

Лекция 7

Крутые мелочи

Браузерная поддержка

Подробнее тут <https://habr.com/ru/companies/nix/articles/342904/>

Подробнее тут <https://frontend.tech-mail.ru/slides/s8/>

Из чего состоит JavaScript?

- ECMA-262
- ECMAScript
- JavaScript
- JavaScript-движок
- TC39

Ecma International

Организация, которая создает стандарты для технологий.

ECMA-262

Это стандарт, изданный Ecma International. В нём прописана спецификация скриптового языка общего назначения.

ECMA-262 — это стандарт, подобный QWERTY, только представляющий собой спецификацию скриптового языка, называющегося ECMAScript. ECMA-262 можно считать учётным номером ECMAScript.

ECMAScript

Описанная в ECMA-262 спецификация создания скриптового языка общего назначения. (спецификация ECMAScript)

«ECMA-262» — это название и стандарта, и спецификации скриптового языка ECMAScript. ECMAScript содержит правила, сведения и рекомендации, которые должны соблюдаться скриптовым языком, чтобы он считался совместимым с ECMAScript.

ECMAScript (как выглядит)

- Декабрь 1999 — ECMAScript 3
- ECMAScript 4 (**abandoned**) — заброшенная версия
- Декабрь 2009 — ECMAScript 5
- Июнь 2011 — ECMAScript 5.1
- **Июль 2015 — ECMAScript 2015 (ECMAScript 6th edition)**
- Июль 2016 — ECMAScript 2016 (ECMAScript 7th edition)
- Июнь 2017 — ECMAScript 2017 (ECMAScript 8th edition)
- *Лето 2018* — ECMAScript 2018 (и так далее)

ES.Next

ES.Next — так временно называют совокупность новых возможностей языка, которые могут войти в следующую версию спецификации. Фичи из ES.Next правильнее называть “предложения” (proposals) , потому что они всё ещё находятся на стадии обсуждения

JavaScript

Скриптовый язык общего назначения, соответствующий спецификации ECMAScript.

Это диалект языка ECMAScript.

JavaScript-движок

Программа или интерпретатор, способный понимать и выполнять JavaScript-код.

JavaScript-движки обычно используются в веб-браузерах, включая V8 в Chrome, SpiderMonkey в Firefox и Chakra в Edge. Каждый движок подобен языковому модулю, который позволяет приложению поддерживать определенное подмножество языка JavaScript.

JavaScript-движок (быстродействие)

Два человека поймут команду JavaScript, но один из них отреагирует раньше, потому что смог быстрее понять и обработать команду. Аналогично, два браузера могут понимать код JavaScript, но один из них работает быстрее, потому что его JavaScript-движок работает эффективнее.

JavaScript-движок (поддержка)

Разные браузеры могут понимать не все команды JavaScript. Говоря о поддержке в браузерах, обычно упоминают о «совместимости с ECMAScript», а не о «совместимости с JavaScript», хотя JavaScript-движки детально анализируют и выполняют JavaScript.

ECMAScript — это спецификация того, как может выглядеть скриптовый язык. Появление новой версии ECMAScript не означает, что у всех движков JavaScript появятся новые функции. Всё зависит от групп или организаций, которые отвечают за обновления JavaScript-движков с учётом новейшей спецификацией ECMAScript.

Среда выполнения JavaScript

В этой среде выполняется JavaScript-код и интерпретируется JavaScript-движком. Среда выполнения предоставляет хост-объекты, на которых и с которыми может работать JavaScript.

Среда выполнения JavaScript — это «существующий объект или система», упомянутые в определении скриптового языка. Код проходит через JavaScript-движок, в котором объект или система анализирует код и разбирает его работу, а потом выполняет интерпретированные действия.

JavaScript-скрипты могут обращаться к приложениям, потому что те предоставляют «хост-объекты» в среде выполнения. На клиентской стороне средой выполнения JavaScript будет веб-браузер, в котором становятся доступными для манипуляций такие хост-объекты, как окна и HTML-документы. На серверной стороне среда выполнения JavaScript — это Node.js. В Node.js предоставляются связанные с сервером хост-объекты, такие как файловая система, процессы и запросы.

Курица или яйцо

JavaScript был создан в 1996 году. В 1997 году Ecma International предложила стандартизировать JavaScript, и в результате появился ECMAScript. Но поскольку JavaScript соответствует спецификации ECMAScript, JavaScript является примером реализации ECMAScript.

Получается, что ECMAScript основан на JavaScript, а JavaScript основан на ECMAScript.

Процесс ТС39

TC39

TC39 (технический комитет 39) — занимается развитием **JavaScript**. Его членами являются компании (помимо прочих, все основные производители браузеров). TC39 регулярно собирается, на встречах присутствуют участники, представляющие интересы компаний, и приглашенные эксперты

Процесс TC39

Процесс TC39 — алгоритм внесения изменений в спецификацию ECMAScript. Каждое предложение по добавлению новой возможности в ECMAScript в процессе созревания проходит ряд этапов

0 этап: идея (strawman)

1 этап: предложение (proposal)

2 этап: черновик (draft)

3 этап: кандидат (candidate)

4 этап: финал (finished)

TC39

TC39 (технический комитет 39) — занимается развитием **JavaScript**. Его членами являются компании (помимо прочих, все основные производители браузеров). TC39 регулярно собирается, на встречах присутствуют участники, представляющие интересы компаний, и приглашенные эксперты

Зачем нам об этом знать?

