ПСКП

ВВЕДЕНИЕ

Клиент-серверное приложение - приложение (программа) с клиент-серверной архитектурой: приложение, состоящее из двух компонент - клиента и сервера; клиент и сервер взаимодействуют между собой в соответствии с заданными правилами (спецификациями, протоколами); для взаимодействия между клиентом и сервером в соответствии с правилами должно быть установлено соответствие.

Инициатором соединения всегда является клиент.

Протокол - правила взаимодействия.

Программирование серверного приложения - разработка серверной компоненты приложения клиент-серверной архитектуры.

В этом курсе будут рассматриваться клиент-серверные приложения, клиент и сервер которых расположены на разных узлах сети Интернет (устройствах, подключенных к сети) и взаимодействуют по протоколу HTTP.

Веб приложения - приложения, имеющие архитектуру клиент-сервер, которые взаимодействуют по протоколу HTTP.

Сеть Интернет - это:

1. Сеть на основе стека протоколов TCP/IP
2. Стандарты Internet (RFC, STD)
3. ***Службы Интернет*** (DNS, SMTP/POP3/IMAP, WWW, FTP, Telnet, SSH,...) - это протокол + сервер, который может работать по этому протоколу.
4. Организации, управляющие сетью Интернет (ISOC, IETF, ICANN, IANA, IAB,..).

Как отличить службу от других приложений сети? Все порты, которые используют службы, их номера находятся в пределах 1024. С номерами портов до 1024 - это службы.

HTTP - протокол прикладного уровня. Ниже находится стек протоколов TCP/IP.

Когда говорят о разработке web-приложения, говорят о разработке ***frontend*** (клиента) и ***backend*** (сервера).

Курс посвящен разработке серверной части web-приложения или иначе разработке web-сервера (backend).

Узел Интернет (хост) - это устройство, имеющее IP-адрес и

подключенное к сети Интернет (обычно к сети Интернет-провайдера). Каждый узел характеризуется своей программно-аппаратной платформой - аппаратной и операционной системой.

Кроссплатформенное приложение - приложение, способное работать на более чем одной программно-аппаратной (аппаратура + операционная система) платформе. Кроссплатформенность может быть достигнута различными способами:

1. На уровне компилятора (C, C++);
2. На уровне среды (фреймворка) исполнения (Java/JVM, C#/.NET CORE, JS/Node.js/V8).

В этом курсе будут рассматриваться приложения, кроссплатформенность которых обеспечивается средой исполнения.

Курс посвящен разработке web-серверов, которые могут работать на более чем одной программно-аппаратной платформе, или иначе, разработке кроссплатформенных web-серверов.

Технологии для разработки кроссплатформенных web-серверов:

- PHP/Apache, LAMP;
- Java/JVM/Application Server;
- C#/ASP.NET CORE;
- Python/Django;
- Ruby on Rails;
- JS/Node.js, ….

В этом курсе будет рассматриваться разработка web-серверов с помощью технологии JS/Node.js.

Web-сервер: ресурсы потребляемые web-сервером, блокирующий ввод/вывод. Его основная задача принимать запросы, обрабатывать из и отправлять ответы.

I/o - ввод/вывод. Оставляет заявку ОС и ждем, пока она сообщит нам, что операция окончена. Наша задача задействовать время ожидания. (Проблема блокирующего ввода/вывода)

CPU - процессор

Синхронный запрос - отправили запрос и ждем ответ, ничего не делаем, ждем окончания операции.

2 подхода для решения проблем блокирующего ввода/вывода:

1. применение многопоточности (ограничение по количеству потоков, каждый поток требует дополнительной памяти);
2. применение паттерна Reactor. Apache – многопоточность, Nginx – Reactor.

NODEJS: закон Адмала, ограниченность возможностей, *speedup* – кратность прироста скорости вычисления, parallel portion – степень

распараллеливания алгоритма (не все можно распараллелить), *number of processors* – количество процессоров.

Показывает как растет производительность в зависимости от количества процессоров. Полоски процент распараллеливания задачи. Все равно приходим в насыщение.

HTTP. Основные свойства. Структура сервера. 21

Web-программирование - это разработка клиент-серверных приложений, компоненты которого взаимодействуют по протоколу HTTP, частный случай программирования в Интернет.

Архитектура веб-приложения

Loading...

Loading...

Loading...

Loading...

Loading...

HTTP основные свойства:

- Версии HTTP/1.1 -действующий (текстовый), HTTP/2 - черновой (не распространен, бинарный)
- 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 (для тех, что не поддерживают шифрование), 443 (для тех, что поддерживают шифрование)
- Для адресации используется URI или URN
- Поддерживается W3C, описан в нескольких RFC

***Структура Request (какую информацию мы может передать серверу в запросе)**:*

- метод;
- URI (часть без префикса http начин с EGH на скрине..);
- версия протокола (HTTP/1.1);
- заголовки (пары: имя/заголовок);
- параметры (пары: имя/заголовок);
- расширение.

В POST параметры передаются в теле запроса, GET - по URI.

