL-адресам веб-прило- жения в файле $CATALINA_BASE/webapps/<название_веб-приложения>/WEB-INF/web.xml. Подробную информацию о структуре файла web.xml см. в п. 3.3.2 «Файл web.xml».

В приведенном далее примере конфигурации настроено ограничение доступа к ресурсам каталога /protected/*. Доступ к защищенному ресурсу имеют пользователи с назначенной ролью appuser:

```
<web-app xmlns="http://xmlns.jcp.org/xml/ns/javaee"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/javaee
  http://xmlns.jcp.org/xml/ns/javaee/web-app_3_1.xsd"
  version="3.1">

  <security-constraint>
    <web-resource-collection>
      <web-resource-name>Protected Area</web-resource-name>
      <url-pattern>/protected/*</url-pattern>
```

```
        <http-method>GET</http-method>
        <http-method>POST</http-method>
      </web-resource-collection>
      <auth-constraint>
        <role-name>appusers</role-name>
      </auth-constraint>
    </security-constraint>

    <!-- Форма аутентификации -->
    <login-config>
      <auth-method>FORM</auth-method>
      <realm-name>LDAP Realm</realm-name>
      <form-login-config>
        <form-login-page>/jsp/login.jsp</form-login-page>
        <form-error-page>/jsp/login.jsp?error=1</form-error-page>
      </form-login-config>
    </login-config>

    <!-- Определение роли -->
    <security-role>
      <role-name>appusers</role-name>
    </security-role>

  </web-app>
```

3. Для применения внесенных изменений необходимо перезапустить службу `tomcat`:

```
systemctl restart tomcat
```

4. Для проверки контроля доступа к защищенному ресурсу веб-приложения перейдите по указанному в файле конфигурации адресу (в примере `http://localhost:8080/sample/protected`). Если настройка выполнена верно, будет выведено окно авторизации, в котором необходимо указать данные (имя и пароль) пользователя, имеющего доступ к защищенному контенту.

3.7.6 Механизм `DataSourceRealm`

Механизм `DataSourceRealm` позволяет выполнять аутентификацию и авторизацию пользователей с использованием пула соединений (`DataSource`) для подключения к базе данных.

Для применения механизма `DataSourceRealm` необходимо выполнить следующий алгоритм действий:

1. Создание базы данных. Предварительно должна быть подготовлена база данных с таблицей пользователей и ролей.

2. Определение пула соединений (`DataSource`) в файле `context.xml`. Подробную информацию о структуре файла `context.xml` см. в п. 3.3.6 «Файл context.xml».

В приведенном далее примере конфигурации описан способ подключения к базе данных:

```
<Context>
...
  <Resource name="jdbc/MyDB"
    auth="Container"
    type="javax.sql.DataSource"
    username="dbuser"
    password="dbpassword"
    driverClassName="com.mysql.cj.jdbc.Driver"
    url="jdbc:mysql://localhost:3306/mydb"/>
...
</Context>
```

Примечание. Необходимо заменить значение параметра `driverClassName= "com.mysql.cj.jdbc.Driver"` на драйвер используемой базы данных. Также рекомендуется убедиться, что значения параметров `url=`, `username=` и `password=` соответствуют используемой базе данных.

3. Настройка использования и область действия механизма `DataSourceRealm` в файле `server.xml`.

В приведенном далее примере определены основные поля, по которым должна производиться аутентификация и авторизация пользователей:

```
<Engine>
  ...
  <Realm className="org.apache.catalina.realm.JDBCRealm"
    driverName="com.mysql.cj.jdbc.Driver"
    connectionURL="jdbc:mysql://localhost:3306/mydb"
    connectionName="dbuser"
    connectionPassword="dbpassword"
    userTable="users "
    userNameCol="user_name"
    userCredCol="user_pass"
    userRoleTable="user_roles"
    roleNameCol="role_name"/>
  ...
</Engine>
```

где:

- `userTable` – имя таблицы, содержащей данные пользователей;
- `userNameCol` – имя столбца, содержащего имена пользователей;
- `userCredCol` – имя столбца, содержащего пароли пользователей;
- `userRoleTable` – имя таблицы, содержащей роли пользователей;
- `roleNameCol` – имя столбца, содержащего названия ролей.

