

JS: Array Methods

НАШИ ПРАВИЛА



Включенная камера



Вопросы по поднятой руке



Не перебиваем друг друга



Все вопросы, не связанные с тематикой курса (орг-вопросы и т. д.), должны быть направлены куратору



Подготовьте свое рабочее окружение для возможной демонстрации экрана (закройте лишние соцсети и прочие приложения)

Повторим;)

■ Как получить доступ к значению свойства объекта?

■ Как удалить свойство из объекта?

■ Когда требуется применение
брекет-синтаксиса?

ЦЕЛЬ

Изучить методы массивов

ПЛАН ЗАНЯТИЯ

- Получение массива значений/ключей объекта
- Reference type
- Деструктуризация
- Методы массивов

Получение значений/ключей

Получили массив ключей и массив значений

```
const person = {  
  name: 'John',  
  age: 25,  
  city: 'Example City'  
};  
  
const keys = Object.keys(person); // ['name',  
  'age', 'city']  
const values = Object.values(person); // ['John',  
  25, 'Example City']
```

Reference type

```
let person = {  
  firstName: 'John',  
  lastName: 'Doe'  
};
```



Одно из фундаментальных отличий объектов от примитивов заключается в том, что объекты хранятся и копируются «по ссылке», тогда как примитивные значения: строки, числа, логические значения и т.д. – всегда копируются «как целое значение».

```
1 let message = "Привет!";  
2 let phrase = message;
```

В результате мы имеем две независимые переменные, каждая из которых хранит строку "Привет!".

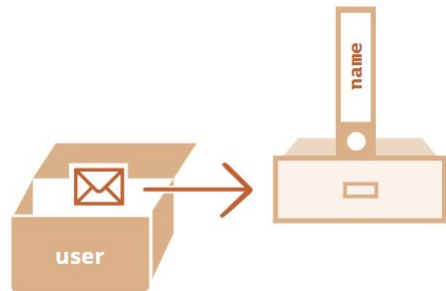


Объекты ведут себя иначе, чем примитивные типы.

Переменная, которой присвоен объект, хранит не сам объект, а его «адрес в памяти» – другими словами, «ссылку» на него.

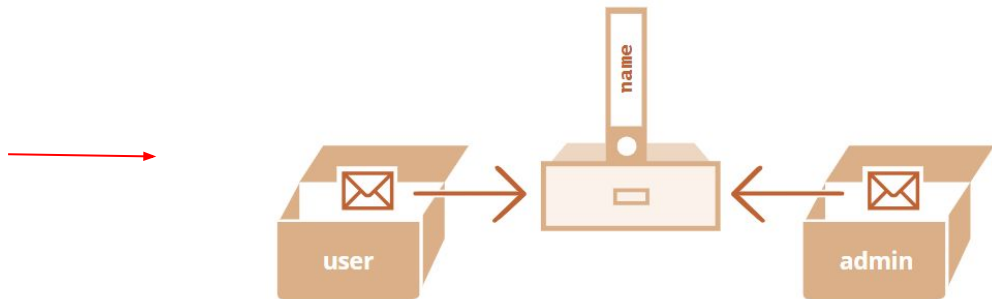
```
1 let user = {  
2   name: "John"  
3 };
```

Вот как это на самом деле хранится в памяти:

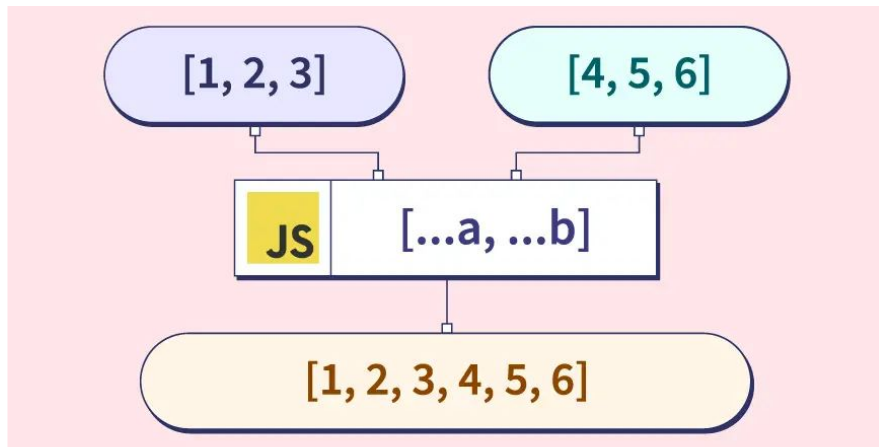


При копировании переменной объекта копируется ссылка, но сам объект не дублируется.

```
1 let user = { name: "John" };  
2  
3 let admin = user; // копируется ссылка
```

