

Объекты в JavaScript

Урок 5





Кадочников Алексей

Frontend-разработчик

- ☀ Веб-разработчик со стажем более 9 лет
- ☀ Преподаватель GeekBrains с 2015 года
- ☀ Автор курсов по Frontend на портале Geekbrains
- ☀ Работал в таких компаниях, как VK и Wizard-C



План курса

1

Знакомство с JavaScript

2

Основы JavaScript

3

Функции в JavaScript

4

Циклы и массивы

5

Работа с объектами
в JavaScript

6

Введение в DOM

7

Работа с DOM

8

Основы событий в
JavaScript

9

Работа с событиями в
JavaScript

10

Основы шаблонизации,
работа с JSON

11







Работа с медиа файлами

12

Основы API, итоги курса



Что будет на уроке сегодня

-  Объекты в JavaScript
-  Преобразование объекта в массив
-  Object.keys
-  Object.values
-  Object.entries
-  Работа с объектами и функции высшего порядка



Объекты в JavaScript

В JS объекты занимают ключевую позицию в построении сложного кода. Это может быть управление моделью HTML-документа, организация хранения данных, упаковка библиотек JavaScript. Тема объектов пронизывает язык практически насквозь.

Главный секрет объектов JavaScript в том, что они **представляют собой коллекцию свойств.**





Преимущества использования объектов

Мы начинаем мыслить не выдуманными сущностями переменных и функций, а переходим на реальный и более высокий уровень представления. Ведь в повседневной жизни мы как раз реализуем концепцию объектов.

Объекты позволяют скрыть сложную структуру данных, дают возможность работать на высоком уровне кода и не тратить время на технические нюансы.



Свойства объектов

Таким образом, у нас есть объект с упакованными внутри него свойствами.

Чтобы начать работать со свойством, нужно указать имя объекта, поставить точку и указать имя свойства. Пример синтаксиса с **точечной записью**:

```
1 const car = {  
2   make: "Audi",  
3   model: "A5",  
4   year: 2023,  
5   color: "red",  
6   passengers: 2,  
7   power: 249  
8 };  
9  
10 console.log(car.model);  
11 car.power = 350;
```

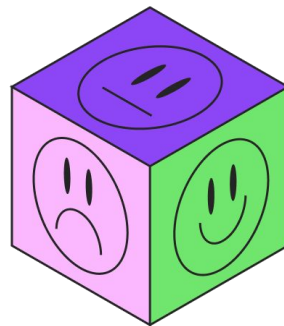


Ключевое отличие примитивных типов от объектов

Ключевое различие между примитивами и объектами — способ их хранения в памяти компьютера.

Примитивные значения сохраняются в переменной напрямую. Присваивая одну примитивную переменную другой, мы копируем значение этой переменной в новую переменную, и они остаются независимыми.

Объекты могут быть очень большими (например, крупный массив). Хранить их значения в каждой переменной не рационально, поэтому используется принцип хранения объектов.





Хранение объектов

При процедурном подходе аргументы передаются функциям по значению. То есть мы создаём копию переменной и что-то с ней делаем, но исходная переменная не меняется.

Те же правила работают для объектов, но ведут себя иначе. Поскольку в переменной хранится ссылка на объект, а не сам объект, при передаче по значению в параметр передаётся копия ссылки, которая указывает на исходный объект.

Таким образом, **если изменится свойство объекта в функции, изменится свойство исходного объекта** (в отличие от примитивов). Все изменения, внесённые в объект внутри функции, будут действовать и после завершения функции.

```
1 const car = {  
2   make: "Audi",  
3   model: "A5",  
4   year: 2023,  
5   color: "red",  
6   passengers: 2,  
7   power: 249  
8 };  
9  
10 function haveRoadTrip(myCar, distance) {  
11   myCar.odometer += distance;  
12 }  
13 haveRoadTrip(car, 150);  
14 console.log(car.odometer);
```



Методы объектов

Кроме свойств, которые описывают состояние объекта, в JS есть методы, определяющие поведение объектов.

