JavaScript

Урок №14: Основы. Контекст вызова this.

Автор Бахшиллоев Бехзод
Ментор по Frontend в PRO UNITY

This.

Методом объекта будем называть ключ объекта, который содержит функцию:

```
const furniture = {
    type: 'chair',
    transform: true,
    price: 9,
    getPrice: function () { return furniture.price}, // object method
    getType: function () { return furniture.type} // object method
}
furniture.getPrice(); // 9
furniture.getType(); // 'chair'
```

JavaScript работает внутри браузера, то есть браузер является как бы окружением для языка.
Любая JS функция также запущена в некотором окружении, вот некоторые из них:
- функция запущена как есть в глобальном потоке:

```
    function foo() {};
    foo();
```

В данном случае окружением для функции (то, что её окружает) является глобальный объект - окно браузера (window).

This.

- функция запущена как метод объекта:

```
const user = {
    age: 45,
    getAge: function() { return user.age}
}
user.getAge();
```

В этом случае функция запущена как метод объекта - то есть объект как бы окружает функцию.

Окружение, в котором выполняется функция, называется ***контекстом вызова функции*** (функция вызвана в контексте некоторого окружения).

Контекст вызова содержит область видимости функции, все её переменные и this.

This.

Внутри любой функции можно получить доступ к объекту, который вызвал функцию. Специальное ключевое слово *this* является ссылкой на этот объект (**this** ещё называют объектом контекста).

Можно сказать, что **this** - это ссылка на объект, который вызывает код в данный момент. **this** - всегда некоторый объект (или null).

В строгом режиме работы функции **(‘use strict’)** при запуске функции в глобальном окружении **this** равно undefined (а не window).

This.

Не важно, как была создана функция - this всегда определяется во время вызова функции.

```
const product = {
    price: '15.2 $',
    getPrice: getPrice
}

function getPrice() { return parseFloat(this.price) }
```

product.**getPrice**(); → this в данном случае - ссылка на product

getPrice(); → this в данном случае - ссылка на глобальный объект window

This.

Приём chaining или цепочка - это когда можно вызывать методы объекта через точку, строя цепочку вызовов:
someObject.method1().method2().method3().method1().method3()...

[3,1,2].sort().join();

Для того, чтобы реализовать подобное поведение, функция (метод объекта) всегда должна возвращать этот объект:

```
const myObject = {
    foo: function () { /*...*/ return this;},
    boo: function () { /*...*/ return this;}
}

myObject .foo(); // вернёт this ---> myObject: { foo: function..., boo: func... }
myObject .foo().boo().foo().boo()...
```

Задачи часть 1:

1. Создать объект, который описывает ширину и высоту прямоугольника, а также метод который может посчитать площадь фигуры:
   const rectangle = {width:..., height:..., getSquare:...};
2. Создать объект, у которого будет цена товара и его скидка, а также
   два метода: для получения цены и для расчета цены с учетом скидки:

```
const price = {
    price: 10,
    discount: '15%',
... };
price.getPrice(); // 10
price.getPriceWithDiscount(); // 8.5
```

Задачи часть 2:

3. Создать объект, у которого будет поле высота и метод “увеличить высоту на один”. Метод должен возвращать новую высоту:

object.height = 10;
object.inc(); // придумать свое название для метода
object.height; // 11;

Задачи часть 3:

4. Создать объект “вычислитель”, у которого есть числовое свойство “значение” и методы “удвоить”, “прибавить один”, “отнять один”. Методы можно вызывать через точку, образуя цепочку методов:

```
const numerator = {
    value: 1,
    double: function () {...},
    plusOne: function () {...},
    minusOne: function () {...},
}
numerator.double().plusOne().plusOne().minusOne();
numerator.value // 3
```

This.

Итак, функцию можно вызвать:

- в глобальном контексте foo();
- как метод объекта object.foo();

Существует третий способ вызвать функцию - принудительно указать функции, какой использовать контекст (передать this внутрь функции).

This. Call.

Метод ***call() вызывает*** функцию с указанным значением this и индивидуально предоставленными аргументами.

```
const info = { price: 10 };
function getPrice() {return this.price;}

getPrice.call(info); // 10

function getPriceWithDiscount(discount) {
    return this.price * (100 - discount) / 100;
}
getPriceWithDiscount.call(info, 10); // 9
```

This. Apply.

Метод **apply**() ***вызывает*** функцию с указанным значением this и аргументами, предоставленными в виде массива.

```
const info = {price: 10};

function getPriceWithDiscount(discount, currency) {
    return this.price * (100 - discount) / 100 + currency;
}

getPriceWithDiscount.apply(info, [10, '$']); // 90$
```

This. Потеря контекста.

Потеря контекста - ситуация, когда функция вызывается с другим контекстом, отличным от ожидаемого:

```
const obj = { test: 111, foo: function () { return this.test } };
obj.foo(); // 111

const boo = obj.foo;
// boo = function () { return this.test }

boo(); // this = window, так как функция запущена в глобальном окружении
```

This. Потеря контекста.

Метод **bind**() ***создаёт новую функцию***, которая при вызове устанавливает
в качестве контекста выполнения this предоставленное значение. Bind также позволяет привязывать к функции аргументы

func.**bind**(context, arg1, arg2...)

```
let user = {uName: 'John', getName: function () {return this.uName}};

user.getName(); // 'John'

let getName = user.getName;

getName(); // undefined
getName = user.getName.bind(user);
getName(); // 'John'
```

This. Материалы.

Call, Apply

This

Bind

Задачи часть 4:

1. Создать объект с розничной ценой и количеством продуктов. Этот объект должен содержать метод для получения общей стоимости всех товаров (цена * количество продуктов)
2. Создать объект из предыдущей задачи. Создать второй объект, который описывает количество деталей и цену за одну деталь. Для второго объекта нужно узнать общую стоимость всех деталей, но нельзя создавать новые функции и методы. Для этого “позаимствуйте” метод из предыдущего объекта.
3. Даны объект и функция:
   let sizes = {width: 5, height: 10},
   getSquare = function () {return this.width * this.height};
   Не изменяя функцию или объект, получить результат функции *getSquare* для объекта sizes

Задачи часть 5:

4.

```
let element = {
    height: 25,
    getHeight: function () {return this.height;}
};

let getElementHeight = element.getHeight;
getElementHeight(); // undefined
```

Измените функцию **getElementHeight** таким образом, чтобы можно было вызвать **getElementHeight**() и получить 25.