Структура Response:

- версия протокола (HTTP/1.1);
- код состояния (1xx, 2xx, 3xx, 4xx, 5xx);
- пояснение к коду состояния;
- заголовки (пары: имя/заголовок);
- расширение.

html - это ответ сервера (на скрине). Рекомендуется имя заголовка начинать с X.

Методы запроса Request:

Заголовки:

- *General:* общие заголовки, используются в запросах и ответах;
- *Request:* используются только в запросах;
- *Response:* используются только в ответах;
- *Entity:* для сущности в ответах и запросах.

Response: Код состояния:

- *1xx:* информационные сообщения;
- *2xx:* успешный ответ;

- *3xx:* переадресация;
- *4xx:* ошибка клиента;
- *5xx:* ошибка сервера.

Uniform Resource Identifier (URI) – унифицированный идентификатор ресурса (документ, изображение, файл, служба, электронная почта,...).

Uniform Resource Location (URL) - унифицированный локатор ресурса , содержащий местонахождение ресурса и способ обращения (протокол) к ресурса, описывает множество URI.

URL и URI одно и тоже в данном курсе. URI, URL, URN – рекомендуется использовать термин URI

Uniform Resource Name (URN) - унифицированное имя ресурса – URI, имя ресурса, не содержащее месторасположение и способ доступа к ресурсу. В будущем URN должен заменить URL (для решения проблем с перемещением ресурсов в Internet).

Браузер - готовый http-клиент. Характеризуется поддержкой JS, CSS, HTML, XML..

Web-браузер: генерация HTTP-запросов.

- адресная URI-строка (GET);
- HTML-тег: **<form>**;
- HTML-тег: **<a>** (GET);
- HTML-тег: **<img>** (GET);
- HTML-тег: **<script>** (GET);
- HTML-тег: **<link>** (GET);
- HTML-тег: **<audio>** (GET);
- HTML-тег: **<video>** (GET);
- HTML-тег: **<frame>** GET), не поддерживается в HTML5;
- Объект web-браузера: **XMLHTTPRequest** - специальный объект, встроенный в браузер, к которому есть JS интерфейс;
- JavaScript API: **web-сокеты** - протокол, который использует протокол HTTP для установки соединения.

Document Object Model (DOM) - это то, каким образом HTML-документ представляется внутри браузера. Это собственность браузера. Интерфейс JavaScript для доступа к содержимому HTML-документа. Стандарт в W3C.

CSS(Cascading Style Sheets) – каскадные таблицы стилей, CSS1, CSS2, CSS3 (текущий уровень), CSS4(в разработке с 2011).

JavaScript Engine - виртуальная машина, которая встроена в браузер

и умеет интерпретировать JS.

Общий подход к разработке веб-приложений

HTTP-запрос - последовательность бит. Сервер преобразует это в объект Request. Надо обработать Request, взять инфу, заполнить ответ и затем отправить его клиенту.

ВВЕДЕНИЕ В NODE.JS

NODEJS **-** программная платформа для разработки серверных web-приложений на языке JS/V8. ***V8***- JS Engine, виртуальная машина, которая встроена в браузер и интерпретирует JS код.

Свойства NODEJS

- основан на *Chrome V8*;
- *среда (контейнер) исполнения* приложений на JavaScript;
- поддерживает механизм *асинхронности*;
- ориентирован на *события*;
- *однопоточный* (код приложения исполняется только в одном потоке, один стек вызовов); обычно в серверах для каждого соединения создается свой поток, в Node.js все соединения обрабатываются в одном JS-потоке;
- *не блокирует* выполнение кода при вводе/выводе (в файловой системе до 4х одновременно);
- в состав Node.js входят инструменты: *npm* – пакетный менеджер***; gyp*** - Python-генератор проектов; *gtest* – Google фреймворк для тестирования C++ приложений;
- использует библиотеки: *V8* – библиотека V8 Engine, *libuv* – библиотека для абстрагирования неблокирующих операций ввода/вывода; *http-parser* – легковесный парсер http-сообщений (написан на C и не выполняет никаких системных вызовов); *c-ares* - библиотека для работы с DNS; *OpenSSL* – библиотека для криптографии; *zlib* – сжатие и распаковка.

Core JS API - входит стандартный набор модулей JS
Binding - связывающий софт
LIBUV - для связи с ОС

Архитектура и принципы работы NODEJS

NODEJS -внешняя платформа. Внутри V8. Он взаимодействует с внешней платформой по принципу подписчик - издатель. Внешняя платформа имеет ряд событий, которые он может генерировать. V8 подписывается на эти события и на них реагирует. Мы разрабатываем

приложение, которое интерпретируется JS Engine. Пишем код, который является обработчиком внешних событий. Стек вызовов (для хранения контекста, помещаются сюда функции) обрабатывает один поток. Heap - каждому приложению выделяется область памяти, из которой мы можем получать и возвращать. Callback queue - для того, чтобы сохранить контекст отложенной процедуры. После того, как кол стек стал пустым, выполняются функции из колбэк очереди.

response.end - поместить в тело ответа. Можно делать отправку.

request.on ('data',..) - событие data, пришли данные. Ловим событие, указываем какую функцию нужно вызвать при появлении события. Но функция не будет выполняться сразу, поступает в колбе очередь. Последняя функция - end.

readFileSync - синхронный вариант функции readFile. Пока не просчитается файл и данные не переместятся в html, сервер не будет делать никаких действий. Если бы была асинхронная, то должна быть функция обратного вызова, колбэк функция. Такой себе подход.