4. Настройка ограничения доступа к определенным URL-адресам веб-приложе- ния в файле `$CATALINA_BASE/webapps/<название_веб-приложения>/WEB-INF/web.xml`. По-

дробную информацию о структуре файла `web.xml` см. в п. 3.3.2 «Файл web.xml».

В приведенном далее примере конфигурации настроено ограничение доступа к ресурсам каталога `/protected/*`. Доступ к защищенному ресурсу имеют пользователи с назначенной ролью `admin`:

```
<web-app>
...
  <security-role>
    <role-name>ADMIN</role-name>
  </security-role>

  <security-role>
    <role-name>USER</role-name>
  </security-role>

  <security-constraint>
    <display-name>Protected Area</display-name>
    <web-resource-collection>
      <web-resource-name>Protected Resources</web-resource-name>
      <url-pattern>/protected</url-pattern>
    </web-resource-collection>
    <auth-constraint>
      <role-name>ADMIN</role-name>
      <role-name>USER</role-name>
    </auth-constraint>
    </security-constraint>

  <login-config>
    <auth-method>BASIC</auth-method>
    <realm-name>MyDB Realm</realm-name>
  </login-config>

  <error-page>
    <error-code>403</error-code>
    <location>/error/error.html</location>
  </error-page>
...
</web-app>
```

4. Для применения внесенных изменений необходимо перезапустить службу `tomcat`:

```
systemctl restart tomcat
```

5. Для проверки контроля доступа к защищенному ресурсу веб-приложения перейдите по указанному в файле конфигурации адресу (в примере `http://localhost:8080/sample/protected`). Если настройка выполнена верно, будет выведено окно авторизации, в котором необходимо указать данные (имя и пароль) пользователя, имеющего доступ к защищенному контенту.

При этом, если указать данные любого другого пользователя, не имеющего доступ

к защищенному ресурсу, должна быть выведена страница с соответствующей ошибкой.

3.8 Настройка отладочных портов для веб-приложений

Для отслеживания возникающих проблем с веб-приложением (ошибки в работе, неверное отображение содержимого страниц и т.д.) в Tomcat доступна возможность просмотра кода веб-приложения и его построчная отладка.

Далее приведены необходимые действия для настройки Tomcat с целью подключения и отладки кода развернутого веб-приложения.

3.8.1 Настройка отладочных портов

Предварительно необходимо настроить соответствующие параметры конфигурации и порты, которые будут использоваться для отладки того или иного веб-приложения. В приведенном ниже примере в качестве порта для отладки будет использован порт `8000`.

В файл конфигурации `$CATALINA_BASE/conf/tomcat.conf` необходимо добавить следующую настройку:

```
  CATALINA_OPTS="-Xrunjdwp:transport=dt_socket,address=*:8000,
server=y,suspend=n"
```

где:

- `transport=dt_socket` – сокет для транспортировки;
- `address=*:8000` – порт для соединения;
- `server=y` – указывает, что JVM будет работать как сервер для отладчика;
- `suspend=n` – указывает, что JVM не будет ожидать подключения от отладчика перед запуском (для включения ожидания необходимо изменить на `suspend=y`).

3.8.2 Настройка удаленных подключений

Для обеспечения возможности удаленных подключений к серверу Tomcat необходимо в файл конфигурации `$CATALINA_BASE/conf/server.xml` в секцию `<Host>` добавить следующую настройку:

```
  <Valve className="org.apache.catalina.valves.RemoteAddrValve"
    allow="127\.\d+\.\d+\.\d+|::1|0:0:0:0:0:0:0:1|192\.168\.1\.\d+"
  />
```

где:

- `allow="127\.\d+\.\d+\.\d+|::1|0:0:0:0:0:0:0:1|192\.168\.1\.\d+"` – разрешает подключение с любого IP-адреса в диапазоне `192.168.1.XXX`.

