

ПМЗ Разработка модулей ПО.

РО 3.1 Понимать и применять принципы объектно-ориентированного и асинхронного программирования.

Тема 1. Введение в ООП.

Лекция 4. Прототипы и прототипное наследование.

Цель занятия:

Познакомиться с прототипами в JavaScript и понять механизм наследования через цепочку прототипов.

Учебные вопросы:

1. Основы прототипов.

2. Цепочка прототипов. Механизм поиска свойств.

1. Основы прототипов.

Прототип в JavaScript — это механизм, который позволяет объектам наследовать свойства и методы друг от друга.

Каждый объект в JavaScript имеет внутреннюю ссылку на другой объект, который называется его прототипом

Когда вы пытаетесь получить доступ к свойству объекта, JavaScript сначала ищет это свойство в самом объекте, а затем — в его прототипе, и так далее, пока не найдет его или не дойдет до конца цепочки ссылок (до **null**).

Свойство `prototype` у функций-конструкторов.

Функции-конструкторы в JavaScript — это функции, используемые для создания объектов.

Когда вы создаете объект с помощью функции-конструктора (с использованием ключевого слова `new`), JavaScript устанавливает свойство `prototype` этой функции в качестве прототипа создаваемого объекта.

Пример:

```
function Person(name) {  
  | |   this.name = name;  
  |  
}  
  
console.log(typeof Person.prototype) // object  
  
// Добавляем метод greet в прототип Person  
Person.prototype.greet = function() {  
  | |   console.log(`Hello, my name is ${this.name}`);  
  |  
};  
  
const alice = new Person('Alice');  
alice.greet(); // "Hello, my name is Alice"
```

Как это работает:

- Создание объекта: Когда вы создаете объект `alice` с помощью `new Person('Alice')`, JavaScript создает новый объект и устанавливает его внутреннее свойство `[[Prototype]]` на `Person.prototype`.
- Наследование: Когда вы вызываете `alice.greet()`, JavaScript ищет метод `greet` сначала в объекте `alice`, а затем в `Person.prototype`, где он и находит его.

Зачем это нужно?

Использование прототипов позволяет экономить память, так как все экземпляры одного и того же типа могут использовать одни и те же методы, определенные в прототипе, вместо того чтобы иметь свои собственные копии этих методов.

Таким образом, прототипы в JavaScript являются ключевым элементом наследования и организации кода.

Различия между `prototype`, `proto` и `[[Prototype]]`

В JavaScript три термина — `prototype`, `__proto__` и `[[Prototype]]` — относятся к механизму прототипного наследования, но они имеют разные значения и назначения.

1. prototype

Это свойство функции-конструктора.

Используется для определения свойств и методов, которые будут доступны всем объектам, созданным с помощью этой функции-конструктора.

Например, если у вас есть функция-конструктор `Person`, вы можете добавить методы в `Person.prototype`, и все экземпляры `Person` будут иметь доступ к этим методам.

Пример:

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

```
Person.prototype.greet = function() {  
  |   console.log(`Hello, my name is ${this.name}`);  
};
```

2. `__proto__`

Это **свойство объектов** (доступное в большинстве современных браузеров).

Оно ссылается на внутреннее свойство `[[Prototype]]` объекта.

Позволяет вам получить доступ к прототипу объекта, но не рекомендуется использовать его напрямую, так как это не стандартное свойство, а скорее «наследие» старых версий JavaScript.

Пример:

```
const alice = new Person('Alice');  
console.log(alice.__proto__ === Person.prototype); // true
```

3. [[Prototype]]

Это внутреннее свойство, которое не доступно напрямую и не может быть изменено.

Каждый объект в JavaScript имеет это свойство, и оно указывает на прототип объекта.

```
> alice
```

```
◀ ◂ Person {name: 'Alice'} ⓘ
```

```
  name: "Alice"
```

```
  ▸ [[Prototype]]: Object
```

```
>
```

Основные отличия:

`prototype`: Свойство функции-конструктора, используемое для определения методов и свойств для объектов, созданных с помощью этой функции.

`__proto__`: Свойство экземпляра объекта, ссылающееся на его прототип; не является стандартом, но широко используется.

`[[Prototype]]`: Внутреннее свойство, которое хранит ссылку на прототип объекта, не доступное напрямую.

2. Цепочка прототипов. Механизм поиска свойств.

Как работает цепочка прототипов?

Объект: Когда вы создаете объект, он имеет ссылку на свой прототип, который может быть другим объектом или null.

Поиск свойства: Когда вы обращаетесь к свойству объекта:

- JavaScript сначала ищет это свойство в самом объекте.
- Если оно не найдено, JavaScript переходит к прототипу объекта (через свойство `[[Prototype]]` или `__proto__`).
- Этот процесс продолжается, пока не будет найдено свойство или не будет достигнут null.

Пример:

```
Object.prototype.sayHello = function(){console.log("Hello!")}  
  
const animal = {  
  makeSound: function(){  
    console.log("Sound!")  
  }  
}  
  
const dog = Object.create(animal);  
dog.sayHello();
```

> dog

◀ ◻ {} ⓘ

◻ [[Prototype]]: Object

▸ makeSound: f ()

◻ [[Prototype]]: Object

▸ sayHello: f ()

Механизм поиска свойства.

- Запрос свойства: Вы вызываете `dog.sayHello()`.
- Поиск в объекте: JavaScript ищет метод `sayHello` в объекте `dog`. Он его не находит.
- Поиск в прототипе: JavaScript переходит к `dog.prototype (animal)` и не находит его.
- Поиск в прототипе прототипа: JavaScript переходит к `animal.prototype (Object)` и находит его там.
- Выполнение метода: Метод вызывается.

Домашнее задание:

1. <https://ru.hexlet.io/courses/js-introduction-to-oop>

7 Прототипы

Знакомимся с механизмом прототипов и учимся правильно создавать абстракции данных в JavaScript

2. Повторить материал лекции.

Материалы лекций:

<https://github.com/ShViktor72/Education2025>

Обратная связь:

colledge20education23@gmail.com