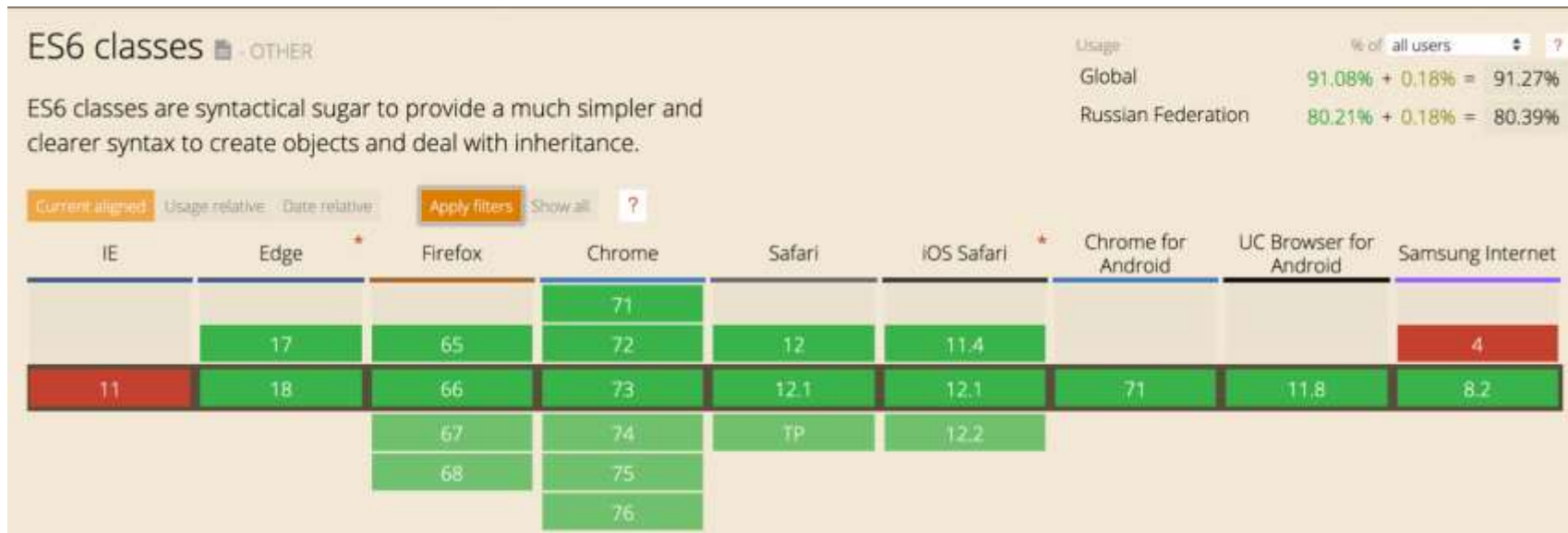
Поддержка версий ECMAScript

<https://kangax.github.io/>

COMPAT ES ECMAScript 5 6 2016+ next intl non-standard compatibility table							
Sort by Engine types		Show obsolete platforms		Show unstable platforms			
Feature name	Current browser	Traceur	Babel + core-js ^[2]	Closure	Type-Script + core-js	es7-shim	IE 11
2016 features		83%	10%	40%	16%	36%	16%
• exponentiation (**) operator	3/3	2/3	3/3	3/3	2/3	0/3	0/3
• Array.prototype.includes	3/3	0/3	3/3	0/3	3/3	2/3	0/3
2016 misc							
• generator functions can't be used with "new"^[7]	Yes	No	No	No	No	No	No

Возможности браузера

<http://caniuse.com/>



Как нам использовать последний
функционал JavaScript и
поддерживать все браузеры?

Полифиллы

Полифилл — это библиотека, которая добавляет в старые браузеры поддержку возможностей, которые в современных браузерах являются встроенными

```
1 if (!Object.is) {  
2     Object.is = function(x, y) {  
3         if (x === y) { return x !== 0 || 1 / x === 1 / y; }  
4         else { return x !== x && y !== y; }  
5     }  
6 }
```

Транспайлинг

Транспайлинг — это конвертация кода программы, написанной на одном языке программирования в другой язык программирования

```
1 // before
2 const f = num => `${num} в квадрате это ${num ** 2}`;
3
4 // after
5 var f = function (num) {
6     return num + ' в квадрате это ' + Math.pow(num, 2);
7 };
```


Babel

Babel — многофункциональный транспайлер, позволяет транспилировать ES5, ES6, ES2016, ES2017, ES2018, ES.Next, JSX и Flow

Babel (использование)

```
1 # устанавливаем модуль
2 $ npm install -D @babel/core @babel/cli @babel/preset-env
3
4 # файл с конфигурацией
5 $ nano .babelrc
6 {
7   "presets": [[
8     "@babel/env",
9     { targets: {edge: "15"}}
10  ]]
11 }
12
13 # запускаем
14 $ babel modern.js --watch --out-file compatible.js
```

Babel (как работает)

- Парсит исходный код и **строит AST**
- Последовательно вызывает набор функций, которые каким-то образом **трансформируют AST** программы
- В процессе трансформации части AST, относящиеся к современному синтаксису, **заменяются на эквивалентные**, но более общеупотребительные фрагменты
- Преобразует модифицированное AST в **новый транспирированный код**

Babel (как работает)

```
1 // ES6
2 const sum = (a, b) => a + b;
3
4 // ES5
5 var sum = function sum(a, b) {
6     return a + b;
7 };
```

А что делать с CSS?

Постпроцессоры (PostCSS)

Постпроцессор — это программа на вход которой дается css, а на выходе получается css.

Постпроцессоры (как работает)

- Исходный файл дается на вход PostCSS и парсится
- Плагин 1 что-то делает
- ...
- Плагин n что-то делает
- Полученный результат преобразовывается в строку и записывается в выходной файл

Постпроцессоры (autoprefixer)

```
1 //in.css
2 div {
3     display: flex
4 }
5
6 //out.css
7 div {
8     display: -webkit-box;
9     display: -webkit-flex;
10    display: -moz-box;
11    display: -ms-flexbox;
12    display: flex
13 }
```


Постпроцессоры (Preset Env)

```
1 //in.css
2 @custom-media --med (width <= 50rem);
3
4 @media (--med) {
5   a:hover {
6     color: color-mod(black alpha(54%));
7   }
8 }
9
10 //out.css
11 @media (max-width: 50rem) {
12   a:hover {
13     color: rgba(0, 0, 0, 0.54);
14   }
15 }
```

Постпроцессоры (CSS Modules)

```
1 //in.css
2 .name {
3   color: gray;
4 }
5
6 //out.css
7 .Logo__name__SVK0g {
8   color: gray;
9 }
```

Bundlers

Подробнее тут <https://habr.com/ru/companies/vk/articles/340922/>

Bundler

Bundler — программа, которая упаковывает сложный проект со многими файлами и внешними зависимостями в один (иногда несколько) файл, который будет отправлен браузеру.

