

Веб- программирование: Основы JavaScript

Содержание

01 Что это такое

02 Базовая база

03 Функции и объекты

04 Массивы

05 Асинхронность

06 Работа с DOM

07 DevTools

08 Ресурсы для изучения

Что такое JavaScript

Определение

JavaScript (сокращенно JS) — это мультипарадигменный язык программирования высокого уровня, который является реализацией спецификации ECMAScript.

Ключевая веб-технология.

Основные характеристики

- Динамическая типизация
- Мультипарадигменность
- Асинхронность (но без параллелизма)
- Интерпретируемость (но не совсем)

Где применяется

- Фронтенд
- Бекенд (через Node.js и другие среды)
- Мобильные устройства (React Native)
- Десктопные приложения (Electron)
- VR/XR
- Embedded
- Космические корабли

**Всё, что может быть написано на JavaScript -
будет написано на JavaScript**

Преимущества

- Универсальность
- Большая экосистема
- Активное сообщество
- Низкий порог входа
- Живые стандарты
- Обратная совместимость

Недостатки

- Обратная совместимость
- Неоднозначные дизайн-решения
- Производительность
- Кросбраузерность
- Сложность поддержки

База

Типы

- Number
- String
- Boolean
- Null
- Undefined
- Symbol
- BigInt
- Object
- Array
- Function

Особенности

- Динамическая типизация — тип определяется автоматически
- Иммутабельность — примитивные типы (первые 6) неизменяемы
- У всех примитивных типов есть классы-обёртки
- Ссылки — сложные типы (Array и Object) передаются по ссылке

Типы: Number

- Хранит целые и дробные числа
- Формат хранения — 64-битное число с плавающей точкой

```
let num = 42;  
let pi = 3.14;
```

Типы: String

- Текстовые данные в одинарных, двойных или обратных кавычках
- Строки неизменяемы

```
let name = "Аня";
```

Интерполяция строк

Интерполяция строк — это способ встраивания выражений (переменных, констант, выражений) непосредственно внутри строковых литералов. В JavaScript это реализуется с помощью шаблонных строк (template strings).

Альтернатива конкатенации: “a” + “b”

```
const a = 5;
const b = 10;
const result = `Сумма: ${a + b}, произведение: ${a * b}`;
```

Типы: Boolean

- Имеет два значения: true (истина) и false (ложь)
- Используется в условных операторах

```
let isReady = true;
```

Типы: Null

- Используется для явного указания отсутствия значения

```
let empty = null;
```

Типы: Undefined

- Присваивается переменной при объявлении без значения

```
let value; // undefined
```

Типы: **Symbol**

- Создает уникальные значения
- Полезен для создания уникальных свойств объектов

```
let unique = Symbol();
```

Типы: BigInt

- Для работы с большими целыми числами
- Создается с помощью суффикса n или конструктора BigInt()

```
let big = 12345678901234567890n;
```

Типы: Object

- Хранит коллекции данных в формате ключ-значение

```
let me = {  
  name: "Аня",  
  age: 31,  
  isStudent: false  
};
```

Типы: Array

- Упорядоченная коллекция значений
- Доступ по индексу
- Могут содержать элементы разных типов

```
let numbers = [1, 2, 'три', {key: 'value'}];
```

Типы: Function

- Особый тип данных в JavaScript, представляющий функции как объекты первого класса
- Функции могут храниться в переменных
- Функции могут передаваться как аргументы
- Функции могут возвращаться из других функций

```
const sum = new Function('a', 'b', 'return a + b;');

console.log(sum(2, 3)); // 5
```

== VS ===

- == (абстрактное равенство) — выполняет приведение типов перед сравнением
- === (строгое равенство) — сравнивает значения без приведения типов

```
5 == '5'      // true
5 === '5'     // false

true == 1     // true
true === 1    // false

null == undefined // true
null === undefined // false

0 == ''        // true
0 === ''       // false
```

Определение типа: typeof

- Возвращает строку с названием типа
- `typeof null` возвращает “object” (известная особенность)
- `typeof NaN` возвращает “number”
- Для всех объектов (включая массивы и функции) возвращает “object”

```
typeof 42; // "number"
typeof "text"; // "string"
typeof true; // "boolean"
typeof undefined; //
"undefined"
```

Переменные

Переменная — это именованный контейнер для хранения данных в памяти компьютера.