Любая асинхронная операция имеет 2 фазы: оставить заявку и обработка ответа.

end - поместить в тело.

ГЛОБАЛЬНЫЕ ОБЪЕКТЫ

global, process, buffer

Отличие global от window

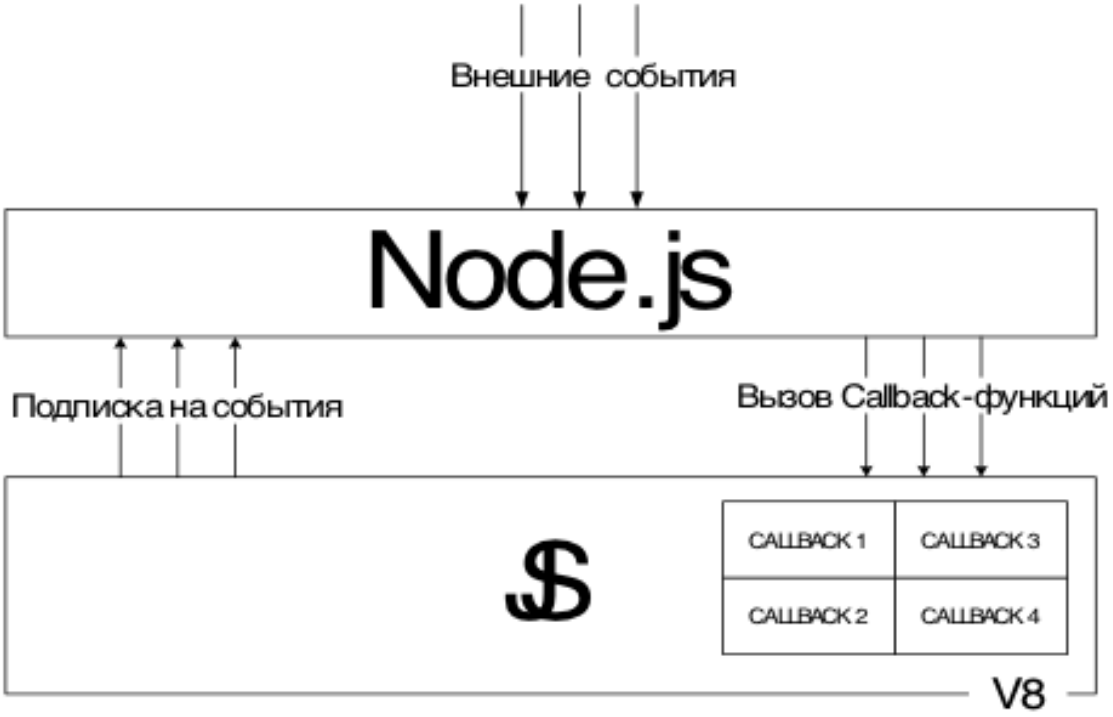
x - в модуле не перекрывается, модуль сохраняет свое состояние, переменная имеет свою область видимости.

Глобал, как и виндоу хранит var данные на уровне модуля. Глобал - общий объект для модуля.

Process - хранит информацию о среде выполнения.

Модули можно разделить на 3 уровня: в core, дополнительные, созданные разработчиками.

6. **Принцип издатель подписчик**



Взаимодействие с NODEJS осуществляется при помощи подхода издатель-подписчик. Просим NODEJS чтобы он нам сообщил о появлении некоторого события.

on - подписка на событие

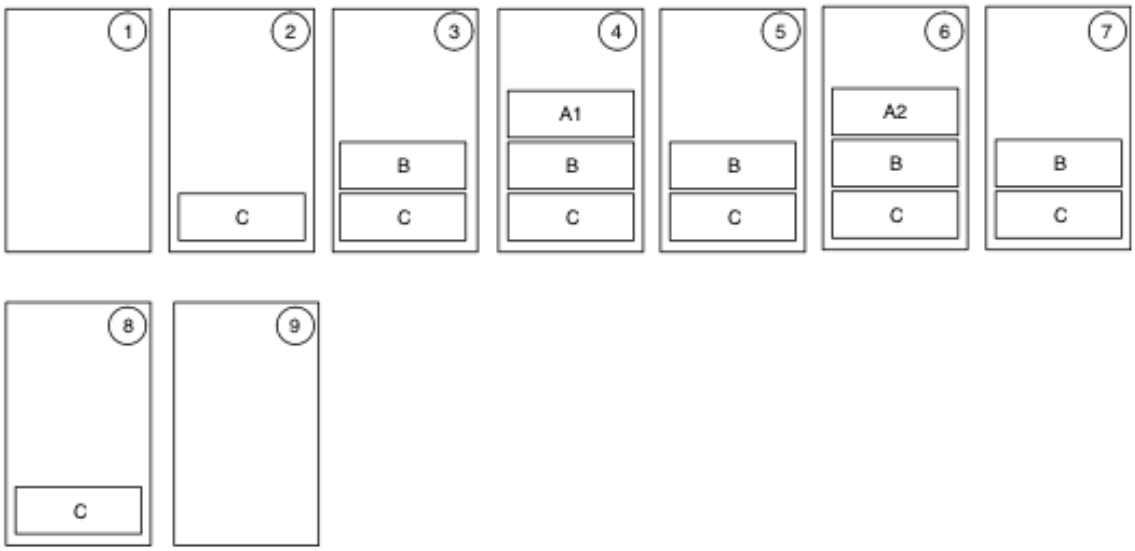
Чтобы поставить в очередь колбэки существует 2 способа: (для отправки ф-ций в очередь)