3.8.3 Открытие портов

Также необходимо открыть указанный порт для соединений и перезапустить конфигурацию `firewalld`:

```
  firewall-cmd --permanent --add-port=8000/tcp
  firewall-cmd --reload
```

Подробную информацию о настройке `firewalld` см. в п. «Брандмауэр Firewall» Руководства администратора. Часть 1.

Для применения внесенных изменений требуется перезапустить службу `tomcat`:

```
systemctl restart tomcat
```

Сервер Tomcat готов к запуску в режиме отладки.

3.8.4 Подключение отладчика

После выполненной настройки можно подключиться к веб-приложению с помощью отладчика в используемой IDE.

Для этого необходимо выполнить следующий алгоритм действий:

1. Создать новую конфигурацию удаленной отладки.
2. Указать хост (для локальной отладки – `localhost`, для удаленной отладки – `IP`-адрес сервера Tomcat) и порт (в приведенном примере – `8000`).
3. Запустить конфигурацию отладки.

При успешном подключении в сообщениях отладчика должно отображаться примерно следующее сообщение:

```
Connection to server
```

3.9 Загрузка статичных страниц

Tomcat можно настроить на чтение файлов из любого каталога и их обслуживание по определенному `URL`. Такая конфигурация полностью отделена от конфигурации веб-приложения, что позволяет запускать Tomcat и обслуживать статические файлы без запуска веб-приложений.

Также загрузка статичных страниц может быть полезна в тех случаях, когда веб-приложение по каким-либо причинам недоступно для запуска, для информирования пользователей о проблеме.

Загрузку статичных страниц можно настроить в основном файле конфигурации Tomcat – `server.xml`.

3.9.1 Настройка статичной страницы

Предварительно необходимо создать каталог для хранения статичных `HTML`-страниц и назначить ему соответствующие права, например:

```
mkdir /usr/share/tomcat/webapps/ROOT/error
chown -R tomcat:tomcat /usr/share/tomcat/webapps/ROOT/
```

Затем необходимо создать непосредственно `HTML`-страницу:

```
nano /usr/share/tomcat/webapps/ROOT/error/error.html
```

со следующим содержимым:

```
<html>
  <body>
```

```
      <h1>
        Sorry, application is not available!
      </h1>
    </body>
  </html>
```

В данном примере в случае ошибки подключения к веб-приложению будет загружаться статичная HTML-страница (`error.html`) с соответствующим уведомлением.

3.9.2 Настройка файла конфигурации

После создания статичной HTML-страницы необходимо внести соответствующие изменения в файле конфигурации обработки веб-контента `$CATALINA_BASE/conf/web.xml`.

Для этого необходимо открыть файл на редактирование:

```
nano $CATALINA_BASE/conf/web.xml
```

и в конец файла добавить следующее содержимое:

```
<error-page>
   <error-code>404</error-code>
   <location>/error/error.html</location>
</error-page>
<error-page>
   <error-code>500</error-code>
   <location>/error/error.html</location>
</error-page>
```

Приведенная настройка означает, что при возникновении ошибок обращения к приложению с кодами 404 или 500 будет выведена созданная статичная HTML-страница с уведомлением.

Для применения внесенных изменений необходимо перезапустить службу `tomcat`:

```
systemctl restart tomcat
```

Для проверки корректности настройки следует открыть браузер и перейти по адресу, который вызывает ошибку 404 или 500 (например, `http://localhost:8080/bad-request`), или вызвать ошибку в веб-приложении.

3.10 Журналирование событий в Tomcat

Tomcat использует систему журналирования, основанную на Apache Commons Logging, и по умолчанию интегрируется с библиотекой `java.util.logging` (JUL). Таким образом, внутреннее ведение журнала Tomcat и ведение журнала любым веб-приложением останутся независимыми, даже если веб-приложение использует Apache Commons Logging.