Объявление переменных

- let — для переменных, значения которых могут меняться
- const — для констант (значения не меняются)
- var — устаревший способ (используется только для поддержки старых браузеров)

```
let name = 'Аня'; // переменная с начальным значением
const PI = 3.14; // константа
let age; // объявление без начального значения
var a = 'a'; // устаревшее
```

Множественное объявление

Можно объявить несколько переменных за один раз:

```
let firstName = 'Аня',
    lastName = 'Мотошкина',
    age = 31;
```

Правила именования

- Имя может содержать:
 - Буквы латинского алфавита
 - Цифры (но не в начале)
 - Символы \$ и _
- Нельзя начинать с цифры
- Нельзя использовать зарезервированные слова (let, const, var, class и т.д.)

Соглашения по именованию

- camelCase — для обычных переменных
- PascalCase — для классов
- SCREAMING_SNAKE_CASE — для констант

```
let userName = 'Аня';
class User {}
const LECTION_DURATION_IN_MIN = 90;
```

Особенности `let`

- Можно изменять значение
- Нельзя повторно объявить (в той же области видимости)

Особенности `const`

- Значение нельзя изменить
- При объявлении обязательно начальное значение
- Для объектов и массивов можно менять содержимое

Операторы

Операторы — это специальные символы или ключевые слова, которые выполняют определенные действия над значениями (операндами). Они являются основой для выполнения любых операций в JavaScript.

Арифметические операторы

- `+` — сложение
- `-` — вычитание
- `*` — умножение
- `/` — деление
- `%` — остаток от деления
- `**` — возведение в степень

Операторы сравнения

- Равенства:
 - `==` — равенство с приведением типов
 - `===` — строгое равенство
 - `!=` — неравенство с приведением типов
 - `!==` — строгое неравенство
- Отношения:
 - `>` — больше
 - `<` — меньше
 - `>=` — больше или равно
 - `<=` — меньше или равно

Логические операторы

- `&&` — логическое И
- `||` — логическое ИЛИ
- `!` — логическое НЕ

Операторы присваивания

- Базовое присваивание: `=` — простое присваивание
- Составные операторы: `+=`, `-=`, `*=`, `/=`, `%=`, `**=`

```
let x = 5;  
x += 3; // равносильно x = x + 3  
console.log(x); // 8
```

Унарные операции

- + — преобразование в число
- - — унарный минус

```
console.log(+ '123'); // 123  
console.log(- '123'); // -123
```

Операторы инкремента и декремента

- `++` — увеличение на 1
- `--` — уменьшение на 1

```
let count = 5;
console.log(count++); // 5
console.log(count); // 6
```

```
let num = 5;
console.log(--num); // 4
```

Условия

- if...else – базовая конструкция
- switch – для множественных условий
- тернарный оператор – краткая форма if...else

Оператор if...else

```
if (условие) {  
    // код, если условие истинно  
} else if (другое условие) {  
    // код, если другое условие истинно  
} else {  
    // код, если все условия ложны  
}
```

Оператор switch

```
switch (выражение) {  
    case значение1:  
        // код  
        break;  
    case значение2:  
        // код  
        break;  
    default:  
        // код по умолчанию  
}
```

Тернарный оператор

- Ухудшает читабельность кода

```
условие ? выражение1 : выражение2
```

```
let result = age >= 18 ? 'Взрослый' : 'Несовершеннолетний';
```

Truthy и falsy значения

Truthy — значения, которые при преобразовании в boolean дают true.
Falsy — значения, которые при преобразовании в boolean дают false.