- Process.nextTick - откладывает выполнение ровно на 1 цикл. Ставим выполнение в начало очереди колбэков.
- setImmediate - ставит в конец очереди колбэков.

класс Buffer – предназначен для работы с двоичными данными: набором октетов.

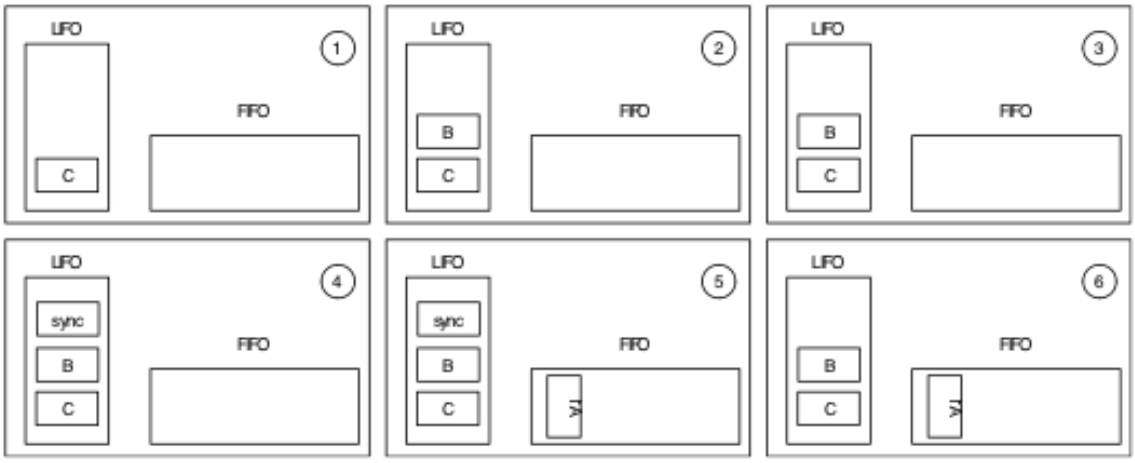
Принцип функционирования стека вызовов

```
function A1(){ /* выполнение A1 */}
function A2(){ /* выполнение A2 */}
function B() { A1(); A2(); /* выполнение B
function C() { B();        /* выполнение C

C();
```

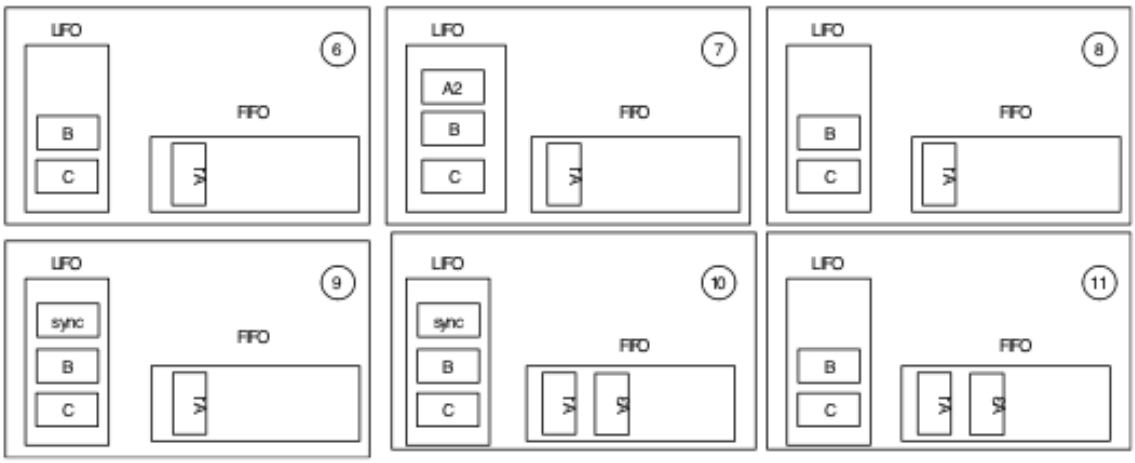


Принцип функционирования стека вызовов и очереди callback-функций

```
function A2(){ /* выполнение A2 */}
function A3(){ /* выполнение A3 */}
function B() { sync(()=>(A1())); A2(); sync(()=>(A3))  /* выполнение B */ }
function C() { B();  /* выполнение C */ }

C();
```