Bundler

```
1 <head>
2   <script src="https://cdnjs.cloudflare.com/ajax/libs/moment.js/2.29.4/moment.min.js"></script>
3   <script src="https://cdnjs.cloudflare.com/ajax/libs/axios/1.4.0/axios.min.js"></script>
4   <script src="index.js"></script>
5   <script src="auth.js"></script>
6   <script src="reg.js"></script>
7 </head>
```

Bundler

```
1 <head>
2   <script src="bundle.js"></script>
3 </head>
```

Чистый html

```
1 // index.js
2 console.log("Hello from JavaScript!");
3 console.log(moment().startOf('day').fromNow());
```

Чистый html

```
1 <!-- index.html -->
2 <!DOCTYPE html>
3 <html lang="en">
4 <head>
5   <meta charset="UTF-8">
6   <title>Example</title>
7   <link rel="stylesheet" href="index.css">
8   <script src="https://cdnjs.cloudflare.com/ajax/libs/moment.js/2.29.4/moment.min.js"></script>
9   <script src="index.js"></script>
10 </head>
11 <body>
12   <h1>Hello from HTML!</h1>
13 </body>
14 </html>
```


Npm

```
1 npm install moment --save
```

Npm

```
1 // index.js
2 console.log("Hello from JavaScript!");
3 console.log(moment().startOf('day').fromNow());
```

Npm

```
1 <!-- index.html -->
2 <!DOCTYPE html>
3 <html lang="en">
4 <head>
5   <meta charset="UTF-8">
6   <title>JavaScript Example</title>
7   <script src="node_modules/moment/min/moment.min.js"></script>
8   <script src="index.js"></script>
9 </head>
10 <body>
11   <h1>Hello from HTML!</h1>
12 </body>
13 </html>
```

Webpack + Npm + CommonJS

```
1  npm install webpack --save-dev
```

Webpack + Npm + CommonJS

```
1 // index.js
2 var moment = require('moment');
3 console.log("Hello from JavaScript!");
4 console.log(moment().startOf('day').fromNow());
```

Webpack + Npm + CommonJS

```
1 // webpack.config.js
2 module.exports = {
3   entry: './index.js',
4   output: {
5     filename: 'bundle.js'
6   }
7 };
8
```

Webpack + Npm + CommonJS

```
1 npx webpack
```

Webpack + Npm + CommonJS

```
1 <!-- index.html -->
2 <!DOCTYPE html>
3 <html lang="en">
4 <head>
5   <meta charset="UTF-8">
6   <title>JavaScript Example</title>
7   <script src="bundle.js"></script>
8 </head>
9 <body>
10  <h1>Hello from HTML!</h1>
11 </body>
12 </html>
```


Webpack + Npm + ESM

```
1 // index.js
2 import moment from 'moment';
3 console.log("Hello from JavaScript!");
4 console.log(moment().startOf('day').fromNow());
```

Webpack + Babel

```
1 // webpack.config.js
2 module.exports = {
3   entry: './index.js',
4   output: {
5     filename: 'bundle.js'
6   },
7   module: {
8     rules: [
9       {
10        test: /\.js$/,
11        exclude: /node_modules/,
12        use: {
13          loader: 'babel-loader',
14          options: {
15            presets: ['env']
16          }
17        }
18      }
19    ]
20  }
21 };
```

Webpack + PostCSS

```
1 // webpack.config.js
2 module.exports = {
3   entry: './index.js',
4   output: {
5     filename: 'bundle.js'
6   },
7   module: {
8     rules: [
9       {
10        test: /\.css$/i,
11        use: {
12          loader: "postcss-loader",
13          options: {
14            // ...
15          },
16        },
17      },
18    ],
19  },
20 };
```

TypeScript

Подробнее тут <https://frontend.tech-mail.ru/slides/s8/>

Зачем нам нужен новый язык?

JavaScript не типизируемый

Типизация

```
1 function print(a) {  
2   console.log(a.data.name)  
3   console.log(a.data.surname)  
4 }
```

```
1 print({  
2   name: 'Ivan',  
3   surname: 'Ivanov',  
4 })
```

```
1 print({  
2   data: {  
3     name: 'Ivan',  
4     surname: 'Ivanov',  
5   }  
6 })
```

Больше новых языков

Dart programming language

```
1 import 'dart:async';
2 import 'dart:math' show Random;
3
4 Stream<double> computePi({int batch: 1000000}) async* { ... }
5
6 main() async {
7     print('Compute  $\pi$  using the Monte Carlo method.');
```

8 await for (var estimate in computePi()) {

9 print('π ≅ \$estimate');

10 }

11 }

CoffeeScript

```
1 class Human
2   constructor : (@name) ->
3
4 class Baby extends Human
5   say      : (msg) -> alert "#{@name} говорит '#{msg}'"
6   saymsg   = (msg) -> alert msg
7   @echo    = (msg) -> console.log msg
8
9 matt = new Baby("Матвей")
10 matt.sayHi()
```

CoffeeScript

```
1 if happy and knowsIt
2   lyrics = while num -= 1
3     "#{num} little monkeys, jumping on the bed"
4 else
5   date = if friday then sue else jill
6
7 for filename in list
8   do (filename) ->
9     fs.readFile filename, (err, contents) ->
10      compile filename, contents.toString()
```

ClojureScript

```
1 (ns hello-world.core
2   (:require [cljs.nodejs :as nodejs]))
3
4 (nodejs/enable-util-print!)
5
6 (defn -main [& args]
7   (println "Hello world!"))
8
9 (set! *main-cli-fn* -main)
```

Elm

```
1 import Html exposing (text)
2
3 main =
4   text (toString (zip ["Tom", "Sue", "Bob"] [45, 31, 26]))
5
6 zip : List a -> List b -> List (a,b)
7 zip xs ys =
8   case (xs, ys) of
9     ( x :: xBack, y :: yBack ) ->
10      (x,y) :: zip xBack yBack
11
12   (_, _) ->
13     []
```

TypeScript

```
1 interface Person {  
2     name: string;  
3     age: number;  
4 }  
5 function meet(person: Person) {  
6     return `Привет, я ${person.name}, мне ${person.age}`;  
7 }  
8 const user = { name: "Jane", age: 21 };  
9 console.log(meet(user));
```