Объекты активны и могут выполнять различные операции. Автомобиль не стоит на месте – он передвигается, включает фары и так далее. Объект car тоже должен уметь совершать эти действия.

Метод можно добавить прямо в объект:

```
1 const car = {  
2   make: "Audi",  
3   model: "A5",  
4   year: 2023,  
5   color: "red",  
6   passengers: 2,  
7   power: 249,  
8   odometer: 0,  
9   startEngine: function() {  
10     console.log("Engine started");  
11   }  
12 };  
13
```



Методы объектов

Усложним объект и добавим к нему функцию `haveRoadTrip`. Она изменится — теперь нам не надо передавать в неё объект автомобиля.

Если рассуждать логически, машина никуда не поедет с заглушенным двигателем. **Поэтому нам понадобятся:**

- Булевое свойство для хранения состояния двигателя (запущен или нет).
- Условная проверка в методе `haveRoadTrip`. Она убедится, что двигатель запущен, прежде чем вы сможете вести машину.



Методы объектов

```
1 const car = {
2   make: "Audi",
3   model: "A5",
4   year: 2023,
5   color: "red",
6   passengers: 2,
7   power: 249,
8   odometer: 0,
9   engineIsStarted: false,
10  startEngine: function() {
11    this.engineIsStarted = true;
12  },
13  stopEngine: function() {
14    this.engineIsStarted = false;
15  },
16  haveRoadTrip: function(distance) {
17    if (this.engineIsStarted) {
18      this.odometer += distance;
19    } else {
20      alert("Сначала запустите двигатель!");
21    }
22  }
23 };
24
```



Перебор значений

```
1 for (const key in object) {  
2   console.log(key + ": " + object[key]);  
3 }
```

Цикл `for in` перебирает свойства объекта, которые последовательно присваиваются переменной `key`.

Мы применили альтернативный способ обращения к свойствам объекта: **запись с квадратными скобками**. Она эквивалентна точечной записи, делает то же самое, но у неё чуть больше гибкости.



Перебор значений. Пример

```
1 const car = {
2   make: "Audi",
3   model: "A5",
4   year: 2023,
5   color: "red",
6   passengers: 2,
7   power: 249,
8   odometer: 0,
9 };
10
11 console.log('Все ключи объекта car');
12 for (const key in car) {
13   console.log(key);
14 }
15 console.log('Все значения объекта car');
16 for (const key in car) {
17   console.log(car[key]);
18 }
```



Преобразование объекта в массив

Часто мы получаем объект, но нам нужно работать с его данными как с массивом. Начинающие программисты на JavaScript пытаются применить метод `map` массива к объекту:

```
1 const object = {  
2   1: 'Ivanov',  
3   2: 'Petrov',  
4 };  
5  
6 const students = object.map((student) => `student: ${student}`); //  
  VM223:6 Uncaught TypeError: object.map is not a function at <  
  anonymous >: 6: 25  
7
```



Преобразование объекта в массив

Код возвращает ошибку, так как у объектов нет метода `map()`.

Мы должны преобразовать объект в массив и уже потом с ним работать.

Давайте рассмотрим методы, которые позволяют это сделать.





Object.keys

К примеру, группа студентов заведена в программе как объект, где ключи — их фамилии, а значения — более подробная информация. Мы хотим получить массив фамилий этих студентов. Для этого используем **Object.keys(<исходный объект>)**.

Object.keys позволяет получить из объекта все ключи первого уровня и положить их в массив.



