JS

JS Core

1. https://jsfiddle.com

programming design patterns

SOLID principles

- программыне сущности: классы, модули, функции и т.п.
- Single Responsibility Principle
  - Принцип единственной обязанности
  - На каждый объект должна быть возложена одна единственная обязанность.
  - Разбивать объекты на более мелкие
- Open/Closed Principle
  - Принцип открытости/закрытости
  - Программные сущности должны быть открыты для расширения, но закрыты для изменения
- Liskov Substitution Principle
  - Принцип подстановки Барбары Лисков
  - Объекты в программе могут быть заменены их наследниками без изменения свойств программы
  - потомок должен дополнять, но не замещать методы родителя
- Interface Segregation Principle
  - Принцип разделения интерфейса
  - Клиенты не должны реализовывать ненужные методы, которые они не будут использовать
  - Разбивать интерфейсы на более мелкие
- The Dependency Inversion Principle DI
  - Критический взгляд на принцип инверсии зависимостей
  - Принцип инверсии зависимостей
  - Реализации на основе абстракций.
  - реализация зависит от абстракции, но не наоборот
  - модули должны зависеть от абстракций
  - модули верхнего уровня должны зависеть от модулей низкого

JS

- декларативный(.forEach) и императивный(for)
- ECMAScript — это язык программирования
  - объектно-ориентированный
  - с прототипной организацией
  - концепция объекта в качестве базовой абстракции
  - динамическая типизация

V8

- интерпретатор ES+WASM ignition/байткод
- компиляторturbofan JIT
- виртуальные машины
- однопоточный, а потому неблокирующий
- FIFO?
- динамическая типизация
  - примитивные типы присваивают значение, объекты - ссылки
  - https://codeburst.io/js-scope-static-dynamic-and-runtime-augmented-5abfee6223fe
- прототипное наследование
  - proto - sorax
  - https://dmitrysoshnikov.medium.com/oo-relationships-5020163ab162
  - почти у всех объектов есть прототип для хранения шаблона объекта, создания экземпляров через конструктор
  - класс - объект-шаблон для создания других объектов, экземпляров класса
  - Object.isPrototypeOf()
  - Object.instanceOf()
  - наследование
    - в конструкторе: `parentCLass.constructor.apply(this,arguments)`
    - в прототипе:
    -
  - `new` для определения нового `this` в новом экземпляре объекта/класса. Без new `this === undefined` в ES5, раньше - `window`

GC

- allocate,use,release memory
- задача точного выяснения того, нужен ли определенный участок памяти, алгоритмически неразрешима.
- https://developer.mozilla.org/en-US/docs/Web/JavaScript/Memory_Management
- вместо алгоритма подсчёта ссылок сейчас браузеры применяют алгоритм проверки доступности(mark-compact). Например, циклические ссылки не считаются мусором в первом варианте, даже если на них нет ссылок извне.
- https://geekbrains.ru/posts/javascript_internals_part2
- активные функции(например, глобальные переменные) помечаются как корни, и от них проверяется доступность
- в отдельном потоке
- очищает не всю кучу, а последовательно по кускам через время
- use strict помогает избежать случайного объявления глобальной переменной(переменная без var или вызов функции с обращением к this.foo в глобальной области)
- глобальные переменные не очищаются, т.к. они корень
- забытые замыкания, со ссылками на глобальные переменные
- забытые таймеры, события, js ссылки на DOM - они не удаляются вместе с удалением узла

Event loop

- What the heck is the event loop anyway? | Philip Roberts | JSConf EU
- Jake Archibald: In The Loop - JSConf.Asia
- https://nodejs.dev/learn/the-nodejs-event-loop
- https://developer.mozilla.org/ru/docs/Web/JavaScript/EventLoop
- https://html.spec.whatwg.org/multipage/webappapis.html#event-loops
- Jake Archibald: все что я знаю про Event Loop в JavaScript (2018)
- https://medium.com/@olinations/the-javascript-runtime-environment-d58fa2e60dd0
- https://jakearchibald.com/2015/tasks-microtasks-queues-and-schedules/
- приоритеты
    1. макрозадачи(без DOM, в тестах, вызов событий тоже в этой очереди)
    2. микрозадачи(после очистки макрозадач) - Promise.then()
    3. микрозадачи с задержкой - setTimeout(cb, time)
    4. отрисовка
- планирование задач
    - задачи про DOM --> RAF
        - https://developer.mozilla.org/en-US/docs/Web/API/window/requestAnimationFrame
        - https://developer.mozilla.org/en-US/docs/Web/API/Window/cancelAnimationFrame
        -

```
    let id = window.requestAnimationFrame(cb:
(timestamp)=>number); cancelAnimationFrame(id);
```

