НЕТОЛОГИЯ

АСИНХРОННОСТЬ

ЕВГЕНИЙ КОРЫТОВ

ЕВГЕНИЙ КОРЫТОВ

Эксперт в разработке и управлении web-проектами

korytoff@gmail.com https://facebook.com

ПЛАН ЗАНЯТИЯ

1. Понятие асинхронности
2. Асинхронные функции

ПОНЯТИЕ АСИНХРОННОСТИ

СИНХРОННОЕ ВЫПОЛНЕНИЕ

До настоящего момента весь код, что вы писали и что мы проходили на лекциях, был синхронный. Принцип синхронного выполнения прост: код выполняется сразу. Иногда для выполнения работы коду требуется значительное время, но суть та же: программа начинает выполняться незамедлительно.

```
1  const factorial = n => {
2    for (var i = 1; n > 1; i *= n--);
3    return i;
4  }
5  // 1 * 2 * 3 * ... * 50
6  factorial(50);
```

АСИНХРОННОЕ ВЫПОЛНЕНИЕ

При разработке ПО бывают и специфические задачи:

1. Функция выполняется очень долго и хочется параллельно выполнять другой код. Например, пока ждем новых данных с сервера, можно работать со старыми.
2. Функция должна выполниться в определённое время. Например, будильник или рассылка Email-уведомлений.
3. Неизвестно точно, когда функция выполнится. Например, экран телефона необходимо включить только тогда, когда он нажмёт на кнопку «Power».

АСИНХРОННОЕ ВЫПОЛНЕНИЕ

Типичный случай из непрограммирования. В данном примере цвет кнопки изменится только при наведении на неё. Когда это может произойти и произойдёт ли вообще - зависит только от посетителя.

```
<button>Нажми на меня</button>
```

```
button:hover {
  background: red;
}
```

`setTimeout`

Для отложенного вызова функций в браузере есть функция `setTimeout(fn, ms)`, где *fn* - функция, которую нужно запустить через ms миллисекунд (1000 миллисекунд = 1 секунде).

Например, этот код через 1 секунду добавит в консоль приветствие:

```
const showGreeting = () => console.log('Поздравляем, вы стали обладателем $1000!');

setTimeout(showGreeting, 1000);
```

НЕТОЧНОСТЬ ВЫЗОВА

Тем не менее, `showGreeting` из прошлого примера выполнится не обязательно через 1 секунду.

Для наглядности, 10 раз вызовем `setTimeout` с одним и тем же параметром:

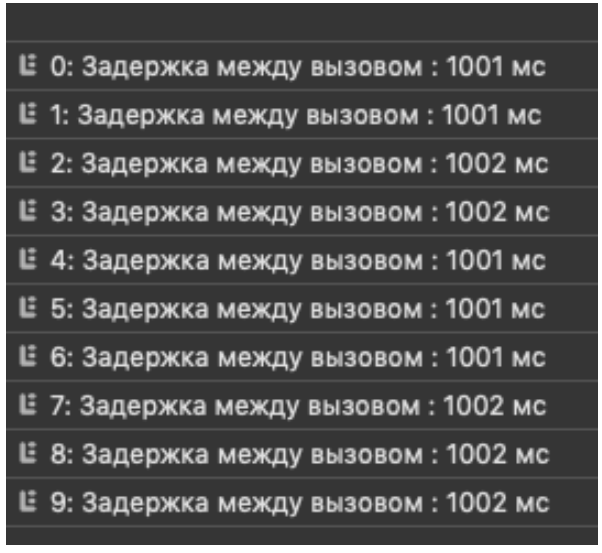
```
const checkDelay = (index, delay) => {
  const start = new Date();
  setTimeout(() => {
    const end = new Date();
    const delay = end - start;

    console.log(`${index}: Задержка между вызовом : ${delay} мс`);
  }, delay);
}

for (let i = 0; i < 10; i++) {
  checkDelay(i, 1000);
}
```

НЕТОЧНОСТЬ ВЫЗОВА

Каждый раз вывод в консоль может немного отличаться, но вот пример вызова:



ОДИН ПОТОК

Данная особенность связана с тем, что код JavaScript выполняется в одном потоке. Проще говоря, в нашем случае, `showGreeting` выполнится, только если в настоящий момент не выполняется вообще ничего.

Если же параллельно с `showGreeting` выполняется какой-либо другой код, интерпретатор JavaScript сначала закончит выполнять его и только потом возьмётся за `showGreeting`.