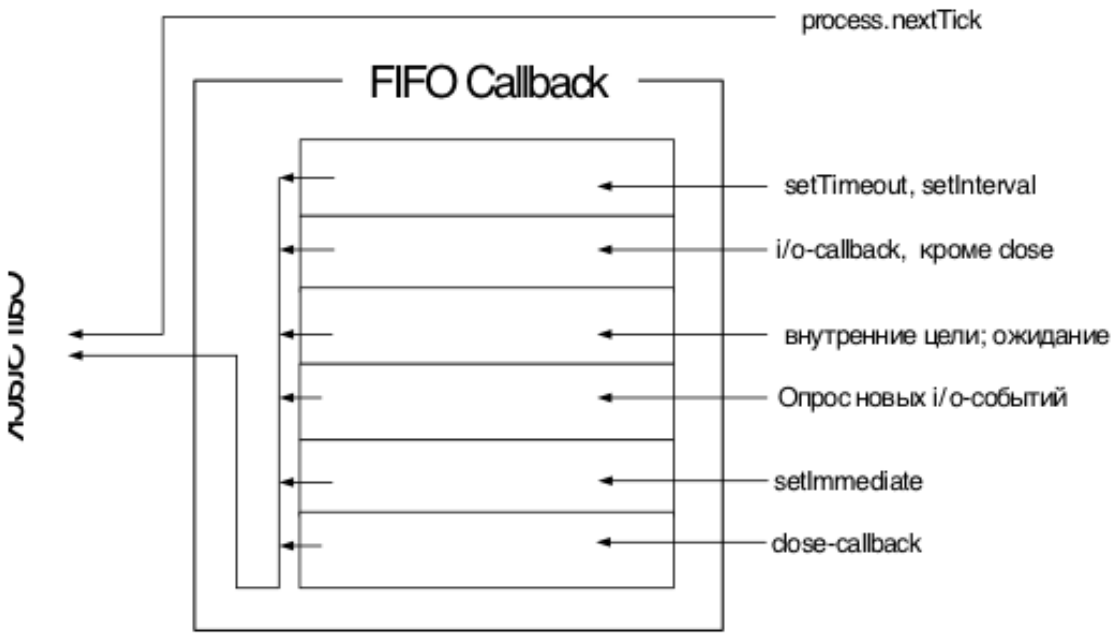

```
function A2(){ /* выполнение A2 */}
function A3(){ /* выполнение A3 */}
function B() { sync(()=>(A1())); A2(); sync(()=>(A3))  /* выполнение B */ }
function C() { B();  /* выполнение C */ }

C();
```



```
function A2(){ /* выполнение A2 */}
function A3(){ /* выполнение A3 */}
function B() { sync(()->{A1()}); A2(); sync(()->{A3})  /* выполнение B */ }
function C() { B();  /* выполнение C */ }

C();
```


```
function A2(){ /* выполнение A2 */}
function A3(){ /* выполнение A3 */}
function B() { sync(()->{A1()}); A2(); sync(()->{A3})  /* выполнение B */ }
function C() { B();  /* выполнение C */ }

C();
```


Структура Calback-очереди



Слева - Call Stack не пропечатался

Очереди в порядке приоритета располагаются. Вверху более приоритетные. При выборе руководствуется приоритетом. Для диспетчеризации доступа к потоку исполнения.

КЛАСС EventEmitter

EventEmitter **-** JS-класс, предоставляющий функциональность для асинхронной обработки событий в NODEJS**.** Входит в ядро NODEJS. Необходим чтобы генерировать события и уведомлять подписчиков, а также подписываться на события.

Событие в программном объекте – это процесс перехода объекта из одного состояние в другое. При этом, об этом переходе могут быть извещены другие объекты. У события есть **издатель** (или генератор) события и могут быть **подписчики** (или обработчики) события.

Для того, чтобы работать с **EventEmitter** необходимо включения двух модулей: ***events*** и ***util***.

EventEmitter: как правило, применяется в качестве базового для пользовательского объекта. Производный от **EventEmitter** объект может быть создан с помощью функции **inherits** модуля **utils**.

```
util.inherits(DB, ee.EventEmitter);
```

Производный от **EventEmitter** объект приобретает функциональность,

позволяющую генерировать и прослушивать события.

Для генерации событий предназначена функция **emit**, а для прослушивания функция **on**.

Таймер **-** механизм, позволяющий генерировать событие или выполнить некоторое действие, через заданный промежуток времени. Позволяет сделать отложенный вызов функции.

setTimeout(), setInterval() реализованы библиотекой **libuv**.

setTimeout() - 1 аргумент - функция обратного вызова, второй - время в миллисекундах, остальное - аргументы для функции обратного вызова. Дает указание внешней среде отсчитать какое-то количество миллисекунд и уведомить о том, что время закончилось. Внешняя среда отсчитывает время и уведомляет, потом SetTimeout ставит функцию в очередь колбэков и передает ей параметры. Срабатывает 1 раз, чтобы повторно использовать необходимо опять завести ее.

setInterval() - работает похоже, но работает периодически. 1 аргумент - функция обратного вызова, второй - периодичность вызова в миллисекундах, остальное - аргументы для функции обратного вызова.

clearTimeout() - отключить таймаут навсегда.

clearInterval() - удалить интервал навсегда.