К Falsy относятся:

- `false` — логическое ложное значение
- 0 (ноль любого типа: число, строка)
- "" или "" — пустая строка
- `null` — отсутствие значения
- `undefined` — необъявленная переменная
- `NaN` — не число

К Truthy относится всё остальное (в том числе любые сложные типы, в том числе) пустой массив и пустой объект

Циклы

- `for` — цикл с заранее заданными условиями
- `while` — цикл с предусловием
- `do...while` — цикл с постусловием
- `for...in` — для перебора свойств объекта
- `for...of` — для перебора значений итерируемых объектов

Цикл for

```
for (начальное выражение; условие; шаг) {  
    // тело цикла  
}  
  
for (let i = 0; i < 5; i++) {  
    console.log(i); // выведет числа от 0 до 4  
}
```

Цикл while

```
while (условие) {  
    // тело цикла  
}  
  
let count = 0;  
while (count < 5) {  
    console.log(count);  
    count++;  
}
```

Цикл do...while

- Особенность: выполняется минимум один раз

```
do {  
    // тело цикла  
} while (условие);  
  
let num = 0;  
do {  
    console.log(num);  
    num++;  
} while (num < 5);
```

Функции и объекты

Функции

Функция — это блок кода, который выполняет определённое действие и может быть вызван многократно. Функции помогают структурировать код и избегать его дублирования.

Объявление функций: декларативное объявление

```
function greet(name) {  
    return `Привет, ${name}!`;  
}
```

Объявление функций: функциональное выражение

```
const greet = function(name) {  
    return `Привет, ${name}!`;  
};
```

Объявление функций: стрелочная функция

Особенности:

- Не имеют собственного this
- Не могут использоваться как конструкторы

```
const greet = name => `Привет, ${name}!`;
```

Область видимости переменных

Область видимости — это часть кода, в пределах которой доступна определённая переменная.

В JavaScript существует три типа областей видимости:

- Глобальная: переменные, объявленные вне функций
- Функциональная (локальная): переменные, объявленные внутри функций
- Блочная: создаётся с помощью фигурных скобок {}

Внутренние области видимости могут обращаться к переменным внешних областей.

Объекты

Объект — это набор свойств, где каждое свойство состоит из ключа (имени) и значения. Объекты используются для хранения коллекций данных и более сложных структур.

Создание объектов

- Литеральный синтаксис

```
const person = {  
    name: 'Аня',  
    age: 31,  
    isStudent: false  
};
```

- Конструктор Object

```
const person = new Object();  
person.name = 'Иван';  
person.age = 25;
```

Свойства объектов

Свойства могут быть:

- Примитивными значениями
- Функциями (методами)
- Другими объектами

Способы доступа:

- Через точку
- Через квадратные скобки

```
console.log(person.name); // Аня  
console.log(person['name']); // Аня
```

Можно добавлять, изменять и удалять свойства в любой момент.

Особенности объектов

Объекты передаются по ссылке, а не по значению

```
const cat = {  
    name: 'Барсик',  
    color: 'orange',  
};  
  
const cat2 = cat;  
cat2.name = 'Мурзик';  
  
console.log(cat.name) // "Мурзик"
```

Методы работы с объектами

- `Object.keys()` — возвращает массив ключей объекта
- `Object.values()` — возвращает массив значений
- `Object.entries()` — возвращает массив пар ключ-значение
- `Object.assign()` — копирует свойства
- `delete` оператор — удаляет свойство
- `JSON.stringify()` — преобразует объект в JSON
- `JSON.parse()` — преобразует JSON в объект

Spread оператор

Spread-оператор (три точки ...) — это синтаксический сахар в JavaScript, который позволяет расширять массивы и объекты.

Позволяет:

- Объединять
- Клонировать

Spread оператор: примеры

```
const obj1 = { a: 1, b: 2 };
const obj2 = { b: 3, c: 4 };
const merged = { ...obj1, ...obj2 };
// { a: 1, b: 3, c: 4 }

const original = { name: 'Аня', age: 31 };
const copy = { ...original };
```

Spread оператор: особенности

- Создает поверхностную копию
- Не работает с циклическими ссылками
- Не копирует методы (свойства-функции)
- Не копирует свойства из прототипа
- Работает и с массивами и с объектами

Массивы

Массивы

Массив — это упорядоченная коллекция элементов, которая позволяет хранить несколько значений в одной переменной.