Журналы Apache Tomcat необходимы для мониторинга, отладки и обслуживания приложений Java, работающих на Tomcat. Журналы фиксируют важную информацию, такую как сведения о запуске сервера, обработке запросов и ошибках приложений.

Журналы помогают разработчикам и системным администраторам устранять неполадки, анализировать трафик и обеспечивать стабильность работы приложений.

Tomcat генерирует несколько журналов, каждый из которых служит определенной цели:

- отслеживание активности приложений;
- мониторинг HTTP-запросов;
- отладка ошибок сервера;
- регистрация событий безопасности.

Файлы журналов расположены в каталоге `/var/log/tomcat/` и доступны для анализа только суперпользователю `root`.

3.10.1 Типы файлов журналов

Apache Tomcat поддерживает несколько типов файлов журналов для отслеживания различных аспектов работы сервера. Каждый файл журнала необходим для фиксации определенных событий, помогая разработчикам и администраторам контролировать, отлаживать и оптимизировать приложения.

Ключевыми журналами в Tomcat являются `catalina.log` и `localhost_access_log.*.txt`.

Журнал `catalina.log`

Журнал `catalina.log` регистрирует события, связанные с последовательностями инициализации и завершения работы Tomcat. Содержимое журнала включает в себя следующую информацию:

- процессы запуска сервера, включая привязку портов и загрузку файлов конфигурации;
- журналы плавного и принудительного завершения работы;
- ошибки конфигурации или проблемы, препятствующие запуску Tomcat.

Журнал `catalina.log` помогает диагностировать сбои при запуске, такие как конфликты портов или отсутствующие библиотеки. Может быть полезен при анализе причин остановки или перезапуска сервера.

Журнал `localhost_access_log.*.txt`

Журнал `localhost_access_log.*.txt` отслеживает все входящие HTTP-запросы, отправленные на сервер. Содержимое журнала включает в себя следующую информацию:

- IP-адрес клиента, метод HTTP-запроса (`GET`, `POST` и т. д.), запрошенный URL-адрес и код состояния ответа;
- пользовательские агенты и время отклика;
- подробное протоколирование запросов и ответов API.

Администраторы анализируют журнал `localhost_access_log.*.txt` для обнаружения подозрительных действий, для мониторинга шаблонов запросов API и оптимизации производительности. Также журнал помогает отслеживать источники трафика и выявлять медленно работающие конечные точки.

Пример записи в журнале:

```
  192.168.101.42 - - [24/Feb/2025:17:25:10 +0300] "GET /sample/
HTTP/1.1" 200 636
```

где:

- `192.168.101.42` – IP-адрес, с которого пришел запрос;
- `[24/Feb/2025:17:25:10 +0300]` – дата и время запроса;
- `"GET /sample/ HTTP/1.1"` – запрос доступа к приложению методом `GET`;
- `200` – статус ответа;
- `636` – размер ответа (в байтах).

3.10.2 Настройка формата журналов

Формат ведения журналов настраивается в файле конфигурации `$CATALINA_BASE/conf/server.xml` в секции `<Valve>`:

```
  <Valve className="org.apache.catalina.valves.AccessLogValve"
         directory="logs"
         prefix="localhost_access_log"
         suffix=".txt"
         pattern="%h %l %u %t \"%r\" %s %b" />
```

где:

- `directory="logs"` – каталог хранения файлов журналов;
- `prefix="localhost_access_log"` – префикс для имен файлов журнала;
- `suffix=".txt"` – расширение файла журнала;
- `pattern="%h %l %u %t \"%r\" %s %b"` – формат записи журнала.

Приведенная конфигурация определяет, что журналы доступа будут сохраняться в виде обычного текста в указанном каталоге `/logs` в структурированном формате.

3.10.3 Настройка ведения журнала

Уровень журналирования

Для настройки параметров ведения журнала используется файл `$CATALINA_BASE/conf/logging.properties`. Администратор может настроить уровни серьезности (критичности) регистрируемых событий, а также формат сообщений и назначение журналов.

