1. Протокол HTTP, основные свойства HTTP, структура запроса и ответа. Протокол HTTPS. Понятие web-приложения, структура и принципы работы web-приложения. Понятие асинхронности.

***HTTP*** (HyperText Transfer Protocol)— протокол передачи данных для передачи гипертекстовых документов.

Является протоколом прикладного (верхнего, 7-го) уровня.

Протокол HTTP предполагает использование клиент-серверной структуры передачи данных. Клиентское приложение формирует запрос и отправляет его на сервер, после чего сервер обрабатывает данный запрос, формирует ответ и передаёт его обратно клиенту.

***HTTP: основные свойства***

- версии HTTP/1.1 – действующий (текстовый), HTTP/2 – черновой (не распространен, бинарный); HTTP / 3 (HTTP/3 — новый синтаксис HTTP на основе UDP.)

- два типа абонентов: клиент и сервер;

- два типа сообщений: request и response;

- от клиента к серверу – request;

- от сервера к клиенту – response;

- на один request всегда один response, иначе ошибка;

- одному response всегда один request, иначе ошибка;

- TCP-порты: 80, 443(HTTPS);

- для адресации используется URI или URN;

- поддерживается W3C, описан в нескольких RFC.(RFC2616)

- http не имеет состояния, но имеет сессию(не существует связи между 2мя запросами)

***Request***:

- метод;

- URI;

- версия протокола (HTTP/1.1);

- заголовки (пары: имя/заголовок);

- параметры (пары: имя/заголовок);

- расширение.

***Response:***

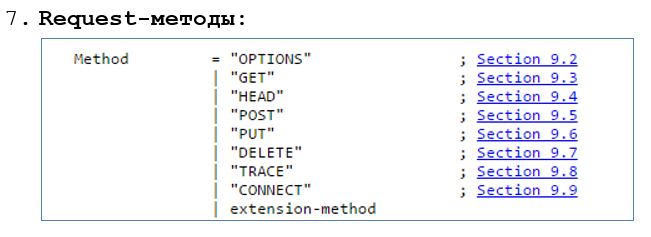
- версия протокола (HTTP/1.1);

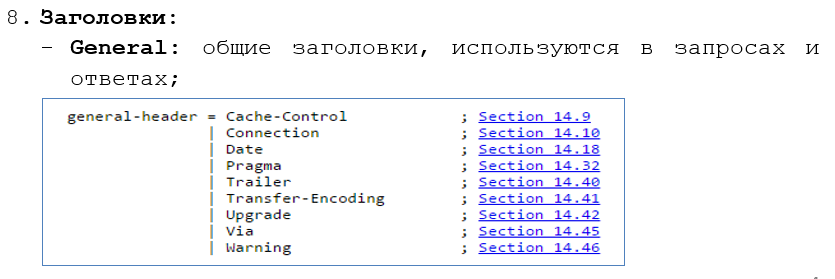
- код состояния (1xx, 2xx, 3xx, 4xx, 5xx);

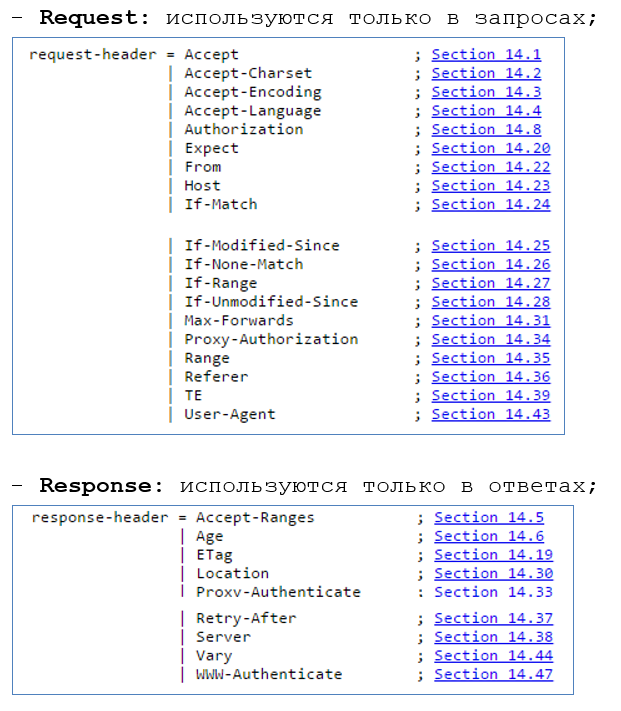
- пояснение к коду состояния;