TypeScript

TypeScript — язык программирования, представленный Microsoft в 2012 году. TypeScript является **обратно совместимым** с JavaScript и компилируется в последний. TypeScript отличается от JavaScript возможностью явного статического назначения типов, а также поддержкой подключения модулей

Преимущества TypeScript

- Аннотации типов и проверка их согласования на этапе компиляции
- Интерфейсы, кортежи, декораторы свойств и методов, расширенные возможности ООП
- TypeScript — **надмножество JavaScript**, поэтому любой код на JavaScript будет выполнен и в TypeScript
- Широкая поддержка IDE и адекватный автокомплит
- Поддержка ES6-модулей из коробки

Переписываем проект на TS

- Переименовываем ***.js** в ***.ts**
- ???????
- PROFIT

Установка и использование

```
1 # установка компилятора
2 $ npm install -g typescript
3
4 # компиляция файла
5 $ tsc helloworld.ts
6
7 # утилита позволяет компилировать и сразу запускать .ts файлы
8 $ npm install -g ts-node
9 $ ts-node script.ts
```

Файл tsconfig.json

```
1 {  
2     "compilerOptions": {  
3         "outDir": "cache/",  
4         "target": "es2016",  
5         "declaration": false,  
6         "module": "commonjs",  
7         "strictNullChecks": true,  
8         "sourceMap": true  
9     }  
10 }  
11 }  
12
```

Запускаем

```
1 npx tsc
```

TS -> JS

```
1 function sum(a: number, b: number): number {  
2     return a + b  
3 }
```

```
1 function sum(a, b) {  
2     return a + b  
3 }
```

Типизация

```
1 interface User {  
2   data: {  
3     name: string  
4     surname: string  
5   }  
6 }  
7  
8 function print(a: User) {  
9   console.log(a.data.name)  
10  console.log(a.data.surname)  
11 }
```

```
1 print({  
2   data: {  
3     name: 'Ivan',  
4     surname: 'Ivanov',  
5   }  
6 })
```

```
1 print({  
2   name: 'Ivan', // Ошибка  
3   surname: 'Ivanov', // Ошибка  
4 })
```

Аннотации типов

```
1 const valid: boolean = true;
2 const count: number = 42;
3 const man: string = 'Jon Snow';
4
5 console.log(man * 2);
6 // Error: The left-hand side of an arithmetic
7 // operation must be of type 'any', 'number' or an enum type
```

Вывод типов

```
1 const valid = true;
2 const count = 42;
3 const man = 'Jon Snow';
4
5 console.log(man * 2);
6 // Error: The left-hand side of an arithmetic
7 // operation must be of type 'any', 'number' or an enum type
```


Аннотации типов

```
1 const valid = true;
2 const count = 42;
3 const man: any = 'Jon Snow';
4
5 console.log(man * 2); //NaN
6 // Зато нет TS ошибок!
7 // Profit!
8
```

Аннотации типов

```
1 const valid = true;
2 const count = 42;
3 const name = 'Jon Snow';
4
5 const values: number[] = [1, 2, 3, 4, 5];
6 const tuple: [string, number] = ['Mean of life', 42];
7
8 enum Color {Red, Green, Blue};
9 const c: Color = Color.Green;
10
```

Функции в TypeScript

```
1 function sum(x: number, y: number): number {  
2     return x + y;  
3 }  
4  
5 const many: number = sum(40, 2);  
6  
7 const gcd = (a: number, b: number): number =>  
8     (b === 0) ? a : gcd(b, a % b);  
9  
10 console.log(gcd(48, 30)); // 6  
11
```

Функции в TypeScript

```
1 function sum(x: number, y?: number): number {  
2     if (y) {  
3         return x + y;  
4     } else {  
5         return x;  
6     }  
7 }  
8  
9 console.log(sum(34, 8)); // 42  
10 console.log(sum(42));   // OK! - 42
```

Функции в TypeScript

```
1 function sum(x: number, y: number = 42): number {  
2     return x + y;  
3 }  
4  
5 console.log(sum(34, 8)); // 42  
6 console.log(sum(42));    // OK! - 84
```

Функции в TypeScript

```
1 function sum(...numbers: number[]): number {  
2     return numbers.reduce((sum: number, current: number): number => {  
3         sum += current; return sum;  
4     }, 0);  
5 }  
6  
7 console.log(sum(1, 2, 3, 4, 5));    // 15  
8 console.log(sum(42, 0, -10, 5, 5)); // 42  
9
```

Функции в TypeScript

```
1 function square(num: number): number;  
2 function square(num: string): number;  
3 function square(num: any): number {  
4     if (typeof num === 'string') {  
5         return parseInt(num, 10) * parseInt(num, 10);  
6     } else {  
7         return num * num;  
8     }  
9 }  
10
```

Функции в TypeScript

```
1 function square(num: string | number): number {  
2     if (typeof num === 'string') {  
3         return parseInt(num, 10) * parseInt(num, 10);  
4     } else {  
5         return num * num;  
6     }  
7 }
```


Интерфейсы

```
1 interface Figure {  
2     width: number;  
3     readonly height: number;  
4 }  
5  
6 const square: Figure = {width: 42, height: 42};  
7 square.width = 15;    // OK  
8 square.height = 15;   // Cannot assign to read-only property
```

Интерфейсы

```
1 interface Figure {  
2     width: number;  
3     height: number;  
4 }  
5 interface Square extends Figure {  
6     square: () => number;  
7 }  
8 const sq = {width: 15, height: 20, square() {  
9     return this.width * this.height;  
10 }};  
11 sq.square();    // 300
```

Классы

```
1 abstract class Class1 {  
2     abstract func1(): void; // необходимо определить в наследниках  
3 }  
4 class Class2 extends Class1 {  
5     static readonly field3: string = 'hello';  
6     protected name: string;  
7     private field1: number;  
8     constructor() { super(); }  
9     public func1(): void { ... }  
10 }
```

Классы

```
1 interface Squarable {  
2     calcSomething(): number;  
3 }  
4  
5 class Square implements Squarable {  
6     width: number;  
7     height: number;  
8  
9     // Error: Class 'Square' incorrectly implements interface 'Squarable'.  
10    // Property 'calcSomething' is missing in type 'Square'.  
11 }
```

