






Frontender[1.0] JavaScript - Прототипы, прототипное наследование. proto, prototype, new Function

 YouTube	https://youtu.be/GjcJRYDaxUM
 Telegram	https://t.me/Dmitry_Kolotilshikov
 Github	https://github.com/DmitryKolotilshikov/
 Boosty	https://boosty.to/dmitry_ko
# Номер урока	41



Полезные ссылки:

<https://learn.javascript.ru/prototypes>



Прототипы в **JavaScript** — это механизм, с помощью которого объекты в **JavaScript** могут наследовать свойства и методы друг от друга. Каждый объект имеет внутреннюю ссылку на другой объект, называемый прототипом.

Мы часто хотим взять что-то и расширить.

Например, у нас есть объект `user` со своими свойствами и методами, и мы хотим создать объекты `admin` и `guest` как его слегка изменённые варианты. Мы хотели бы повторно использовать то, что есть у объекта `user`, не копировать/переопределять его методы, а просто создать новый объект на его основе.

Прототипное наследование — это возможность языка, которая помогает в этом.

В JavaScript объекты имеют специальное скрытое свойство

`[[Prototype]]` (так оно названо в спецификации), которое либо равно `null`, либо ссылается на другой объект. Этот объект называется «прототип»:

Прототип даёт нам немного «магии». Когда мы хотим прочитать свойство из

объекта `object`, а оно отсутствует, JavaScript автоматически берёт его из прототипа. В программировании такой механизм называется «**прототипным наследованием**»



- **Прототип объекта (`__proto__`):**

- Это скрытое свойство, которое указывает на прототип другого объекта, от которого он наследует.
- Если свойство или метод не найдено у самого объекта, поиск продолжается в его прототипе.

- **`prototype` у функций-конструкторов:**

- Каждая функция-конструктор имеет объект `prototype`, который используется для создания прототипа объектов, создаваемых с помощью этого конструктора.

- **Наследование по цепочке прототипов:**

- Если прототип объекта тоже имеет прототип, поиск продолжается по цепочке, пока не будет найдено свойство или достигнут конец цепочки (обычно `Object.prototype`).



Рекомендации

- Не изменяйте напрямую `__proto__`, используйте `Object.create()` или функции-конструкторы.
- Современный JavaScript также предлагает классы (`class`), которые используют прототипы под капотом, но упрощают синтаксис.
- Лишний раз лучше избежать использование прототипов

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <script src="script.js"></script>
  <title>Prototypes</title>
</head>
<body>
</body>
</html>
```

```
// prototypes - прототипы

const animal = {
  eats: true
};

const rabbit = {
  jumps: true
};

rabbit.__proto__ = animal;

// console.log(rabbit);

// -----
const animal1 = {
  eat() {
    console.log("Я ем");
  },
};
```

```

// Создаем объект с прототипом animal
const dog = Object.create(animal1);

dog.eat(); // "Я ем" (наследуется от animal)

dog.bark = function () {
  console.log("Гав!");
};

dog.bark(); // "Гав!"

// -----
// Функция - конструктор

function Person(name) {
  this.name = name;
}

// Добавляем метод через prototype
Person.prototype.sayHello = function () {
  console.log(`Привет, меня зовут ${this.name}`);
};

// Создаем объекты
const john = new Person("Джон");
const jane = new Person("Джейн");

john.sayHello(); // "Привет, меня зовут Джон"
jane.sayHello(); // "Привет, меня зовут Джейн"

console.log(john instanceof Person); // true
console.log(john.__proto__ === Person.prototype); // true

// -----
// можно добавить собственные методы к встроенным объектам

Array.prototype.printElements = function () { // добавляем новые метод ко всем массивам
  this.forEach((element, index) => { // this указывает на массив, на котором метод в
    console.log(`Элемент ${index + 1}: ${element}`);
  });
};

const myArray = [10, 20, 30, 40, 50];
myArray.printElements();

```