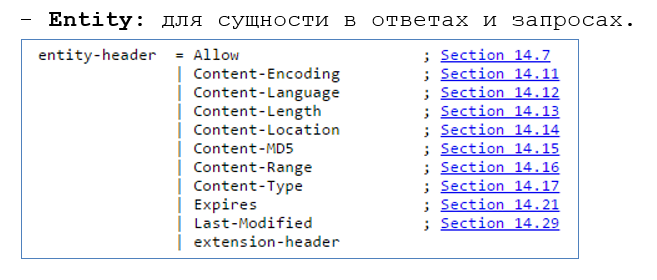
- заголовки (пары: имя/заголовок);

- расширение.









***Response: Код состояния:***

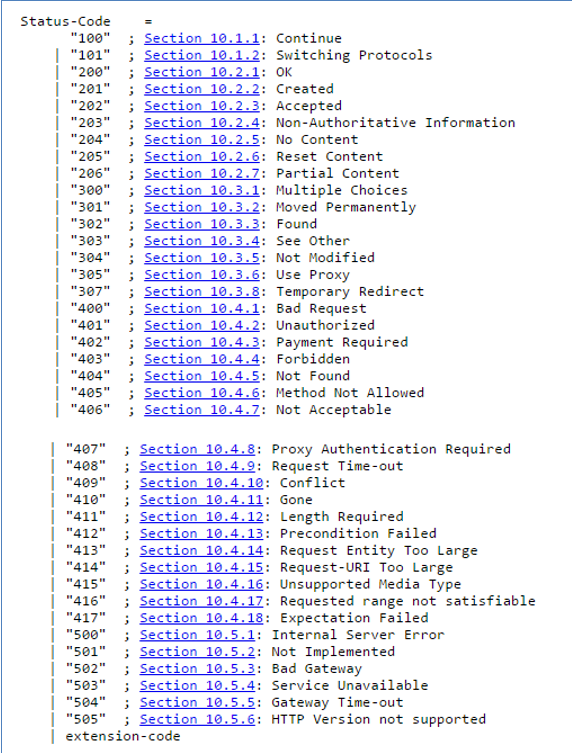
- 1xx: информационные сообщения;

- 2xx: успешный ответ;

- 3xx: переадресация;

- 4xx: ошибка клиента;

- 5xx: ошибка сервера.



***HTTPS*** (HyperText Transfer Protocol Secure) — расширение протокола HTTP для поддержки шифрования в целях повышения безопасности. Данные в протоколе HTTPS передаются поверх криптографических протоколов TLS или SSL. Для HTTPS по умолчанию используется TCP-порт 443.

**Клиент-серверное** приложение, у которого клиент и сервер взаимодействуют по протоколу HTTP называется ***web-приложением***. Когда говорят о разработке web-приложения, говорят о разработке frontend (клиента) и backend (сервера). Клиент является инициатором соединения.

**АРХИТЕКТУРА ПРИЛОЖЕНИЯ**

Есть клиент и есть сервер – две части одной и той же программы, которые взаимодействуют между собой по протоколу HTTP.

Между клиентом и сервером ходит 2 типа сообщений:

· от клиента к серверу – **request**

· от сервера к клиенту – **response**

**Обычный запрос проходит несколько стадий**:

1. Открытие соединения.

2. Отправка запроса.

3. Получение ответа.

4. Закрытие соединения.