Уровни критичности событий могут принимать следующие значения:

- `SEVERE` – ошибки, которые могут привести к сбою приложения;
- `WARNING` – предупреждения о потенциальных проблемах;
- `INFO` – общая информация о работе сервера;
- `CONFIG` – информация о конфигурации;
- `FINE`, `FINER`, `FINEST` – подробные сообщения для отладки.

Синтаксис имеет следующий вид:

```
  # Уровень журналирования для сервера
  org.apache.catalina.level = INFO
```

```
   org.apache.catalina.handlers = java.util.logging.ConsoleHandler

   # Уровень журналирования для приложений
   org.apache.catalina.startup.level = FINE
```

Формат сообщений

Администратор может настроить формат сообщений, используя обработчики (`handlers`) событий. Пример использования файлового обработчика:

```
   # Определение файлового обработчика
   1catalina.org.apache.juli.FileHandler.level = INFO
   1catalina.org.apache.juli.FileHandler.directory = ${catalina.
base} /logs
   1catalina.org.apache.juli.FileHandler.prefix = catalina.
   1catalina.org.apache.juli.FileHandler.formatter = java.util.
logging.SimpleFormatter
```

Обработчики определяют, в какой каталог будут записываться журналы. В файле `logging.properties` можно настроить несколько обработчиков, например:

```
   # Консольный обработчик
   java.util.logging.ConsoleHandler.level = INFO
   java.util.logging.ConsoleHandler.formatter = java.util.logging.
SimpleFormatter

   # Файловый обработчик
   1catalina.org.apache.juli.FileHandler.level = INFO
   1catalina.org.apache.juli.FileHandler.directory = ${catalina.
base}/logs
   1catalina.org.apache.juli.FileHandler.prefix = catalina.
   1catalina.org.apache.juli.FileHandler.formatter = java.util.
logging.SimpleFormatter
```

3.10.4 Ротация журналов

Ротация файлов журналов в Tomcat по умолчанию осуществляется ежедневно с использованием файловых обработчиков. Журналы записываются в заданный в файловом обработчике каталог (`$catalina.base/logs`) с указанием даты создания файла:

```
   ll -h /var/log/tomcat/
```

```
   -rw-r--r--. 1 tomcat tomcat  23K апр 29 16:47 catalina.2025-04-
29.log
   -rw-r--r--. 1 tomcat tomcat    0 апр 30 10:10 catalina.2025-04-
30.log
```

Однако в целях безопасности бывает необходима настройка ротации файлов журналов при достижении файлами определенного размера. Для настройки автоматического

управления ротацией создайте файл конфигурации `logrotate` для журналов Tomcat:

```
nano /etc/logrotate.d/tomcat
```

и укажите необходимые настройки, например:

```
/var/log/tomcat/*log {
missingok
maxsize 25M
rotate 5
compress
copytruncate
create 0640 tomcat tomcat
}
```

где:

- `/var/log/tomcat/*log` – настройка будет применена для всех файлов журналов Tomcat с расширением `.log` (настройку можно указать для каждого отдельного файла журнала);
- `missingok` – не выдавать ошибку, если файл журнала отсутствует;
- `maxsize 25M` – максимальный размер файла журнала, в данном примере `25 МБ`;
- `rotate 5` – количество хранимых ротаций, в данном примере `5`;
- `compress` – сжимать старые файлы;
- `copytruncate` – создать копию исходного файла журнала, а затем удалить его содержимое, чтобы уменьшить размер файла;
- `create 0640 tomcat tomcat` – создавать новые файлы с указанными правами и владельцем.

Таким образом, при достижении файлом журнала максимально установленного размера в `25 МБ` будет выполнено его архивирование. Копия журнала будет сохранена в каталоге `/var/log/tomcat` с указанием даты ее создания:

Пример файла:

```
ll -h /var/log/tomcat/
```

```
-rw-r--r--. 1 tomcat tomcat    0 мая 5 11:16 catalina.rotate.
2025-05-05.log
-rw-r--r--. 1 tomcat tomcat 2,8K мая 5 10:10 catalina.rotate.
2025-05-05.log-20250505
```