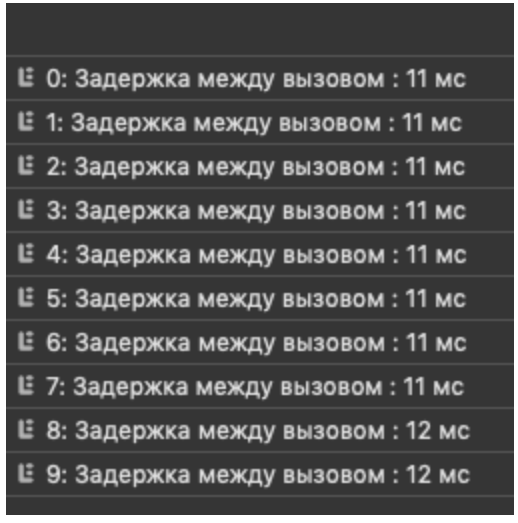
ОДИН ПОТОК

Для демонстрации нам необходимо сделать два действия: уменьшить задержку для наглядности и нагрузить интерпретатор долгими вычислениями:

```
for (let i = 0; i < 10; i++) {
  checkDelay(i, 10);
}
```

ОДИН ПОТОК

Вот «эталонный» результат, который тоже может немного отличаться от вызова к вызову:



ОДИН ПОТОК

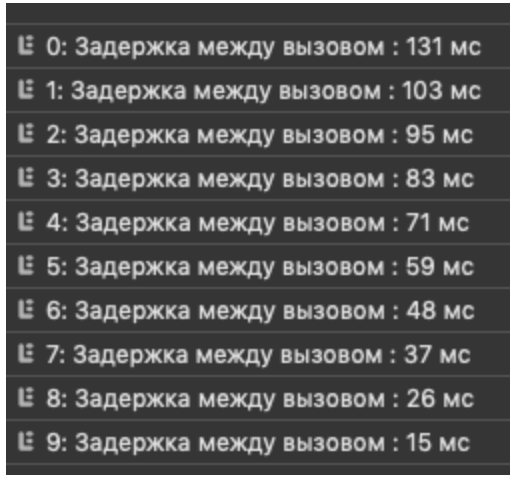
Теперь «нагрузим» JavaScript бесполезной долгой работой:

```
// делает много бесполезной работы. Главное, чтобы долго!
const idle = n => {
  let sum = 0;
  for(let i = 0; i < n; i++) {
    sum += i;
  }
}

for (let i = 0; i < 10; i++) {
  checkDelay(i, 10);
  idle(1000000);
}
```

ОДИН ПОТОК

И результат существенно изменится:



Это значит, что функция в `setTimeout` будет вызвана, только если у неё будет такая возможность. JavaScript не несёт никакой ответственности за то, что `setTimeout` сработает вовремя, но обещает это сделать как можно раньше в рамках заданной задержки.

ФУНКЦИИ ОБРАТНОГО ВЫЗОВА

Это всего лишь термин и ничего кроме. *Функцией обратного вызова* (callback) называется такая, которая:

1. Передаётся в другую функцию высшего порядка (HOF) как аргумент.
2. HOF должен вызвать эту функцию по достижению определённого условия.

Рассмотрим прошлый пример:

```
const showGreeting = () => console.log('Добрый день, я консольный бог!');

setTimeout(showGreeting, 1000);
```

В нашем случае функцией высшего порядка является `setTimeout`, а функцией обратного вызова является `showGreeting`.

МИНИМАЛЬНАЯ ЗАДЕРЖКА

Вы можете вызывать `setTimeout` и без параметра временной задержки. В таком случае функция обратного вызова сработает как можно скорее (опять же, без каких-либо обязательств).

Пример 1:

```
const showGreeting = () => console.log('Добрый день, я консольный бог!');

setTimeout(showGreeting);
```

МИНИМАЛЬНАЯ ЗАДЕРЖКА

Пример 2:

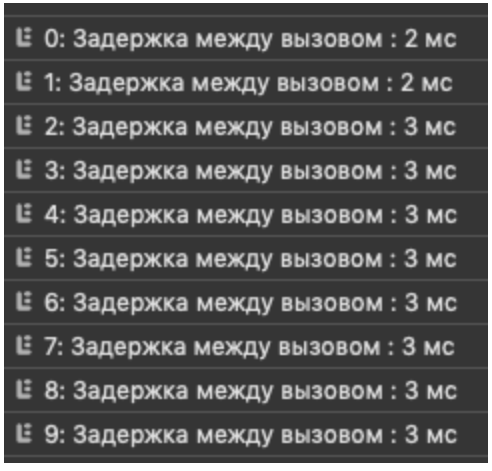
```
const checkDelay = ( index, delay ) => {
  const start = new Date;
  setTimeout(() => {
    const end = new Date();
    const delay = end - start;

    console.log(`${index}: Задержка между вызовом : ${delay} мс`);
  }, delay);
}

for (let i = 0; i < 10; i++) {
  // второй параметр в данном случае будет равен undefined
  checkDelay(i);
}
```

