@startwithmani

@M.serisha Kothapalli

Factory Functions in JS

swipe

@startwithmani

@M.serisha Kothapalli

Factory Functions

Factory functions are those functions which **creates new objects** and **return** them.

In Simple terms

Just like a factory produces **products**. This Factory functions produce **objects**

Why to create a object using function when you have someting like const obj ={} or using new Object().

lest see it with an practical example in next slide.

swipe

@startwithmani

@M.serisha Kothapalli

Need of Factory Functions

let's say you are working an project where you need to create an employee object with their designations.

```
let mani = {
    name: 'Mani',
    designation: 'Developer',
    showInfo() {
        return this.name + ' is a ' + this.designation;
    }
};

let ben = {
    name: 'Ben',
    designation: 'Designer',
    showInfo() {
        return this.name + ' is a ' + this.designation;
    }
};
```

Looks fine, But what if i have 1000's of employees.yes you can do the same thing for all 1000 employees.You are just ending up violating the basic principle of software development DRY(Don't repeat yourself)

swipe

Creation of Factory Functions

- You can just declare a normal **JavaScript function.** which **returns** an **object**. You have factory function ready to use.

```
//using factory function
function createEmployee1(employeeName, employeeDesignation) {
    return {
        name: employeeName,
        designation: employeeDesignation,
        greet() {
            return `Hi I'm ${this.name} and I am ${this.designation} `
            }
        }
}
```

- You can **call** this function as a regular function as many times as you want to create new employee objects.
- let's create two employee objects **mani**, **ben**

```
let mani = createEmployee1('Mani','Developer');
let ben = createEmployee1('Ben','Designer');

console.log(mani);
console.log(ben);
```

swipe

@startwithmani

@M.serisha Kothapalli

Analysing the output

```
▼ {name: 'Mani', designation: 'Developer', greet: ƒ}
    designation: "Developer"
  ▶ greet: ƒ greet()
    name: "Mani"
  ▶ [[Prototype]]: Object

▼ {name: 'Ben', designation: 'Designer', greet: ƒ}
    designation: "Designer"
  ▶ greet: ƒ greet()
    name: "Ben"
  ▶ [[Prototype]]: Object
```

- One main point to notice over here is that **greet method** is in the **same hieracy** just like **name** and **designation properties** are present.
- They are **not** in the prototype but instead they are attached **directly** to the object itself.
- You will understand the reason, why am I pointing this when you move to the next slide

swipe

Memory issues

- The factory function does not use inheritance by default.

lets just try to update the logic of greet function as below

```
//using factory function
function createEmployee1(employeeName, employeeDesignation) {
    return {
        name: employeeName,
        designation: employeeDesignation,
        greet() {
            return `Hi I'm ${this.name} and I am ${this.designation} `
            }
        }
}


let mani = createEmployee1('Mani','Developer');
let ben = createEmployee1('Ben','Designer');

 mani.greet = function (){
    return `Hello, A warm Welcome to you, I'm ${this.name} and I am ${this.designation} `
 }
console.log(mani.greet());  //Hello,A warm Welcome to you, I'm Mani and I am Developer
console.log(ben.greet());   //Hi I'm Ben and I am Designer
```

- The **ben** object **greet** function is **not updated**. **mani's** is **updated**
- So what I was trying to say is that, mani and ben are completely **different objects** in memory. They are occupying **seperate** space in the memory.

swipe

Reusability

- One more thing that I can notice from previous example is that If we wanted to modify the logic of the greet function for all the objects.

- I **can't** really change it by **modifying** at **one place**.I need to change it for every single object **mani**,**ben**.That's not a great idea.
- Think about it when you have thousands of employee's and you are executing 1000 lines of code inside the greet function and every single object function is taking up seperate space in memory.Oops there will memory issues.

- There is a **workaround solution** for this we can see it in further slides.

swipe

Using Object.create

- You can add inheritance with the factory function in order to use the same methods across all objects that you create. In this way, these object methods will be stored in the memory just once.
- **Object.create()** method to add inheritance.This method creates a new object using an existing object.

```
//Syntax

Object.create(proto)
Object.create(proto, propertiesObject)
```

- Lets see in next slide how these object methods can be stored in the memory just once utilizing object.create.

swipe

@startwithmani

@M.serisha Kothapalli

```
const greet = {
    greet() {
        return `Hi I'm ${this.name} and I am ${this.designation} `
    }
}

function createEmployee2(employeeName, employeeDesignation) {
    let employee = Object.create(greet);
    employee.name = employeeName;
    employee.designation = employeeDesignation;
    console.log(employee);
    return employee

}

let mani = createEmployee2('Mani', 'Developer');
let ben = createEmployee2('Ben', 'Designer');

mani.__proto__.greet = function () {
    return `Hello, A warm Welcome to you, I'm ${this.name} and I am ${this.designation} `
}
console.log(mani.greet());
console.log(ben.greet());
```

```
▼ {name: 'Mani', designation: 'Developer'}
    designation: "Developer"
    name: "Mani"
  ▼ [[Prototype]]: Object
    ▶ greet: ƒ ()
    ▶ [[Prototype]]: Object
▼ {name: 'Ben', designation: 'Designer'}
    designation: "Designer"
    name: "Ben"
  ▼ [[Prototype]]: Object
    ▶ greet: ƒ ()
    ▶ [[Prototype]]: Object
```

```
Hello, A warm Welcome to you, I'm Mani and I am Developer
Hello, A warm Welcome to you, I'm Ben and I am Designer
```

Step by step Explanation of code

swipe

- **Step-1** : Remove the greet() method from the createEmployee1 and move this method to another object.
- **Step-2** : So instead of creating an object in function using empty object {...}.lets create the object using object.create().
- **Step-3** : Here we are passing the greet object prototype. so here at this employee object we will be having an empty object with prototype having greet method.

****Important note****

So at this step it will create an empty object and **attaches** greet to **prototype**. Just reiterating the sentence it is attached to prototype but not directly to the object as you can see in first output in previous slide.

swipe

- **Step-4** : In nextstep we are then adding the other properties name and designation to this object.
- **Step-5** : Finally we are returning the object employee.

Updating the greet method

- Now if you are trying to change the logic of the function, we do it by accessing the proto.

- So that fixed the two issues now all together.

  1) mani and ben now share the same object method in memory

  2) If you wanted to modify the logic of the greet fun, you can just do it at one place

swipe

Did you find it helpful??

Like this post!

Share with your friends

Save it for later

Follow for more!

@startwithmani

@M.serisha Kothapalli