Object.keys

```
1 const group1 = {
2   "Ivanov": {
3     practicePlace: "ldu-1",
4     practiceTime: 56
5   },
6   "Petrov": {
7     practicePlace: "kamaz",
8     practiceTime: 120
9   },
10  "Sidorov": {
11    practicePlace: "ldu-1",
12    practiceTime: 148
13  },
14  "Belkin": {
15    practicePlace: "GosDZU",
16    practiceTime: 20
17  },
18  "Avdeev": {
19    practicePlace: "LPK",
20    practiceTime: 160
21  }
22 }
23
24 const group1Students = Object.keys(group1);
25 console.log(group1Students); // ["Ivanov", "Petrov", "Sidorov",
26                               "Belkin", "Avdeev"]
27
```



Object.values

Но чаще мы имеем дело со значениями в объектах. Например, нам нужно вывести данные студентов в виде таблицы.

С помощью метода **Object.values(<исходный объект>)** можем получить значения объекта в виде массива.

```
1 const car = {  
2   make: "Audi",  
3   model: "A5",  
4   year: 2023,  
5   color: "red",  
6   passengers: 2,  
7   power: 249,  
8   odometer: 0,  
9 };  
10  
11 console.log(Object.values(car)); // ['Audi', 'A5', 2023, 'red', 2,  
12   249, 0]
```



Object.entries

Порой нужны и ключи объекта, и их значения. Например, если мы хотим вывести таблицу с фамилиями и данными о студентах. Здесь поможет метод **Object.entries(<исходный объект>)**.

Object.entries позволяет получить из объекта ключи и значения в виде массива массивов. То есть в результате его работы мы получаем массив, содержащий массивы из двух элементов: ключ и его значение.

Давайте улучшим таблицу отработанной практики, добавим туда колонку с фамилией студента.



Object.entries

```
1 const car = {
2   make: "Audi",
3   model: "A5",
4   year: 2023,
5   color: "red",
6   passengers: 2,
7   power: 249,
8   odometer: 0,
9 };
10
11 console.log(Object.entries(car));
12 // 0: (2) ['make', 'Audi']
13 // 1: (2) ['model', 'A5']
14 // 2: (2) ['year', 2023]
15 // 3: (2) ['color', 'red']
16 // 4: (2) ['passengers', 2]
17 // 5: (2) ['power', 249]
18 // 6: (2) ['odometer', 0]
19
```



Глобальный объект Window

В любой среде исполнения JavaScript всегда есть глобальный объект:

- объект Window в браузере;
- Global в Node.js;
- WorkerGlobalScope в воркере.

```
> window
< Window {0: Window, 1: Window, 2: Window, 3: Window, 4: Window, 5: Window,
  6: Window, 7: Window, 8: Global, 9: Window, 10: Window, window: Window, sel
f: Window, document: document, name: "", Location: Location, ...}

> window.console
< console {debug: f, error: f, info: f, log: f, warn: f, ...}
  ▶ assert: f assert()
  ▶ clear: f clear()
  ▶ context: f context()
  ▶ count: f count()
  ▶ countReset: f countReset()
  ▶ debug: f debug()
  ▶ dir: f dir()
  ▶ dirxml: f dirxml()
  ▶ error: f error()
  ▶ group: f group()
  ▶ groupCollapsed: f groupCollapsed()
  ▶ groupEnd: f groupEnd()
  ▶ info: f info()
  ▶ log: f log()
  ▶ memory: (...)
  ▶ profile: f profile()
  ▶ profileEnd: f profileEnd()
  ▶ table: f table()
  ▶ time: f time()
  ▶ timeEnd: f timeEnd()
  ▶ timeLog: f timeLog()
  ▶ timeStamp: f timeStamp()
  ▶ trace: f trace()
  ▶ warn: f warn()
  Symbol(Symbol.toStringTag): "Object"
  ▶ get memory: f ()
  ▶ set memory: f ()
  ▶ __proto__: Object
```



Работа с объектами
и функции высшего порядка





Работа с объектами и функции высшего порядка

- map
- filter
- reduce
- some
- find





map

map — один из самых используемых методов при работе с массивом. Он позволяет проитерировать весь массив и создать на его основе новый.

map с английского — карта. Метод позволяет сделать «карту соответствия» исходного массива и нового. Принимает аргумент-функцию, в которую при работе передаются аргументы:

- текущий элемент массива,
- его индекс,
- полный массив.