        - запускает обратный вызов перед рендерингом, возвращает номер для отмены обещания
        - стили могут не применяться, если перетирают друг друга
        - getComputedStyle() заставит применить стили после назначения свойств и отрисовать их на GPU
        - до 2019 safari выполнял RAF после рендеринга https://bugs.webkit.org/show_bug.cgi?id=177484
    - фоновые --> requestIdleCallback
        - https://developer.mozilla.org/en-US/docs/Web/API/Window/requestIdleCallback
        -

```
    let id = window.requestIdleCallback(cb:()=>number,options?:
{timeout:number})
```

    -
  - тяжёлая --> webworker
    - общение с DOM через postMessage
      - https://developer.mozilla.org/en-US/docs/Web/API/Window/postMessage
      - общение через адрес страницы
    - нет DOM, событий
    - есть XHR, promise
    - свой контекст, вне CSP документа

```
    // worker.js
    onmessage = function(e) {
        console.log('Worker: Message received from main script');
        const result = e.data[0] * e.data[1];
        if (isNaN(result)) {
            postMessage('Please write two numbers');
        } else {
            const workerResult = 'Result: ' + result;
            console.log('Worker: Posting message back to main
script');
            postMessage(workerResult);
        }
    }

    // index.js
    if (window.Worker) {
        const myWorker = new Worker("worker.js");

        first.onchange = function() {
            myWorker.postMessage([first.value, second.value]);
            console.log('Message posted to worker');
        }

        myWorker.onmessage = function(e) {
            result.textContent = e.data;
            console.log('Message received from worker');
        }
    }

    myWorker.terminate();
```

  - приоритетная --> setTimeout(cb, 0)
    - понижает приоритет отрисовки
    - добавляет событие в очередь

- очередь задач queue task
  - привязаны источники событий

    - мышь
    - клава
    - XHR

- serviceworker
    - работа по сети, кэширование
    - являются web worker'ами под капотом

```
if ('serviceWorker' in navigator) {
window.addEventListener('load', function() {
    navigator.serviceWorker.register('/sw.js');
});
}
```

- DOM
    - Critical Rendering Path: DOM, CSSOM, Render tree, Layout, Paint
    - https://developer.mozilla.org/ru/docs/Web/Performance/Critical_rendering_path
    - Построение DOM инкрементально. Ответ в виде HTML превращается в токены, которые превращаются в узлы (nodes). Узлы (nodes) связаны с Render Tree с помощью иерархии токенов.
    - Чем больше количество узлов (node) имеет приложение, тем дольше происходит формирование DOM tree
    - Если формирование DOM инкрементально, CSSOM - нет. CSS блокирует рендер: браузер блокирует рендеринг страницы до тех пор, пока не получит и не обработает все CSS-правила. CSS блокирует рендеринг, потому что правила могут быть перезаписаны
    - CSS-правила ниспадают каскадом, вложенные узлы наследуют стили от родительских.
    - наименее специфичные селекторы срабатывают быстрее.
    - Дерево рендера охватывает сразу и содержимое страницы, и стили: это место, где DOM и CSSOM деревья комбинируются в одно дерево. Для построения дерева рендера браузер проверяет каждый узел (node) DOM, начиная от корневого (root) и определяет, какие CSS-правила нужно присоединить к этому узлу.
    - Дерево рендера охватывает только видимое содержимое. Например, секция head может не включаться в дерево. display: none так же не включается в дерево (как и потомки этого узла).
    - В тот момент, когда дерево рендера (render tree) построено, становится возможным этап компоновки (layout). Компоновка зависит от размеров экрана.
    - отрисовка (paint) пикселей на экране