Имеет свойство `length`, которое возвращает размер

Массивы могут содержать элементы разных типов

Создание массивов

- Литеральный синтаксис

```
const arr = [1, 2, 3, 4, 5];
```

- Конструктор Array

```
const arr = new Array(1, 2, 3);
const emptyArr = new Array(5);
// создает массив длиной 5 с undefined
```

Методы массивов

- `push()` — добавляет в конец
- `pop()` — удаляет последний
- `unshift()` — добавляет в начало
- `shift()` — удаляет первый
- `splice()` — изменяет содержимое
- `slice()` — создает копию части массива
- `forEach()` — выполняет функцию для каждого элемента
- `map()` — создает новый массив с результатами вызова функции
- `filter()` — создает массив из элементов, прошедших проверку
- `indexOf()` — возвращает индекс элемента
- `includes()` — проверяет наличие элемента
- `find()` — находит первый подходящий элемент
- `findIndex()` — возвращает индекс найденного элемента
- `reduce()` — сводит массив к одному значению

Методы массивов

- `join()` — объединяет элементы в строку
- `sort()` — сортирует массив

Методы массивов: forEach

```
const arr = ['cat', 'dog', 'parrot'];
arr.forEach((item, index) => {
    item = item + index
});

// ['cat0', 'dog1', 'parrot2']
```

Методы массивов: map

```
const arr = ['cat', 'dog', 'parrot'];
const arr2 = arr.map((item, index) =>
{
    return item + index;
});

// arr: ['cat', 'dog', 'parrot']
// arr2: ['cat0', 'dog1', 'parrot2']
```

Методы массивов: reduce

```
const arr = [1, 2, 3];
const sum = arr.reduce((acc, curr) => acc + curr, 0);
```

Асинхронность

Асинхронность

Асинхронное программирование — это подход к написанию кода, при котором выполнение некоторых операций не блокирует основной поток выполнения программы.

- Предотвращение блокировки интерфейса
- Оптимизация производительности
- Параллельное выполнение операций
- Работа с внешними сервисами

Callback-функции

Callback — это функция, которая передается в качестве аргумента другой функции и выполняется после завершения операции.

```
function fetchData(callback) {  
    setTimeout(() => {  
        callback('Данные получены');  
    }, 1000);  
  
    fetchData(data => {  
        console.log(data);  
    });  
}
```

Promise

Promise — это объект, представляющий завершение или неудачу асинхронной операции.

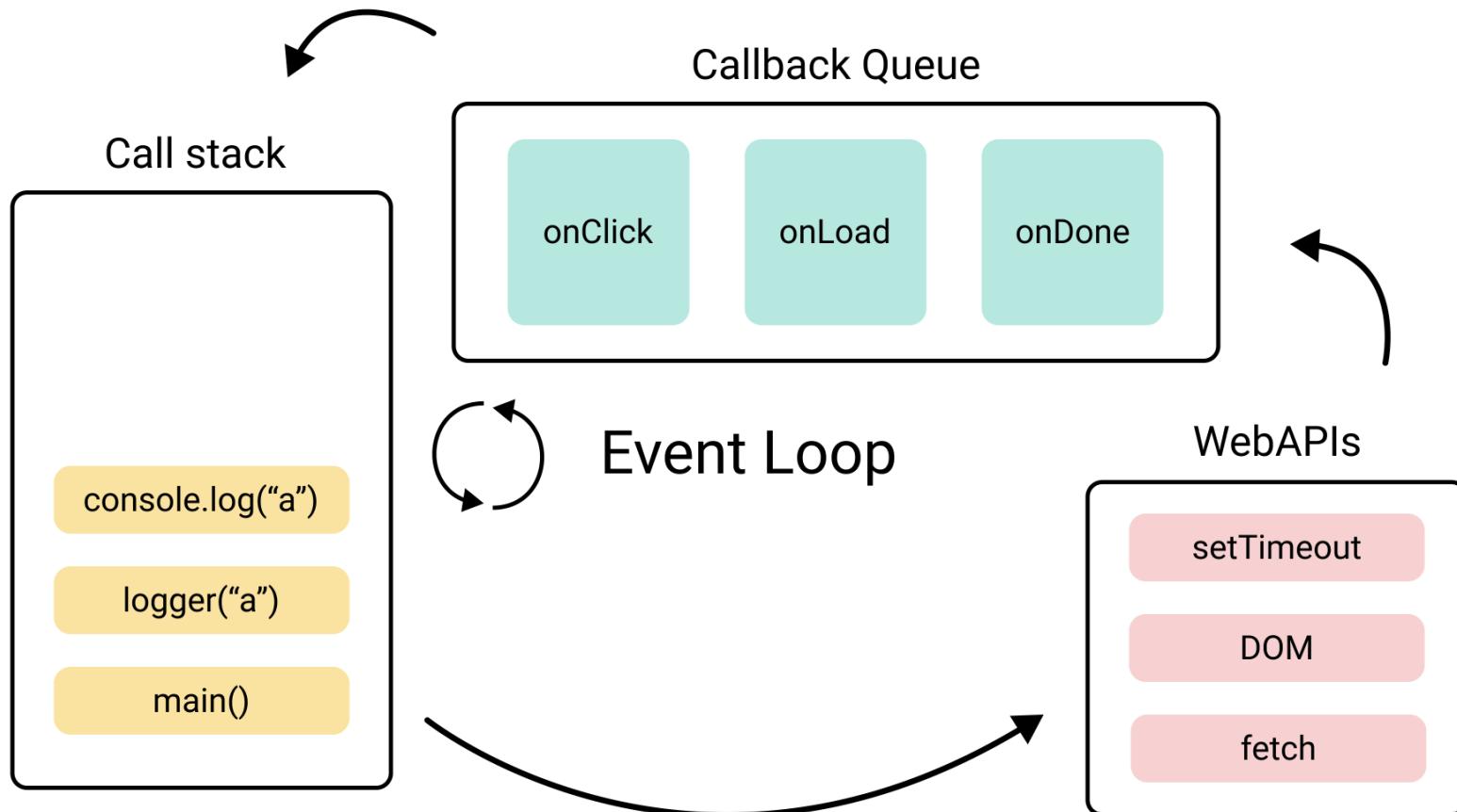
```
function fetchData() {  
    return new Promise(resolve => {  
        setTimeout(() => {  
            resolve('Данные получены');  
        }, 1000);  
    })  
}  
  
fetchData().then(data => {  
    console.log(data);  
});
```