**URI (uniform resource identifier)** – унифицированный идентификатор ресурса (документ, изображение, файл, служба, электронная почта).

**URL** – унифицированный локатор ресурса - URI, содержащий местонахождение ресурса и способ обращения к ресурсу.

**URN** – унифицированное имя ресурса – URI, не содержащее в себе месторасположение и метод доступа к ресурсу.

**PURPL** (persistent uniform resource locator) – постоянный унифицированный локатор ресурса. Доступ к конечному ресурсу через redirect. Это некоторая база данных, содержащая месторасположение и способ доступа к ресурсу.

**Клиент: веб-браузер**

Умеет интерпретировать программу, которую ему присылает сервер.

Веб-браузер умеет генерировать HTTP-запросы:

– если есть адресная строка, и он делает запрос

– есть ряд HTML-тегов, при интерпретации которых браузер делает запрос (form, a, img, script, link, audio, video)

– объект веб-браузера: XMLHTTPRequest

– JS API

**Общие принципы построения веб-приложений**

– веб-ресурсы приложения

– запросы и ответы

– фильтры

– кэш (данных и вывода)

– слушатели событий

– принципы безопасности

**HTTP–сервер** – программа, которая слушает некоторый порт на своем компьютере.

**Порт** – номер приложения, которому адресован этот запрос. Понятие порт существует на уровне tsp-протокола. Порт слушает входящие сообщения.

**Сессия** – серверный объект, хранящий информацию о соединении с клиентом, создается при первом обращении.

Если timeout превышен, то session разрушается и при последующем запросе создается новый экземпляр. Каждая сессия имеет собственный идентификатор (Session ID, 128 или больше бит (16 байт)). Каждый request принадлежит какой-то сессии. Сессия характеризуется двумя параметрами: timeout и session ID.

Обычно в request сервер записывает либо идентификатор этой сессии, либо просто программную ссылку на этот объект сессии.

*Если с одним и тем же сервером работает много клиентов, у каждого из них своя сессия.*

**Куки** – это порция информации, которая может быть сохранена на стороне клиента по инициативе сервера. Когда клиент делает первый запрос, сервер проверяет, есть ли у него заголовок с именем куки. Если этого заголовка нет, то он считает, что это первый запрос и для него создается сессия.

Клиент должен хранить информацию о том, в рамках какой сессии мы отправляем эти запросы. Браузер в себе сохраняет либо файл куки, но чаще сохраняет **localstorage** (поддерживается на стороне браузера).

**Конфигурационный файл** (обычно xml) содержит в себе некоторые статические характеристики приложения. Он служит для создания контекста веб-приложения.

**Контекст веб-приложения** – системный объект общий для всех сессий. Предназначен для хранений информации об одном веб-приложений, общий для всех сессий. Как правило, формируется сразу при загрузке веб-сервера.

**Фильтр** – серверный объект, препроцессор запроса, предназначенный для предварительно обработки объекта request. К одному ресурсу может быть построена цепочка фильтров, последний в цепочке – ресурс. Фильтр может прервать цепочку и сам сформировать ответ клиенту.

**Фильтры** загружены всегда, вместе с контекстом. Фильтры привязываются к URI.

Информация о цепочке фильтров и привязке этих фильтров к ресурсу хранится в контексте.

Фильтр может не пустить запрос к ресурсу. Он может сам ответить, не пуская его дальше: к следующему фильтру или ресурсу.

**Слушатели событий (Listener)** – серверные объекты для обработки событий жизненного цикла веб-приложения. С помощью Listener можно врезаться в ход выполнения приложения и что-то там изменит.

**Кэш** – это системный объект, предназначенный для хранения данных в оперативной памяти с целью ускорения работы веб-приложения. Часто в кэше запоминаются response.

Для каждого запроса создается новый экземпляр приложения. Общим остается контекст. В рамках серии запросов остается сессия. На каждый новый запрос создается новый экземпляр приложения.