Функция-аргумент должна сделать необходимые вычисления и вернуть новые элементы, из которых будет построен новый массив.



map. Пример

```
1 const studentsPracticeTime = [
2   {
3     firstName: "Ivanov",
4     practiceTime: 56
5   },
6   {
7     firstName: "Petrov",
8     practiceTime: 120
9   },
10  {
11    firstName: "Sidorov",
12    practiceTime: 148
13  },
14  {
15    firstName: "Belkin",
16    practiceTime: 20
17  },
18  {
19    firstName: "Avdeev",
20    practiceTime: 160
21  }
22 ];
23
24 // Мы хотим вывести в таблицу студентов и информацию для каждого,
    прошел ли он практику уже. Практика будет считаться пройденной, если
    студент отработал 120 часов или больше.
25
26 const dataForTable = studentsPracticeTime.map((student) => {
27   if (student.practiceTime < 120) {
28     return { // Мы возвращаем новый объект, более удобный для
        вывода.
29       Student: student.firstName,
30       Practice: "Not passed"
31     };
32   } else {
33     return {
34       Student: student.firstName,
35       Practice: "Passed"
36     };
37   }
38 });
39
40 console.table(dataForTable); // В консоль можно выводить разными
    способами, если использовать метод table, и передать туда массив или
    объект, этот метод выводит данные в виде таблицы.
```

(index)	Student	Practice
0	Ivanov	Not passed
1	Petrov	Passed
2	Sidorov	Passed
3	Belkin	Not passed
4	Avdeev	Passed



filter

Метод `filter` используется для фильтрации элементов массива по правилу, которое задаёте вы сами. Позволяет исключить из исходного массива лишние элементы и получить новый массив.

Как и `map`, метод `filter` получает в качестве аргумента функцию обратного вызова с теми же аргументами. Только эта функция возвращает `false`, если элемент нужно исключить, и `true`, чтобы он попал в новый массив.



filter. Пример

Возьмём предыдущий пример и отфильтруем массив студентов, проходящих практику. Оставим только тех, кто практику уже прошёл.

```
1 const studentsPracticeTime = [{
2   firstName: "Ivanov",
3   practiceTime: 56
4 },
5 {
6   firstName: "Petrov",
7   practiceTime: 120
8 },
9 {
10  firstName: "Sidorov",
11  practiceTime: 148
12 },
13 {
14   firstName: "Belkin",
15   practiceTime: 20
16 },
17 {
18   firstName: "Avdeev",
19   practiceTime: 160
20 }
21 ];
22
23 // Мы хотим отфильтровать массив студентов, оставив в новом массиве
24 // только тех, кто уже прошел практику. Практика будет считаться
25 // пройденной, если студент отработал 120 часов или больше.
26
27 const studentsPassedPractice = studentsPracticeTime.filter((student)
28   => {
29     if (student.practiceTime < 120) return false
30     return true
31   });
32
33 console.log(studentsPassedPractice); // Получили новый массив, в
34 // котором только те студенты, кто уже прошел практику.
35
36 [ {
37   // "firstName": "Petrov",
38   // "practiceTime": 120
39 }, {
40   // "firstName": "Sidorov",
41   // "practiceTime": 148
42 }, {
43   // "firstName": "Avdeev",
44   // "practiceTime": 160
45 } ]
```



reduce

Метод `reduce` не так просто понять с первого раза. Его называют свёрткой, так как он проходится по всему массиву и позволяет собрать и обработать его значения в новую форму.

например, в одно значение (допустим сумму значений всех элементов).

Чтобы лучше понять этот метод, посмотрим на алгоритм без `reduce`, а потом оптимизируем его с помощью `reduce`.