NODEJS работает до тех пор, пока есть события, требующие обработки; если выполнить для таймера **unref** (чтобы события не учитывались как события, требующие обязательной обработки; не обращать внимание на обработку этих событий), то события, генерируемые таймером не будут учитываться при завершении работы Node.js, **ref** (обратный unref) – противоположная операция.

NODEJS-HTTP-сервер

HTTP - формат передачи данных по TCP. Полудуплексный - клиент отправляет запрос и в конце концов должен получить ответ.

Сокет - объект ОС, который предназначен для обмена данными по сети.

Самый низкоуровневый пакет - HTTP. Для создания HTTP сервера.

Очередь подключений - очередь, образованная сервером, заявок на выполнение accept.

Запрос поступает в виде битовой последовательности. NODEJS обрабатывает ее и создает 2 объекта: request и response.

keepAliveTimeout - время сохранения соединение (по умолчанию 5000). Если поступил запрос на установку соединения и в течение заданного времени оно не было установлено - отказ.

timeout - сообщить о бездействии (умолчание - 12000). Если клиент ничего не отправляет в течение заданного времени, то соединение будет разорвано.

Из сокета может быть получена та же информация, что и при работе в TCP-сервере.

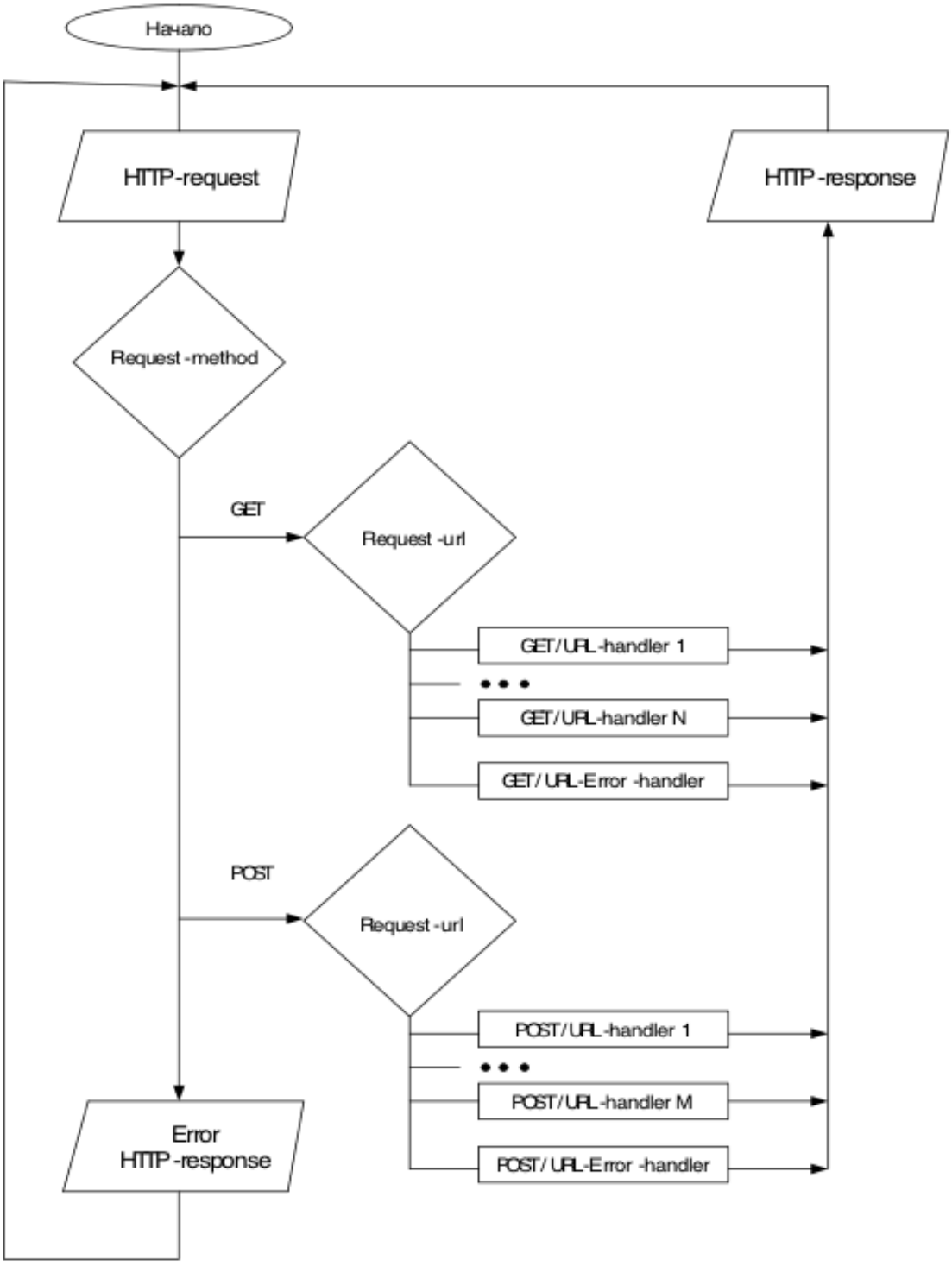
```
console.log('socket.localAddress = ', socket.localAddress);
console.log('socket.llocalPort = ',   socket.localPort);
console.log('socket.remoteAddress = ', socket.remoteAddress);
console.log('socket.remoteFamily = ', socket.remoteFamily);
console.log('socket.remotePort = ', socket.remotePort);
console.log('socket.bytesWritten = ', socket.bytesWritten);
```