**Постоянное соединение**. Использование одного TCP-соединения для многократных пар запрос-сервер вместо последовательного открытия новых соединений для каждой пары запрос-ответ. Клиент может запросить постоянное соединение с помощью заголовка *Connection: Keep-Alive*, сервер подтверждает заголовком Connection: Keep-Alive.

**Пул соединение с базой данных**. Несколько предварительно и постоянно открытых соединений с сервером СУБД, которые используют приложения. Выбор подключения из пула по open, возврат в пул close. Если все подключения пула заняты, запрос на соединение ставиться в очередь.

**Пул соединение** – два постоянно открытых коннекта в базе данных. К этому пулу устанавливается очередь запросов. Можно регулировать количество этих соединений в зависимости от нагрузки.

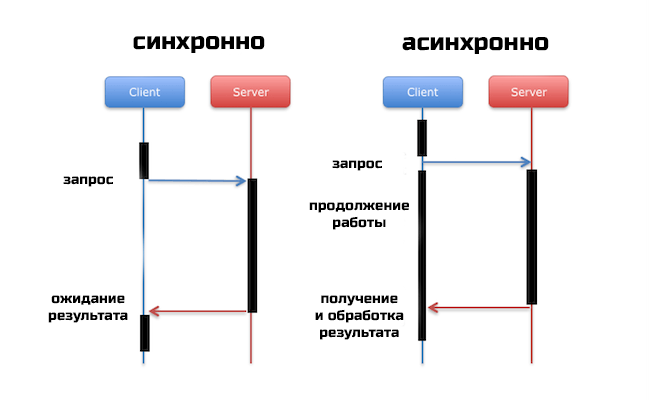
Создается несколько соединений, к пулу отправляется sql-запрос, который пул перенаправляет к БД.

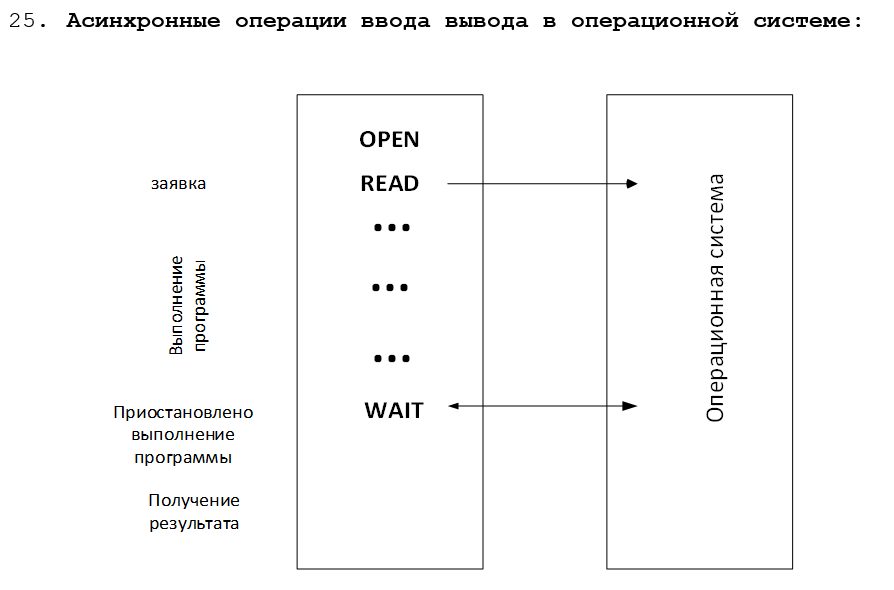
**Кэширование на стороне браузера**. Если запрос обыкновенный, без кэширования, то при каждом новом запросе эта картинка будет скачиваться. Если картинка кэшируется, то все сохраняется на стороне браузера и запрос получается легковесным.