МИНИМАЛЬНАЯ ЗАДЕРЖКА

Пример вызова:



Вызов без второго аргумента равносилен указанию значения 0. У каждого браузера всё равно есть минимальный порог, меньше которого задержка не будет.

```
const showGreeting = () => console.log('Добрый день, я консольный бог!');

// можно было и не указывать 2 параметр
setTimeout(showGreeting, 0);
```

clearTimeout

Запланированную задачу можно отменить с помощью функции `clearTimeout`. Для этого необходимо передать идентификатор таймаута, возвращаемый от `setTimeout`.

```
// сообщение в консоль не будет выведено
const id = setTimeout(() => console.log('Я хочу жить :('), 500);

clearTimeout(id);
```

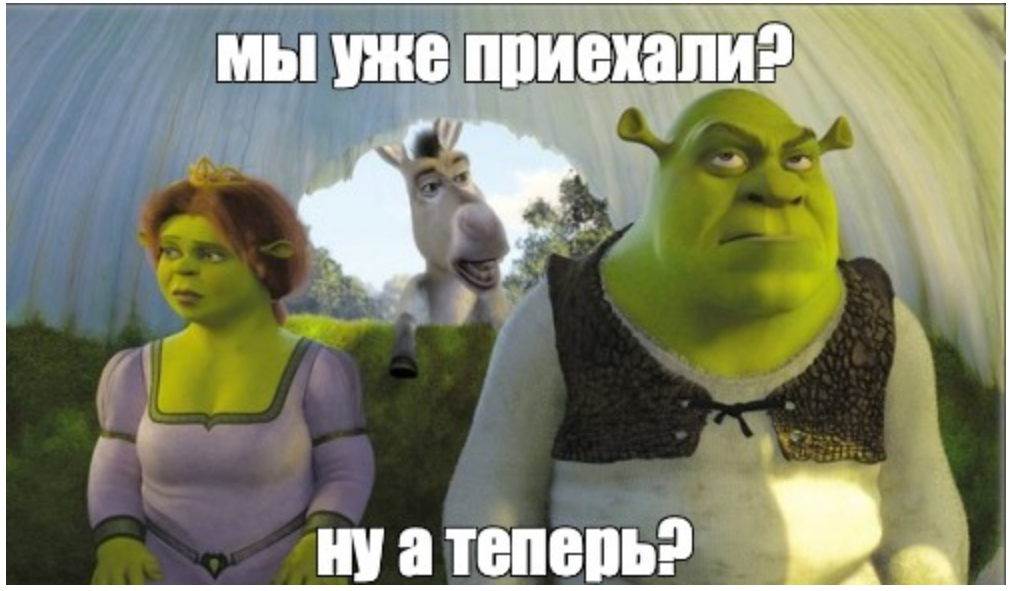
```
// сообщение в консоль не будет выведено
const sendPayment = () => console.log( 'Вам начислена зарплата!' ),
const isCrisis = true;
const id = setTimeout(sendPayment, 1000);

// не выплачиваем сотрудникам деньги. У нас кризис!
if (isCrisis) {
  clearTimeout(id);
}
```

`setInterval` (ДЕМО)

Аналогично с `setTimeout`, `setInterval` запускает функцию обратного вызова. Но делает это постоянно!

```
1 | const areWeHome = () => console.log( 'Мы уже приехали?' );
2 | // будет запускать areWeHome до посинения
3 | setInterval( areWeHome, 1000 );
```



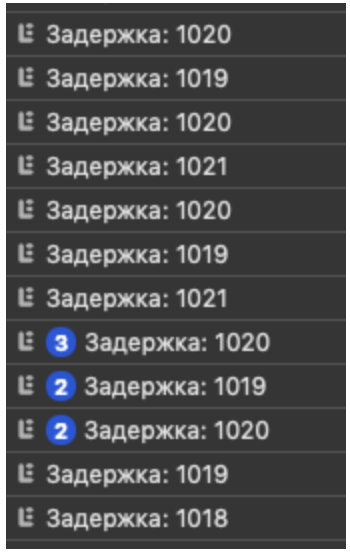
БОЛЕЗНЬ ЗАПУСКА

Как и у `setTimeout`, `setInterval` также «болеет» непредсказуемым временем запуска функций обратного вызова.

```
let start = new Date();
setInterval(() => {
  const end = new Date();
  const delay = end - start;
  console.log(`Задержка: ${delay}`);
  start = new Date;
}, 1000 );
```

БОЛЕЗНЬ ЗАПУСКА

Пример выполнения:



ПЕРЕДАЧА АРГУМЕНТОВ

Все аргументы, которые передаются в `setTimeout` и `setInterval` после второго, становятся аргументами callback-функции в момент вызова:

```
// выведет сообщение «Блиц, скорость без границ!» через 1с.
setTimeout(console.log, 1000, 'Блиц, скорость без границ!');
```

```
const sum = (a, b) => a + b;

// Выведет 29 через 1с.
setTimeout(sum, 1000, 10, 19);
```

РАБОТА С HTTP

Наиболее распространенный случай работы с асинхронным кодом — запросы по HTTP.

