

JavaScript: Prototypes

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)

ПОИГРАЕМ ;)

■ Что возвращает метод `reduce`

■ Для чего используется метод `reverse`?

■ Что делает метод `join`?

■ Какие методы массивов вы знаете?

■ Какие методы массивов изменяют исходный массив?

ЦЕЛЬ

Изучить прототипную модель наследования объектов в JavaScript

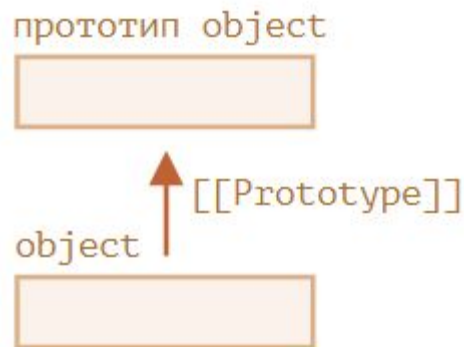
ПЛАН ЗАНЯТИЯ

- Прототип
- This
- Global object
- Call, apply, bind

Часто в программировании 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.

В JavaScript объекты имеют специальное скрытое свойство **[[Prototype]]**, которое либо равно null, либо ссылается на другой объект. Этот объект называется «**прототип**»



Когда мы хотим прочитать свойство из object, а оно отсутствует, JavaScript автоматически берёт его из прототипа.

Прототип

Это внутренняя ссылка на другой объект

Для создания одного объекта на основе другого используется свойство `__proto__`

```
let animal = {  
  eats: true,  
  sleep() {  
    console.log("Zzz-zzz-zz");  
  },  
};  
  
let panda = {  
  __proto__: animal,  
};
```


Прототип

Важное замечание: отношение прототипного наследования - это **отношение между объектами**.

Если один объект имеет специальную ссылку `__proto__` на другой объект, то при чтении свойства из него, если свойство отсутствует в самом объекте, оно ищется в объекте `__proto__`.

```
let animal = {  
  eats: true,  
  walk() {  
    /* этот метод не будет использоваться в rabbit */  
  }  
};  
  
let rabbit = {  
  __proto__: animal  
};  
  
rabbit.walk = function() {  
  alert("Rabbit! Bounce-bounce!");  
};  
  
rabbit.walk(); // Rabbit! Bounce-bounce!
```

Прототип

Свойство для задания
прототипа

```
Object.setPrototypeOf(obj, prototype)
```

Свойство для проверки
прототипа

```
Object.getPrototypeOf(obj)
```

THIS

Ключевое слово ``this`` в JavaScript используется для обращения к текущему объекту. Контекст ``this`` зависит от того, как вызывается функция.

```
const person = {  
  name: 'John',  
  introduce: function() {  
    console.log(`Hello, my name is ${this.name}.`);  
  }  
};  
  
person.introduce(); // "Hello, my name is John."
```

Глобальный объект

Глобальный объект хранит переменные, которые должны быть доступны в любом месте программы.

Это включает в себя как встроенные объекты,

например, **Array**, так и характерные для окружения свойства, например, **window.innerHeight** – высота окна браузера.

```
console.log(window);
```

```
// сработает только в браузере
```

```
alert("Привет"); // это то же самое, что и
```

```
window.alert("Привет");
```

CALL, APPLY

call и apply:


Методы используются для вызова функции с указанием конкретного объекта в качестве `this`. Разница между ними в передаче аргументов - `call` передает аргументы по одному, `apply` передает массив аргументов.

```
function greet(message) {  
  console.log(`${message}, ${this.name}.`);  
}  
  
const person = { name: 'John' };  
  
greet.call(person, 'Hello'); // "Hello, John."  
greet.apply(person, ['Hi']); // "Hi, John."
```

BIND

Метод **bind** создает новую функцию, привязывая указанный объект к `this` внутри функции.

```
function greet(message) {  
    console.log(`${message}, ${this.name}.`);  
}  
  
const person = { name: 'John' };  
  
const greetPerson = greet.bind(person);  
greetPerson('Hola'); // "Hola, John."
```



Ваша новая IT-профессия – Ваш новый уровень жизни

Программирование с нуля в
немецкой школе AIT TR GmbH