Без reduce

```
1 const studentsPracticeTime = [{
2   firstName: "Ivanov",
3   practiceTime: 56
4 },
5 {
6   firstName: "Petrov",
7   practiceTime: 120
8 },
9 {
10    firstName: "Sidorov",
11    practiceTime: 148
12 },
13 {
14    firstName: "Belkin",
15    practiceTime: 20
16 },
17 {
18    firstName: "Avdeev",
19    practiceTime: 160
20 }
21 ];
22
23 // Посчитаем сколько всего часов практики отработали студенты.
24 let totalTime = 0; // Объявим переменную для хранения суммы всех
    часов.
25
26 for (let index = 0; index < studentsPracticeTime.length; index++) {
27   totalTime = totalTime + studentsPracticeTime[index].practiceTime;
28 }
29
30 console.log(totalTime); // 504
```



reduce

А теперь применим reduce для этой же цели.

Метод принимает два аргумента.

1. Функция обратного вызова с четырьмя аргументами (обычно используют два, реже три):

- аккумулятор,
- текущий элемент массива,
- индекс этого элемента,
- весь массив.

Функция обратного вызова должна вернуть обновлённое значение аккумулятора, которое будет передано в следующую итерацию.

2. Первоначальное значение аккумулятора.

C reduce



```
1 const studentsPracticeTime = [{
2   firstName: "Ivanov",
3   practiceTime: 56
4 },
5 {
6   firstName: "Petrov",
7   practiceTime: 120
8 },
9 {
10  firstName: "Sidorov",
11  practiceTime: 148
12 },
13 {
14  firstName: "Belkin",
15  practiceTime: 20
16 },
17 {
18  firstName: "Avdeev",
19  practiceTime: 160
20 }
21 ];
22
23 // Посчитаем сколько всего часов практики отработали студенты.
24 const totalTime = studentsPracticeTime.reduce((acc, student) => { //
25   // Первое значение - это функция обратного вызова, которая будет
26   // получать значение аккумулятора (acc) при каждой итерации; и текущий
27   // элемент массива (student).
28   return acc + student.practiceTime;
29 }, 0); // Второй аргумент - это первоначальное значение аккумулятора
30 // - 0.
31
32 console.log(totalTime); // 504
33
34 const studentsPracticeTime = [
35 {
36   firstName: "Ivanov",
37   practiceTime: 56
38 },
39 {
40   firstName: "Petrov",
41   practiceTime: 120
42 },
43 {
44   firstName: "Sidorov",
45   practiceTime: 148
46 },
47 {
48   firstName: "Belkin",
49   practiceTime: 20
50 },
51 {
52   firstName: "Avdeev",
53   practiceTime: 160
54 }
55 ];
56
57 // Посчитаем сколько всего часов практики отработали студенты.
58 const totalTime = studentsPracticeTime.reduce((acc, student) => { //
59   // Первое значение - это функция обратного вызова, которая будет
60   // получать значение аккумулятора (acc) при каждой итерации; и текущий
61   // элемент массива (student).
62   return acc + student.practiceTime;
63 }, 0); // Второй аргумент - это первоначальное значение аккумулятора
64 // - 0.
65
66 console.log(totalTime); // 504
```




some

Метод `some` используется, когда нужно проверить, что в массиве есть хоть один подходящий нам элемент. Например, есть ли среди всех студентов хоть кто-то, кто прошел практику.

Принимает функцию обратного вызова, которая вызывается для каждого элемента массива с аргументами:

- текущий элемент массива,
- его индекс,
- весь исходный массив;

Функция должна проверить условие и вернуть `true`, если элемент подходит, и `false`, если не подходит. Когда будет найден первый подходящий элемент, выполнение метода `some` прекратится, он вернёт `true`. Иначе метод пройдёт по всем элементам массива и вернёт `false`. Если исходный массив пустой, метод сразу вернёт `false`.