- микрозадачи
    - resolved promise
    - mutation observer(нельзя добавить второй наблюдатель, пока не пустая очередь микрозадач)
    - intersection observer
    - стартуют после очистки очереди макрозадач

    - выполняются до очистки очереди микрозадач
    - могут порождать новые микрозадачи

- ```
  console.log(1);
  const promise = new Promise(resolve => {  resolve() });
  promise.then(() => { setTimeout(() => {  console.log(2); })});
  setTimeout(() => {  console.log(3); }, 500);
  setImmediate(() => { console.log(4);});
  setTimeout(() => { console.log(5); }, 0);
  promise.then(() => { console.log(6);});
  console.log(7);


  // 1 7 6 undefined 4 5 2 3
  ```

Новое в ES6 https://tc39.es/ecma262/

- ES6 по-человечески
- let, const
    - let, const - блочная область видимости(scope), не всплывают, не добавляются в this, не существуют до своего объявления
    - const - неизменяемое значение(примитив), ссылка(объект). Object.freeze() позволяет защитить первый уровень вложенных объектов.
    - var - функциональная область видимости, требуют изоляции в замыканиях
- Стрелочные функции
    - не имеют своего this, arguments
- Параметры функций по умолчанию
- Spread/Rest оператор
- Расширение возможностей литералов объекта
- Восьмеричный и двоичный литералы
- Деструктуризация массивов и объектов
- Ключевое слово super для объектов
- Строковые шаблоны и разделители
- for...of , for...in
    - for of
        - нельзя пользовать аннотации
        - по итерируемым свойствам
        - Array, nodeList
        -
    - for in
        - по перечисляемым свойствам, игнорируя неитерируемые
        - нет строгого порядка обхода
        - нежелательно менять/добавлять свойства до прохода итератора
        - нельзя деструктурирующие присваивания
        - object
- Map/set
  ```

    -
- WeakMap/WeakSet
    - Weak поощряет сборку мусора, т.к. содержит меньше ссылок
    - нельзя итерировать, т.к. нет итератора
- Классы в ES6
- Тип данных Symbol
    - https://www.programiz.com/javascript/symbol
    - https://www.javascripttutorial.net/es6/symbol/
    - https://medium.com/intrinsic/javascript-symbols-but-why-6b02768f4a5c
    - https://javascript.info/symbol
    - не перечисляемый(enumerable)
- Итераторы
    - ? ссылки на все элементы
- Генераторы
- Classes
- Promises
- Symbol
- String.includes()
- String.startsWith()
- String.endsWith()
- Array.from()
- Array keys()
- Array find()
- Array findIndex()
- New Math Methods
- New Number Properties
- New Number Methods
- New Global Methods
- Object entries
- JavaScript Modules

ES2021(12)

- https://levelup.gitconnected.com/top-5-javascript-es12-features-you-should-start-using-now-b16a8b5353b1
    - promise.any()
    - ||= Logical OR assignment operator
        - if false
        - `a || (a = b)`
    - &&= Logical AND assignment operator
        - is true
        - `a && (a = b)`
    - ??= Nullish coalescing assignment operator
        - is null or undefined
        - `a ?? (a = b)`

ES modules

- https://v8.dev/features/modules#mjs
- https://hacks.mozilla.org/2015/08/es6-in-depth-modules/
    - автоматически "use strict"
    - можно делать import/export
-

```
// <script type="module" src="main.js"></script>

// Aggregating modules
export * from 'nested1.js'
export { name } from 'nested2.js'

export {a,b,c}
const a = false, b = 0, c = '';

// Dynamic module loading
import('./modules/myModule.js')
    .then((module) => {
        // Do something with the module.
    });

// Top level await
const colors = fetch('../some.json')
    .then(response => response.json());

export default await colors;
// import colors from './modules/getColors.js';
```

- в браузере пока лучше использовать .js вместо .mjs - нужен `Content-Type text/javascript`. Иначе будет strict MIME type checking error: `"The server responded with a non-JavaScript MIME type"`
- локально через `file://` не работает - CORS
- по-умолчанию strict mode
- по-умолчанию используют defer script attribute
- модули исполняются только один раз, даже в нескольких