Generics

```
1 class Queue<T> {  
2     private data = [];  
3     push = (item: T) => this.data.push(item);  
4     pop = (): T => this.data.shift();  
5 }  
6  
7 const queue = new Queue<number>();  
8 queue.push(0);    // OK  
9 queue.push('1'); // Error: cannot push a string
```

Generics

```
1 function makeKeyValue<K, V>(key: K, value: V): { key: K; value: V } {  
2     return {key, value};  
3 }  
4  
5 const pair = makeKeyValue('days', ['ПН', 'БТ']);  
6 pair.value.push('СР', 'ЧТ', 'ПТ', 'СБ', 'ВС'); // OK  
7 pair.value.push(42);    // Error: cannot push a number  
8
```

Декораторы

```
1 class Utils {  
2     @memoize  
3     static fibonacci (n: number): number {  
4         return n < 2 ? 1 : Utils.fibonacci(n - 1) + Utils.fibonacci(n - 2)  
5     }  
6 }  
7 console.time('count');  
8 console.log(Utils.fibonacci(50));  
9 console.timeEnd('count');    // оооочень долго
```

Декораторы

```
1 function memoize (target, key, descriptor) {  
2     const originalMethod = descriptor.value;  
3     const cash = {};  
4     descriptor.value = function (n: number): number {  
5         return cash[n] ? cash[n] : cash[n] = originalMethod(n);  
6     }  
7 }  
8 console.log(Utils.fibonacci(1000)); // 7.0330367711422765e+208  
9 console.timeEnd('count'); // count: 5.668ms
```


Как типизировать JavaScript

TypeScript Declaration Files

TypeScript Declaration Files (.d.ts) — служат для описания интерфейсов, экспортируемых классов и методов для модулей, написанных на обычном JavaScript

*.d.ts файл

```
1 interface JQueryStatic {  
2     ajax(settings: JQueryAjaxSettings): JQueryXHR;  
3     (element: Element): JQuery;  
4     (html: string, ownerDocument?: Document): JQuery;  
5     (): JQuery;  
6 }  
7  
8 declare var $: JQueryStatic;  
9 declare module 'jquery' {  
10     export = $;  
11 }
```

Webpack + TypeScript

```
1 // webpack.config.js
2 module.exports = {
3   entry: './index.ts',
4   output: {
5     filename: 'bundle.js',
6   },
7   module: {
8     rules: [
9       {
10         test: /\.tsx?$/,
11         use: 'ts-loader',
12         exclude: /node_modules/,
13       },
14     ],
15   },
16 };
```

Технологии сборки

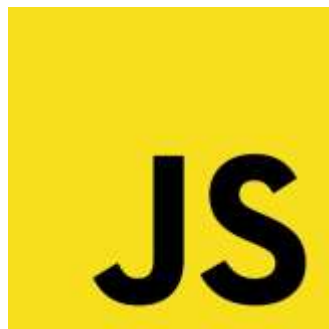
HTML



CSS



BABEL



npm

Веб реального времени

Что объединяет эти сервисы?

- сервис отслеживания общественного транспорта
- соц-сеть с чатами
- браузерная онлайн-игра

Возможность взаимодействовать с
клиентом в реальном времени

В чем проблема?

В браузере мы используем протокол HTTP, поэтому сервер может что-то сообщить клиенту, только когда сам клиент запросит какую-либо информацию.

Т.е. у сервера нет адекватной возможности оповещать клиента.

Способы решения

- Short Polling
- Long Polling
- Server-sent events
- WebSockets

Short Polling

1. Клиент запрашивает у сервера данные
2. Сервер отвечает двумя способами:
 - a. Отправляет пустой ответ
 - b. Отправляет нормальный ответ в теле
3. Как только клиент получает ответ от сервера, он ждет пару секунд, и снова начинает повторять процесс.

Long Polling

1. Клиент запрашивает у сервера данные
2. Сервер отвечает двумя способами:
 - a. Если есть новые данные, он их отдает
 - b. Если данные нет, то сервер оставляет соединение открытым на определенный период времени, и когда находят новые данные, отдает их клиенту

WebSocket

Новый (относительно) протокол полнодуплексной связи (может передавать и принимать одновременно) поверх TCP-соединения, предназначенный для обмена сообщениями между браузером и веб-сервером в режиме реального времени. С помощью его API вы можете отправить сообщение на сервер и получить ответ без выполнения отдельного HTTP-запроса, причем этот процесс будет событийно-управляемым.

Были созданы, чтобы обойти ограничение HTTP на формат запрос/ответ и дать возможность отправлять сообщения **с сервера на клиент**.

Преимущества WebSocket

- Поддерживает возможность отправки сообщений с сервера на клиент
- Все сообщения проходят через одно TCP-соединение, поэтому отсутствует overhead на открытие соединения
- Очень хорошая поддержка браузерами и очень простой API использования
- Именно поэтому WebSocket'ы очень удобно использовать для написания:
 - Реалтаймовых игр (для обмена сообщениями между клиентом и сервером)
 - Чатов и веб-мессенджеров
 - PUSH-уведомления и прочие нотификации от сервера

Создание WebSocket

```
1 const ws = new WebSocket('ws://example.com/ws');
2
3 // если страница загружена по https://
4 const ws = new WebSocket('wss://example.com/ws');
5
6 // События WebSocket
7
8 // соединение установлено
9 ws.addEventListener('open', listener);
10 // пришло новое сообщение
11 ws.addEventListener('message', listener);
12 // ошибка
13 ws.addEventListener('error', listener);
14 // сокет закрылся
15 ws.addEventListener('close', listener);
16
```


Работа с WebSocket

После создания объекта WebSocket **необходимо дождаться**, пока соединение не откроется и не установится

```
1 ws.onopen = function() {  
2     console.log('Соединение установлено, можно отправлять сообщения!');  
3  
4     // Отправка текста  
5     ws.send('Hello!');  
6     ws.send(JSON.stringify({ x: 100, y: 150 }));  
7  
8     // Отправка бинарных данных (например файлы из формы)  
9     ws.send(form.elements[0].file);  
10 };  
11
```