some

Метод удобен для проверки больших массивов, так как останавливается сразу после нахождения подходящего элемента, не тратит ресурсы зря.

```
1 const studentsPracticeTime = [
2   {
3     firstName: "Ivanov",
4     practiceTime: 56
5   },
6   {
7     firstName: "Petrov",
8     practiceTime: 120
9   },
10  {
11    firstName: "Sidorov",
12    practiceTime: 148
13  },
14  {
15    firstName: "Belkin",
16    practiceTime: 20
17  },
18  {
19    firstName: "Avdeev",
20    practiceTime: 160
21  }
22 ];
23 // Мы хотим найти студента Belkin и посмотреть сколько времени он
   отработал на практике.
24 const studentBelkin = studentsPracticeTime.find((student) => {
25   return student.firstName === "Belkin";
26 });
27 console.log(studentBelkin.practiceTime); // 20
28
29
```



Деструктуризация





Деструктуризация

Порой нам часто приходится обращаться к определённым элементам массива или ключам объекта в нашем алгоритме. Чтобы каждый раз не писать имя массива/объекта с путём до нужного элемента, мы можем сохранить его данные в отдельную переменную.

Новый стандарт ES2015 позволил легко получать данные из массивов и объектов, сохраняя их в новые переменные, деструктуризируя массив или объект.



Деструктуризация

Как мы могли скопировать данные из массива или объекта в отдельные переменные раньше:

```
1 // Сбор данных из объекта.
2 const student = {
3   firstName: "Ivan",
4   lastName: "Petrov",
5   age: 21,
6 };
7
8 const firstName = student.firstName; // мы объявляем отдельно
   переменную, под каждый нужный нам параметр.
9 const lastName = student.lastName;
10 const age = student.age;
11
12 // Сбор данных из массива.
13 const students = ["Ivanov", "Petrov", "Belkin"];
14 const student1 = students[0];
15 const student2 = students[1];
16 const student3 = students[2];
17
18 Не очень удобно. Деструктуризация дала нам гораздо более простой и
   удобный синтаксис:
19 // Сбор данных из объекта.
20 const student = {
21   firstName: "Ivan",
22   lastName: "Petrov",
23   age: 21,
24 };
25
26 const { firstName, lastName, age } = student; // Деструктуризация -
   мы объявляем все переменные в фигурных скобках, название должно
   совпадать с нужным нам параметром.
27
28 // Сбор данных из массива.
29 const students = ["Ivanov", "Petrov", "Belkin"];
30 const [student1, student2, student3] = students; // Деструктуризация
   - Тут мы указываем имена переменных в квадратных скобках, и в них по
   порядку будут записаны элементы массива.
31
32
```



Деструктуризация

Теперь мы можем объявлять все нужные переменные в одном месте и получать данные из объектов или массивов одной строкой.

Деструктуризация также позволяет задавать значения по умолчанию для переменных на случай, если в объекте не окажется ключа, который мы запрашиваем, или в элементе массива будет значение `undefined`.







Деструктуризация


При деструктуризации объекта можно переименовать переменные, в которые будут сохраняться ключи, если мы, например, не хотим или не можем использовать переменную с таким же именем как и ключ:

```
1 // Сбор данных из объекта.
2 const student = {
3   firstName: "Ivan",
4   lastName: "Petrov",
5   age: 21,
6 };
7
8 // Деструктурируем значение ключа firstName в переменную
   studentName. И зададим для возраста значение по умолчанию, равное
   20.
9 const { firstName: studentName, lastName, age = 20 } = student;
10
11 // Сбор данных из массива.
12 const students = ["Ivanov", "Petrov", "Belkin"];
13 const [student1, student2, student3] = students; // Тут мы указываем
   имена переменных в квадратных скобках, и в них по порядку будут
   записаны элементы массива.
14
15
```



Итоги урока

-  Рассмотрели объекты в JavaScript
-  Разобрали преобразование объекта в массив
-  Узнали о методах `Object.keys` `Object.values` `Object.entries`
-  Рассмотрели работу с объектами и функции высшего порядка

Спасибо 
за внимание