Promise: методы

- `then()` — обработка успешного выполнения
- `catch()` — обработка ошибок
- `finally()` — выполнение после завершения
- `Promise.all()` — параллельное выполнение
- `Promise.race()` — выполнение первого завершенного

Event Loop

Event Loop (цикл событий) — это механизм, который позволяет JavaScript выполнять асинхронный код, не блокируя основной поток выполнения.



Работа с DOM

DOM

DOM (Document Object Model) — это программный интерфейс, который представляет HTML-документ в виде иерархической структуры объектов.

- Node — базовый элемент DOM-дерева
- Element — HTML-элемент
- Document — корневой объект
- Attributes — атрибуты элементов

Получение DOM-элементов

- По айди: `document.getElementById('id');`
- По тегу: `document.getElementsByTagName('div');`
- По классу: `document.getElementsByClassName('class-name');`
- По селектору (один элемент): `document.querySelector('.class-name');`
- По селектору (много): `document.querySelectorAll('.class-name');`

Создание и удаление DOM-элементов

- Создание: `document.createElement('div');`
- Добавление в DOM-дерево: `document.body.appendChild(newElement);`
- Удаление: `element.remove();`

Работа с событиями

```
const handler = function() {
    console.log('Клик!');
};

// добавление обработчика
element.addEventListener('click',
    handler);

// Удаление обработчика
element.removeEventListener('click',
    handler);
```

Какие события бывают



https://developer.mozilla.org/en-US/docs/Web/API/Document_Object_Model/Events

DevTools

DevTools

Тут демо :)

Ресурсы для изучения

Ресурсы

- Серия книг You Don't Know JS:
<https://github.com/getify/You-Dont-Know-JS>
- Учебник: <https://javascript.info/>
- Человеческие спеки: <https://developer.mozilla.org/en-US/>
- Человеческие спеки на русском: <https://doka.guide/js/>
- Форум: <https://stackoverflow.com/questions>
- Курсы (не РФ, платно): <https://frontendmasters.com/>
- Курсы (РФ, платно): <https://htmlacademy.ru/>
- Курсы (РФ, платно): <https://practicum.yandex.ru/>
- ... и ещё миллион всего...

Вопросы