Что происходит в браузере



Событие error и close

```
1 ws.onerror = function(error) {  
2     // произошла ошибка в отправке/приёме данных или сетевая ошибка  
3     console.log('Ошибка ' + error.message);  
4 };  
5  
6 ws.onclose = function(event) {  
7     // 1000 - штатное закрытие сокета (коды WebSocket из 4х цифр)  
8     // 1001 - удалённая сторона исчезла  
9     // 1002 - ошибка протокола  
10    // 1003 - неверный запрос  
11    console.log('Код: ' + event.code);  
12    console.log('Причина: ' + event.reason);  
13 };  
14
```

Событие message — обработка сообщений с сервера

```
1 ws.onmessage = function(event) {  
2     const data = event.data;  
3     const message = JSON.parse(data);  
4  
5     console.log('Прислали сообщение: ' + message.text);  
6  
7     // или, если есть глобальная шина событий  
8     bus.emit(message.event, message.payload);  
9 };  
10
```

А ещё можно слать и принимать бинарные данные

```
1 var buffer = new ArrayBuffer(128);  
2 socket.send(buffer);  
3  
4 var intview = new Uint32Array(buffer);  
5 socket.send(intview);  
6  
7 var blob = new Blob([buffer]);  
8 socket.send(blob);  
9
```

Как использовать WebSocket

Договориться о своём "надпротоколе" обмена сообщениями между клиентом и сервером — зафиксировать форматы всех сообщений в приложении.

Например:

```
1 {  
2     "action": "FIRE",  
3     "payload": { "cell": "b4" }  
4 }  
5  
6 {  
7     "action": "FIRE_RESULT",  
8     "payload": { "state": "Убил" }  
9 }  
10
```

Как использовать WebSocket

Написать обёртку вокруг WebSocket, которая будет внутри себя заниматься отправкой и приёмом сообщений, а наружу будет предоставлять удобный интерфейс:

```
1 const websocketService = new WebSocketService('/ws');
2
3 websocketService.send('FIRE', { "cell": "b4" });
4 websocketService.subscribe('FIRE_RESULT', function (payload) {
5     const state = payload.state;
6     game.reRender(state);
7 });
8
```

Server Sent Events

Спецификация Server-Sent Events описывает встроенный класс EventSource, который позволяет поддерживать соединение с сервером и получать от него события.

Как и в случае с WebSocket, соединение постоянно.

Отличия от WebSocket

- Однонаправленность: данные посылает только сервер
- Только текст
- Обычный HTTP

Итого по Server Sent Events

Объект EventSource автоматически устанавливает постоянное соединение и позволяет серверу отправлять через него сообщения.

Он предоставляет:

- Автоматическое переподключение с настраиваемой retry задержкой.
- Идентификаторы сообщений для восстановления соединения. Последний полученный идентификатор посылается в заголовке Last-Event-ID при пересоединении.
- Текущее состояние, записанное в свойстве readyState.

Это делает EventSource достойной альтернативой протоколу WebSocket, который сравнительно низкоуровневый и не имеет таких встроенных возможностей (хотя их и можно реализовать).

Для многих приложений возможностей EventSource вполне достаточно.

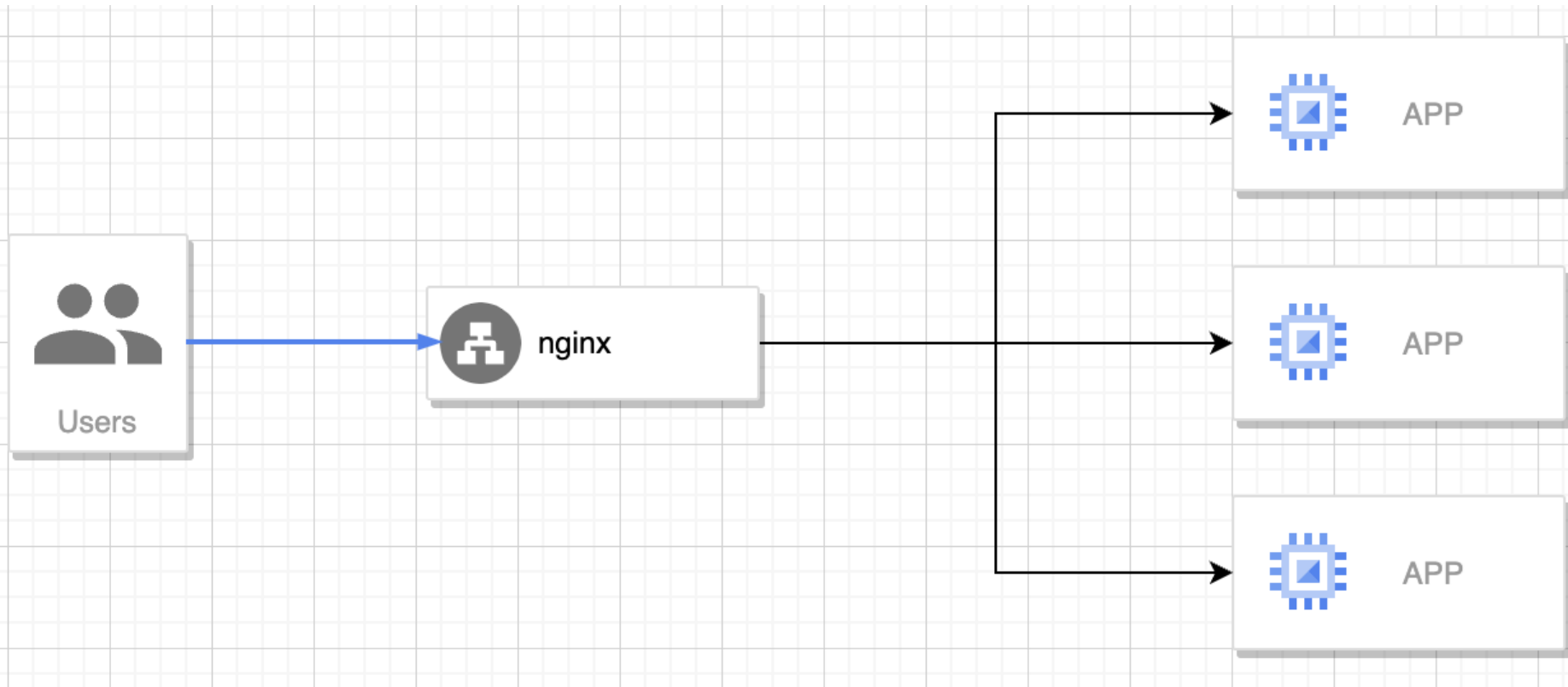
Веб-сервера

Готовый сервер по конфигу

Все вы когда-нибудь слышали слова **nginx**, **apache** и тд. Все это готовые сервера, который будут работать по конфигу, который вы им специально описываете.