Длина данных ограничивается размером пакета TCP. Если отправляем много байт, то все эти байты разбиваются (кадры - дейтаграммы - пакеты). Нету гарантии того, что наши данные придут за 1 порцию. Поэтому надо предусматривать возможность того, что данные могут прийти несколькими порциями. С помощью события data. Данные передаются в формате предусмотренном объектом buffer. Описывает октетную последовательность данных.

```
let buf='';
req.on('data', (data)=>{console.log('request.on(data) =', data.length); buf += data;}); // получить фрагментами
req.on('end',  ()=>{console.log('request.on(end) =', buf.length)})                      // все данные пришли
```

HTTP-сервер: простейший сервер, типичный цикл работы

Сначала проверяем метод, потом url, если есть - обрабатываем, нет - ошибка.



Статические ресурсы - данные, которые представляют собой файлы, которые хранятся на сервере и отправляются клиенту.

Второй параметр listen - очередь подключений. Функция accept выбирает клиентов из очереди подключений.

Модули

CommonJS **-** группа, которая проектирует, прототипирует и стандартизирует различные JavaScript API.

Модуль – фрагмент кода, специальным образом оформленный и размещенный, может использоваться приложением, является фундаментальной единицей структурирования кода Node.js-приложений. Текстовый файл, содержащий текст на языке JS.

Модуль используемый несколькими приложениями называют ***пакетом***.

Реализованные требования CommonJS

- поддержка **require** для импорта модуля;
- ***имя модуля*** – строка, может включать символы идентификации путей;
- модуль должен явно экспортировать всю свою функциональность, поддержка объекта ***export***;
- переменные внутри модуля не видимы за его пределами.

NODEJS: require

- первый require кэширует модуль, т.е повторной загрузки нет, всегда используется один и тот же экземпляр;
- удалить из кэша можно с помощью **delete require**;
- если модуль удален, то для его использования нужен новый **require**.

Разделить на 3 части

- В ядре
- Тот, который разработали сами, вручную
- Установили с помощью npm (в глобальный или локальный репозиторий)

Для локального пакета (устанавливается в папку с приложением) поиск осуществляется в **node_modules** по восходящему принципу. После поиска среди локальных пакетов, осуществляется поиск среди глобальных пакетов.

Если в качестве имени указана папка, то дополнительная информация в файле **package.json.**

pipe() - формирования из входного потока в выходной.

WebSocket-сервер

longpool запросы

Установка соединения по веб-сокету всегда по HTTP. Клиент сервер должны уметь работать с веб-сокетами. Клиент инициатор соединения.

Веб-сокет - это проток; соединение дуплексное (- есть 2 канала, 1 от

клиента к серверу, 2 - от сервера к клиенту). Надстройка над TCP.

TCP - дуплекс.

Браузерный объект.

Дуплексный поток - можем и писать, и читать.

Ping/pong для проверки соединения. Пинг проверяет соединения, другая сторона должна ответить понгом. Сервер делает понг автоматически.

rpc-websockets

Веб-сервис - веб-приложение, но в качестве клиента выступает другое приложение. Отправляют данные на сторону клиента. Все веб-сервисы делятся на 2 группы:

- Rest - интерфейс представляется в виде набора URI и параметров вызова
- RPC - удаленный вызов процедур. Есть 2 стандарта: SOAP и JSON-RPC.

Для HTTP - транспорт - TCP. У RPC транспорт WS в данном случае.

File System

Самый низкий уровень при работе с файловой системой - уровень API ОС. Там те элементарные системные вызовы, ниже которых мы спуститься не можем. Самое низкоуровневое программирование.

Синхронный/асинхронный ввод/вывод

Если в параметрах функции мы передаем функцию обратного вызова, то функция асинхронная. Иначе - асинхронная.

Если поток выполнения нашей программы приостанавливается при вызове функции ввода/вывода, то эта функция синхронная. Если не останавливает поток, то она асинхронная.

На уровне ОС существуют понятия синхронный/асинхронный ввод/вывод. Есть api для этого. Причем синхронный является частным случаем асинхронного. На самом деле весь ввод/вывод асинхронный. Просто пишется функция, которая вызывает асинхронный ввод/вывод и ждет завершения, поэтому нам кажется, что она синхронная.

NODEJS предоставляет некоторую надстройку над API ОС. Для работы с файловой системой есть модуль, который входит в ядро node.js - fs. Все операции могут быть либо синхронные, либо асинхронные.

Текущий директорий выделяет ОС, как правило тот, из которого запускается exe. Может быть и другой. В c++ где sln например.