РАССМОТРИМ ЭТОТ КОД

```
// 1. Создаем запрос
var xhr = new XMLHttpRequest();

// 2. Опрелеяем функцию обратного вызова
xhr.onreadystatechange = processResponse;

// Этот код выполниться кода запрос будет в пути
function processResponse(e) {
  if (xhr.readyState === 4) {
    // Запрос выпонился!
    console.log(xhr.responseText);
  } else {
    // Запрос еще выполняется
    console.log('Загружаем ...')
  }
}

// 3. Определяем куда и как отправлять запрос
xhr.open('GET', 'employees.json', true);

// 4. Отправляем запрос
xhr.send();

console.log('Другая важная работа ...'
```

ЖИВОЙ ПРИМЕР

https://repl.it/repls/CourteousPoisedBrace

РЕЗУЛЬТАТА РАБОТЫ

```
Загружаем ...
Другая важная работа ...
Загружаем ...
Загружаем ...
[
  {
    "name": "Jim",
    "inOffice": false
  },
  {
    "name": "Joe",
    "inOffice": true
  },
  {
    "name": "John",
    "inOffice": true
  }
```

ОПТИМИЗАЦИЯ ВЫЧИСЛЕНИЙ

ПРОБЛЕМА ДОЛГИХ ВЫЧИСЛЕНИЙ

Второй вид функций: выполняющиеся очень долго. При больших значениях n страница будет существенно подвисать:

```
const sum = n => {
  let sum = n;
  for (let i = 0; i < n; i++){
    sum += i;
  }
  return sum;
}

sum( 100000 );
```

РЕШЕНИЕ

Интервалы и таймеры - идеальное решение для долгих или рекурсивных вычислений. С помощью них мы можем проделывать полезную работу небольшими порциями, не нагружая страницу.

```
// вычисляем сумму от 0 до n, по itemsPerStep элементов за раз
const sumStep = (n, itemsPerStep, onload) => {
  const size = Math.ceil(n / itemsPerStep); // количество шагов
  let index = 0; // текущий шаг
  let sum = 0; // сумма вычислений

  // эта функция будет вызываться каждые 500 мс
  return () => {
    // окончание вычислений
    if (index === size) {
      onload(sum);
      return;
    }

    // начальные и конечные значения шага
    const start = index * itemsPerStep;
    const end = Math.min((index + 1) * itemsPerStep, n + 1);

    // сами вычисления
    for (let i = start; i < end; i++){
      sum += i;
    }

    console.log(`Шаг ${index}: ${sum}`);
    index++;
    // планируем новый шаг
    scheduleStep();
  };
};


// функция обратного вызова для вывода результата
const onload = result => console.log(`Результат вычислений: ${result}`);

// задаем начальные настройки
const step = sumStep( 1000000, 1000, onload );
const scheduleStep = () => setTimeout( step, 500);

scheduleStep();
```

ТАЙМАУТЫ В ИНТЕРФЕЙСАХ

Использование таймаутов и интервалов в JavaScript - довольно распространенное явление. На веб-сайтах вы наверняка встречались со следующими примерами:

1. Появление всплывающих окон в заданное время.
2. Таймеры обратного отсчёта на странице («До конца акции осталось...»).
3. Показ подсказок при поиске только по окончанию ввода в текстовое поле.
4. Простая анимация.
5. Автоматическая смена слайдов в фотогалерее.
6. Проверка в простых чатах новых сообщений.

ЧЕМУ МЫ НАУЧИЛИСЬ?

- Разобрались с основами таймеров и интервалов.
- Научились работать с датами.
- Узнали о «Болезни запуска» `setTimeout` и `setInterval`.
- Научились оптимизировать долгие и рекурсивные вычисления.
- Узнали разницу между синхронной и асинхронной работой.
- Разобрались как отправлять HTTP-запросы и обрабатывать результаты.

ДОМАШНЕЕ ЗАДАНИЕ

Давайте посмотрим ваше домашнее задание.

- Вопросы по домашней работе задаем в чате Slack!
- Задачи можно сдавать по частям.
- Зачет по домашней работе проставляется после того, как приняты все **все задачи**.

НЕТОЛОГИЯ

Спасибо за внимание! Время задавать вопросы

ЕВГЕНИЙ КОРЫТОВ


korytoff@gmail.com


https://facebook.com