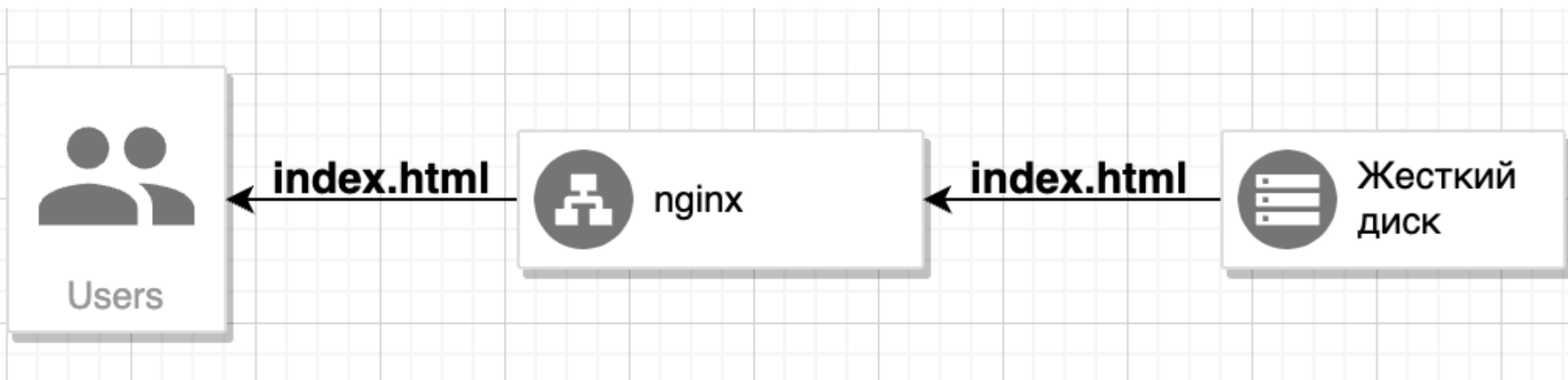
Зачем это нужно?

Например, прокси



А еще, отдавать статику

При этом, как правило, веб-сервера умеет эту статику сжимать и кэшировать, что позволяет быстрее доставлять ее до клиента.



```


1 server {
2     server_name example.ru www.example.ru;
3     access_log /var/log/nginx/example/access.log;
4     error_log /var/log/nginx/example/error.log;
5
6     location / {
7         root /home/ubuntu/example/dist;
8         try_files $uri /index.html;
9     }
10
11     location /api/v1 {
12         proxy_pass http://localhost:8088;
13         proxy_set_header Host $host;
14         proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
15     }
16
17     location /api/v1/chat/ws {
18         proxy_pass http://localhost:8080;
19         proxy_set_header Upgrade $http_upgrade;
20         proxy_set_header Connection "upgrade";
21         proxy_set_header Host $host;
22         proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
23     }
24
25     location /static {
26         root <рус хранится статика>;
27     }
28
29     listen 443 ssl http2; # managed by Certbot
30     ssl_certificate /etc/letsencrypt/live/ykoya.ru/fullchain.pem; # managed by Certbot
31     ssl_certificate_key /etc/letsencrypt/live/ykoya.ru/privkey.pem; # managed by
32     Certbot
33     ssl_dhparams /etc/letsencrypt/ssl-dhparams.pem; # managed by Certbot
34 }
35
36 server {
37     if ($host = www.ykoya.ru) {
38         return 301 https://$host$request_uri;
39     } # managed by Certbot
40
41
42     if ($host = ykoya.ru) {
43         return 301 https://$host$request_uri;
44     } # managed by Certbot
45
46
47     listen 80;
48     server_name ykoya.ru www.ykoya.ru;
49     return 404; # managed by Certbot
50 }
51

```


Прогрессивные интернетные приложения

PWA

PWA (Progressive Web Application) - технология в web-разработке, которая визуально и функционально трансформирует сайт в приложение (мобильное приложение в браузере).

 <https://s3.eu-central-1.amazonaws.com/techno>

Выберите операцию ▼

Calc

Reset

Скачиваемые файлы

Поделиться...

Найти на странице

Добавить на главный экран

Полная версия



Настройки

Справка/отзыв



https://s3.eu-central-1.amazonaws.com/technopark-cdn/sample-nativ

2



Добавить на главный экран



Calc

ОТМЕНА

ДОБАВИТЬ

Calc

Reset

13:51

MTS TELE 2 73%



Сбербанк



Музыка



ICQ



Почта Mail..



MAPS.ME



Calc



Telegram



2ГИС



Почта



Радио



Навигатор



Рокетбанк



Аэроэспр.



Мой МТС



ВКонтакте



Slack



Телефон



Сообщения



Chrome



Браузер



Камера

Что нужно сделать, чтобы завелось?

- Service Worker;
- Application shell (оболочка для быстрой загрузки с Service Worker);
- Web App Manifest;
- Push Notifications (включено в Service Worker);
- SSL-сертификат для передачи данных по протоколу HTTPS.

manifest.json

```
1 <link rel=manifest href="/manifest.json">
```

```
2
```

manifest.json

```
1 {  
2     "name": "Calculator",  
3     "short_name": "Calc",  
4     "lang": "ru-RU",  
5     "start_url": "/index.html",  
6     "display": "fullscreen", // standalone, minimal-ui, browser  
7     "orientation": "landscape", // portrait, any  
8     "background_color": "#0F0848",  
9     "theme_color": "#0F0848",  
10    "related_applications": [ ... ]  
11    "prefer_related_applications": false  
12    "icons": [ ... ]  
13 }  
14
```


Apple Meta Tags

```
1 <link rel=apple-touch-icon href="/imgs/icon-152.png">
2 <meta name=theme-color content=#0F0848>
3
4 <meta name=application-name content="Calc">
5 <meta name=apple-mobile-web-app-title content="Calc">
6
7 <meta name=mobile-web-app-capable content=yes>
8 <meta name=apple-mobile-web-app-capable content=yes>
9 <meta name=apple-mobile-web-app-status-bar-style content=#0F0848>
10
```