Директорий - файл, который хранит ссылки на другие файлы.

unlink - удалить файл.

buffer - с помощью этого объекта можно создавать различные

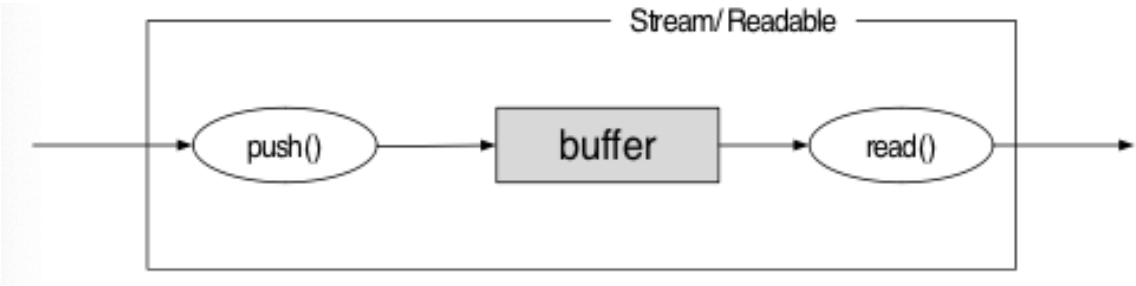
области памяти и заполнять их различными двоичными значениями.

Поток - абстракция над данными, которая позволяет нам представлять данные в виде последовательности последовательности байтов.

Виды:

- Writable - запись
- Readable - чтение
- Duplex - читать и писать, не обязательно в 1 файл
- Transform - *берет входной поток и трансформирует в выходной??*

Модуль stream для работы с потоками. Является ядерным. Позволяет программировать на нескольких уровнях.



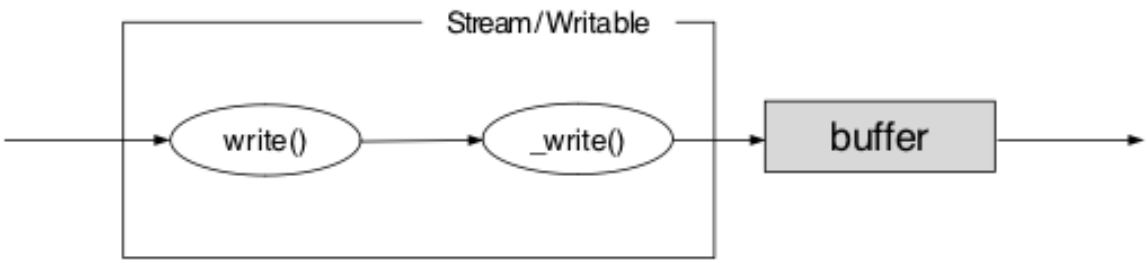
pipe() - в поток для чтения, берет данные из потока для чтения и переносит их в поток для записи.

```
const  Readable = require('stream').Readable;

let rs = new Readable();     // поток на чтение

rs.push('aaaaa');            // поместить в поток
rs.push('bbbbb');            // поместить в поток
rs.push('ccc\n');            // поместить в поток
rs.push(null);               // конец данных

rs.pipe(process.stdout);     // канал из потока в поток
```

```
D:\PSCA\Lec10>node 10-12.js
aaaaabbbbbccc
```

```
require('fs').createReadStream('./Files/File14.txt').pipe(process.stdout);
```

_read() - чтение входного потока.




```
const  Writable = require('stream').Writable;
let    ws = Writable();
ws._write =  (chunk, enc, next) => {
    console.log('_write:', chunk.toString());
    next();
};
process.stdin.on('data', (chunk)=>{ ws.write(chunk);})
```

```
D:\PSCA\Lec10>node 10-17
jj
_write: jj

kkk
_write: kkk

gg
_write: gg
```

MSSQL

1433 порт зарезервирован для mssql
1521 для oracle

prepare statement - позволяет делать динамические запросы. Можно для select, insert, delete, update (прим в лк)

```
ps.input('mincap', sql.Int);
ps.input('maxcap', sql.Int);
ps.prepare('select auditorium_name, auditorium_capacity '
        + 'from AUDITORIUM WHERE auditorium_capacity between  @mincap AND @maxcap',
    if (err) cb(err, null);
    else  ps.execute({mincap: mincap, maxcap: maxcap}, cb);
```

Пул соединений - специальное ПО, которое позволяет сразу же открыть несколько соединений с бд. Причем существует 2 параметра: минимальное и максимальное количество соединений. При старте от автоматически открывает минимальное количество соединений и те запросы, что приходят на сервер, начинают занимать эти соединения. Если запросов больше, то пул открывает новое соединение. До тех пор пока не достигнет максимального значения соединений. Если пришло

больше, чем максимальное, то возникает очередь пула соединений. Запрос ждет. Таким образом, это некоторый набор соединений, который увеличивается до максимального по мере необходимости + очередь.

Интерфейсы: ado, jdbc, odbc, oledbc

MongoDB

Насколько данные сложные зависит использование sql бд или nosql.

Структура в sql бд не зависит от использования, а от природы данных.

Nosql не следим за нормализацию, чтобы данные были построены правильно. Основная задача - построить данные так, чтобы они были удобные для приложения. Часто такие бд применяются совместно с бд sql как кэширующие бд.

MongoDB - это набор коллекций. Каждый набор коллекции - это JSON-объект. Есть возможность индексировать коллекции.