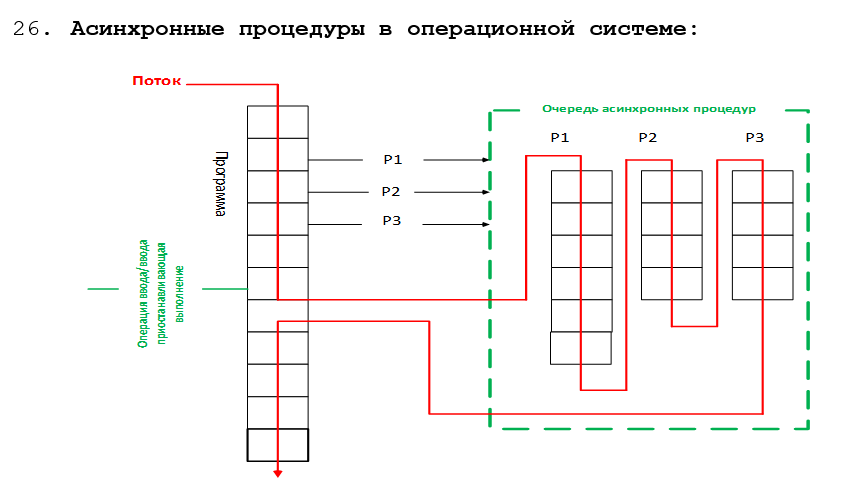
**Синхронный** - блокирует поток до получения сообщения, асинхронный – нет.

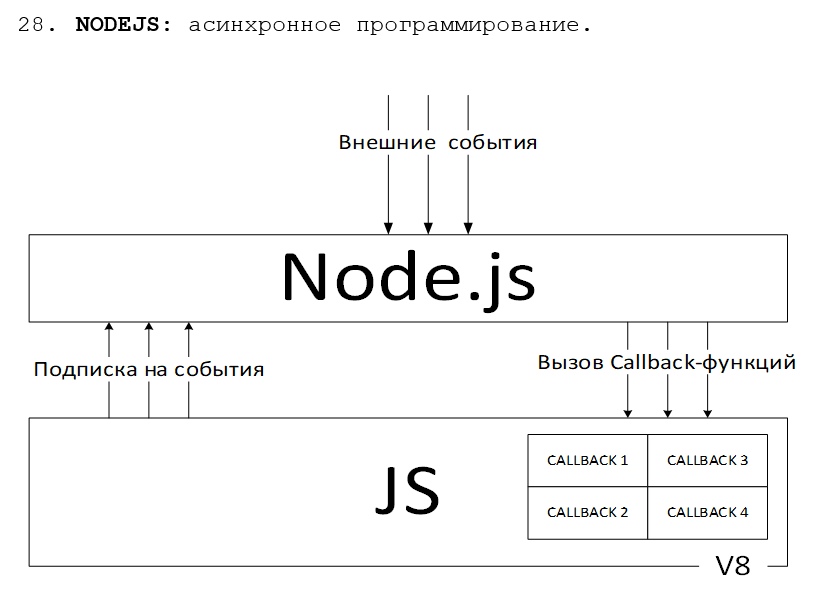
***Понятие асинхронности***: операция называется асинхронной, если ее выполнение осуществляется в 2 фазы: 1) заявка на исполнение; 2) получение результата; при этом участвуют два механизма: A-механизм, формирующий заявку и потом получающий результат; B-механизм, получающий заявку от A, исполняющий операцию и отправляющий результат A; продолжительность исполнения операции B-механизмом, как правило, непредсказуемо; в то время пока B-механизм исполняет операцию, А-механизм выполняет собственную работу. Применение асинхронности не противоречит применению многопоточности.











Взаимодействие с NODEJS осуществляется при помощи подхода издатель-подписчик. Просим NODEJS чтобы он нам сообщил о появлении некоторого события. on - подписка на событие Чтобы поставить в очередь колбэки существует 2 способа: (для отправки ф-ций в очередь) Process.nextTick - откладывает выполнение ровно на 1 цикл. Ставим выполнение в начало очереди колбэков. setImmediate - ставит в конец очереди колбэков.