Преимущества PWA

PWA совмещает в себе свойства нативного приложения и функции браузера, что имеет свои преимущества:

- PWA поддерживается наиболее популярными ОС: Windows, iOS, Android. При этом загрузить можно на десктоп, смартфон, планшет, терминал в торговом зале;
- обновления добавляются разработчиками удалённо. Пользователи видят изменения и улучшения, но им не требуется скачивать эти обновления самостоятельно;
- PWA индексируется Google и другими поисковыми системами;
- благодаря сценарию Service Worker, который запускается браузером в фоновом режиме, и стратегии кэширования, обеспечивается возможность работы офлайн;
- Frontend отделён от Backend'а. Меньше времени и ресурсов тратится на разработку/переработку дизайна и логики взаимодействия PWA с клиентом;
- PWA можно установить без «Google play» и App Store, а также вопреки запрету устанавливать приложения из неизвестных источников. Лояльно относятся к PWA и антивирусные программы. Одновременно с этим передача данных происходит по протоколу HTTPS, поэтому PWA безопасно;
- с февраля 2019 года PWA можно добавлять в App Store и Google Play, давая пользователю возможность скачать приложение из привычного источника.

Недостатки PWA

Технология PWA не универсальна и имеет ряд недостатков:

- не все устройства и не все операционные системы поддерживают полный набор возможностей PWA;
- невозможно наладить активное участие пользователей iOS (например приложение может хранить локальные данные и файлы размером только до 50 Мбайт, нет доступа к In-App Payments (встроенные платежи) и многим другим сервисам Apple, нет интеграции с Siri), поддержка iOS начинается с версии 11.3;
- работа офлайн ограничена;
- работу PWA может ограничивать неполный доступ к аппаратным компонентам;
- нет достаточной гибкости в отношении «специального» контента для пользователей (например программы лояльности);
- при использовании PWA увеличивается расход заряда батареи мобильного устройства.

React (нет)

Как работает React

Virtual DOM

Главная проблема DOM — он никогда не был рассчитан для создания динамического пользовательского интерфейса.

Virtual DOM — это техника и набор библиотек/алгоритмов, которые позволяют нам улучшить производительность на клиентской стороне, избегая прямой работы с DOM путем создания и работы с абстракцией, имитирующей DOM-дерево

Virtual DOM

- Вместо того, чтобы взаимодействовать с DOM напрямую, мы работаем с его **легковесной копией**
- Мы можем вносить изменения в копию, исходя из наших потребностей, а после этого применять изменения к реальному DOM
- При этом происходит сравнение DOM-дерева с его виртуальной копией, **определяется разница и запускается перерисовка того, что было изменено**

Картиночка

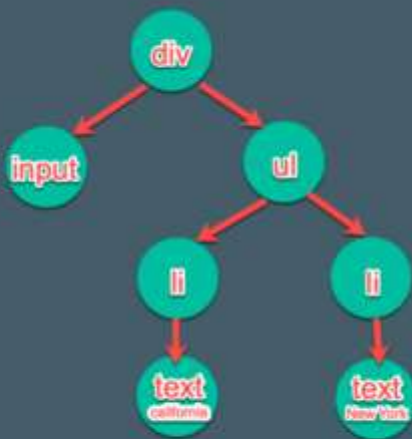
The App



Real DOM

```
<div>
  <input type="text" placeholder="Search">
  <ul>
    <li>California</li>
    <li>New York</li>
  </ul>
</div>
```

DOM Tree



VDOM

```
{
  "nodeName": "div",
  "children": [
    {
      "nodeName": "input",
      "attributes": {
        "type": "text",
        "placeholder": "Search",
        "onChange": ""
      },
      "children": []
    },
    {
      "nodeName": "List",
      "attributes": {
        "items": [
          "California",
          "New York"
        ]
      },
      "children": []
    }
  ]
}
```


Virtual DOM

Такой подход работает быстрее, потому как не включает в себя все тяжеловесные части реального DOM. Но только если мы делаем это правильно. Есть две проблемы:

- когда именно делать повторную перерисовку DOM?
- как это сделать эффективно?

Когда?

Когда данные изменяются и нуждается в обновлении. Есть два варианта узнать, что данные изменились:

- Первый из них — «dirty checking» (грязная проверка) заключается в том, чтобы опрашивать данные через регулярные промежутки времени и рекурсивно проверять все значения в структуре данных
- Второй вариант — «observable» (наблюдаемый) заключается в наблюдении за изменением состояния. Если ничего не изменилось, мы ничего не делаем. Если изменилось, мы точно знаем, что нужно обновить

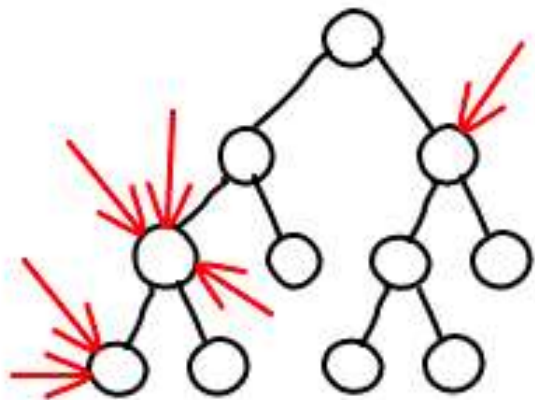
Как?

Что делает этот подход действительно быстрым:

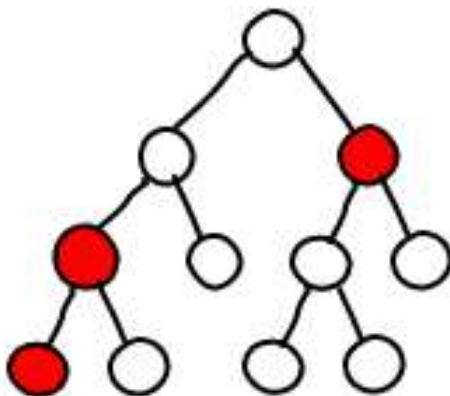
- Эффективные алгоритмы сравнения
- Группировка операций чтения/записи при работе с DOM
- Эффективное обновление под-деревьев

Наглядно

setState



Dirty



Re-rendered

