Contents

[**1.** **Prototype** 2](#_Toc66984153)

[**2.** **\_\_proto\_\_** 2](#_Toc66984154)

[**3.** **Reference Vs Value** 2](#_Toc66984155)

1. **Prototype**

Dùng trong **function constructor**

function Dog(name) { // the usual constructor function

this.name = name

}

Dog.prototype = ...

**prototype** is the object that is used to build \_\_proto\_\_ when you create an object with new.

**prototype** is **not available** on the instances themselves (or other objects), but only on the constructor functions.

Add method/property vào function constructor

1. **\_\_proto\_\_**

Dùng trong **instance** của function constructor

function Dog(name) { // the usual constructor function

this.name = name

}

Let a = new Dog(‘Hector’);

a.\_\_proto\_\_ = ....

Add method/property vào function constructor

**\_\_proto\_\_** is available everywhere.

1. **Reference Vs Value**



Khi gán d = c thì nó chỉ gán địa chỉ/address của c thôi chứ ko gán value của c

Khi ==, === thì nó chỉ so sánh địa chỉ/address thôi chứ ko so sánh value

function add(arr, el) { arr.push(el)} =>Khi truyền input as array thì nó cũng sẽ lấy địa chỉ/address rồi implement chứ ko lấy giá trị của array

function add(arr, el) { arr = [el] } => Nhưng khi gán nó = 1 array khác thì nó sẽ có địa chỉ khác

nếu declare **const arr = [1,2,3]**

thì **vẫn modified được bằng** các phương thức của arr(chứ **ko phải phép gán**). Vì nó ko làm thay đổi địa chỉ/address của const này. Và const obj cũng chỉ lưu địa chỉ/address

this

this (aka "the context") is a special keyword inside each function: giá trị của this chỉ **phụ thuộc** cách function được **gọi/called** (runtime binding)chứ **ko** phải cách nó được định **nghĩa/defined**

1. **this**

theo Mozilla:

*In most cases, the value of this is determined by how a function is called (runtime binding). It can't be set by assignment during execution, and it may be different each time the function is called.*

Được định nghĩa khi function được gọi chứ ko phải khi set/define this. (this nó đi theo tiếng gọi của function). Nên ai gọi nó thì nó sẽ đi theo người đó.

Trong Literal Object

const c = {

        name: 'c',

        go: function () {

            console.log(this)

        },

    }

    c.go()

khởi tạo 1 const c và gọi: c.go() thì **this** sẽ đi theo c

 let { go, name } = c

    go()

còn cách trên là dùng destructuring để lấy các properties và methods trong c ra. Cách này giúp gọn code hơn nhưng có vấn đề:

property name vẫn là của c (ok)

method go vẫn trỏ đến c, nhưng rõ ràng khi ta gọi go() thì go() nó được Window object gọi (vì ko có . ở trước nên mặc định là Window) (failed)

=> ko nên dùng destructuring với methods.

 const c = {

        name: 'c',

        numbers: [1, 2, 3, 4, 5],

        onCount: function () {

            this.numbers.forEach(function(n) {

                console.log(this.name + ' - ' + n)

            })

        }

    }

c.onCount()

c.onCount() thì có vấn đề: trong .forEach là 1 callback function.(console.log(this) trước hàm forEach thì this vẫn là c) Theo Mr.Mosh: function thì sẽ trỏ tới Window

khắc phục: dùng arrow function: (n) => console.log(n) hoặc tạo 1 biến trong hàm onCount let \_this = this

Trong Constructor Function

function A() {

        this.name = 'A';

        this.say = function () {

            console.log(this)

        }

    }

    let m = new A();

    m.say()

dùng từ khóa new để tạo 1 instance từ A

m.say() thì say() sẽ trỏ tới m, m là instance của A nên this trỏ tới A.

Trong Class

class B {

        constructor(name, age) {

            this.name = name;

            this.age = age

        }

        onSayHello() {

            console.log(`Hello I am ${this.name} - ${this.age}`)

        }

    }

    let h = new B('hector', 22);

    h.onSayHello()

tạo instance từ B rồi gọi onSayHello() thì this sẽ trỏ tới h, h là instance từ B nên trở đến B