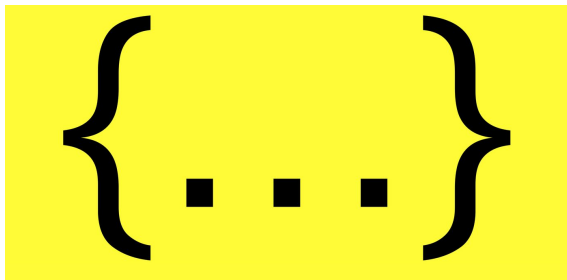


Деструктуризация



Оператор расширения - spread

В JavaScript оператор расширения (spread operator) представляет собой синтаксическую конструкцию, которая используется для создания копий массивов, объединения массивов, передачи аргументов функции и других подобных задач.



1. Для массивов:

В приведенном примере `...arr1` разворачивает элементы массива `arr1` в новом массиве `arr2`.

```
const arr1 = [1, 2, 3];  
const arr2 = [...arr1, 4, 5, 6];  
console.log(arr2); // [1, 2, 3, 4, 5, 6]
```

2. Для объектов:

Здесь spread оператор используется для создания нового объекта obj2, который содержит все свойства из obj1, а также новые свойства.

```
const obj1 = { key1: 'value1', key2: 'value2' };  
const obj2 = { ...obj1, key3: 'value3', key4: 'value4' };  
console.log(obj2);  
// { key1: 'value1', key2: 'value2', key3: 'value3', key4: 'value4' }
```

3. Передача аргументов функции

В данном примере spread оператор используется для передачи элементов массива в качестве аргументов функции `sum`.

```
function sum(a, b, c) {  
  return a + b + c;  
}  
  
const numbers = [1, 2, 3];  
console.log(sum(...numbers)); // 6
```

Деструктурирующее присваивание



[{...}]

Деструктурирующее присваивание – это специальный синтаксис, который позволяет нам «распаковать» массивы или объекты в несколько переменных, так как иногда они более удобны.



Деструктуризация массива

```
1 // у нас есть массив с именем и фамилией
2 let arr = ["Ilya", "Kantor"];
3
4 // деструктурирующее присваивание
5 // записывает firstName = arr[0]
6 // и surname = arr[1]
7 let [firstName, surname] = arr;
8
9 alert(firstName); // Ilya
10 alert(surname);  // Kantor
```

Деструктурирующее присваивание ничего не уничтожает, его задача – только скопировать нужные значения в переменные.

Деструктуризация объекта

Деструктурирующее присваивание также работает с объектами.

```
1 let options = {  
2   title: "Menu",  
3   width: 100,  
4   height: 200  
5 };  
6  
7 let {title, width, height} = options;  
8  
9 alert(title); // Menu  
10 alert(width); // 100  
11 alert(height); // 200
```

Для явного задания имени переменной используем синтаксис:
название ключа: имя переменной

```
let {title: optionName, width, height} = options;
```

Map

Метод `map()` создает новый массив с результатами вызова предоставленной функции для каждого элемента в массиве.

```
const numbers = [1, 2, 3, 4, 5];  
const squaredNumbers = numbers.map(num => num ** 2);  
// squaredNumbers: [1, 4, 9, 16, 25]
```

ForEach

Метод `forEach()` выполняет предоставленную функцию один раз для каждого элемента массива.

Возвращаемое значение:
undefined

```
const numbers = [1, 2, 3, 4, 5];  
numbers.forEach(num => console.log(num));  
  
// Output: 1, 2, 3, 4, 5
```


Filter

Метод `filter()` создает новый массив с элементами, для которых предоставленная функция возвращает `true`

```
const students = [
  { name: "Alice", grade: 85 },
  { name: "Bob", grade: 90 },
  { name: "Charlie", grade: 78 },
];

const highGrades = students.filter((student) => student.grade
> 80);

// highGrades: [{ name: 'Alice', grade: 85 }, { name: 'Bob',
grade: 90 }]
```



Ваша новая IT-профессия – Ваш новый уровень жизни

Программирование с нуля в
немецкой школе AIT TR GmbH