Theory: Function constructor

18 minutes 0 / 5 problems solved

You already know how to create an object. Let's imagine a situation where you need to generate a hundred similar objects. Of course, it would be really challenging to do this manually. In this topic, we will learn how to automate this tedious process.

§1. Function constructor

Let's say that you own a huge auto park and buy several new cars every month. Today you have decided to buy a Mercedes and an Audi:

```
const car1000 = {
  brand: "Mercedes",
  year: 2020
};

const car1001 = {
  brand: "Audi",
  year: 2019
};
```

In this example, we created two automobile objects with two properties, but in real life, vehicles have dozens of technical characteristics. Following the previous logic, we must enter each property manually, which doesn't sound terrible for two cars, but what if you need to add a hundred cars with real data?

When manipulating similar objects, it is easier to use the **function constructor**. Let's find out how it works.

```
function Car(brand, year) {
  this.brand = brand;
  this.year = year;
}

const car1000 = new Car("Mercedes", 2020);
const car1001 = new Car("Audi", 2019);
```

We have defined the `Car` function and used it to create two car objects using the keyword `new`. In this case, the `Car` function is called the **function constructor**. It looks like a regular function but it is used to create similar objects.

§2. Function constructor features

In the example above, we created `car1000` and `car1001` objects, both of which are instances of `Car` objects. The result of invoking `new Car ("Mercedes", 2020)` is as follows:

```
{
  brand: "Mercedes",
  year: 2020
}
```

This way, we don't have to define an object every time we add a new car. Instead, we call the car constructor function with specific parameters.

The constructor functions have some unique features.

1. We spelled the constructor function with a capital letter (`Car`). While this is not a rule, it is a naming convention between developers. Please use this recommendation: it allows us to find the function constructor faster.

2. It is essential to call the function constructor with the keyword `new`.

Current topic:

Function constructor

Topic depends on:

× Type conversion

Table of contents:

3. Inside the constructor function, `this` does not have a value. After a new object has been created, the value of `this` becomes the new object.

4. Constructors always return a new object without using the `return` keyword inside them. By default, they return `this`. You can easily change this logic and use `return` whenever you want.

```
function Car(brand, year) {
  this.brand = brand;
  this.year = year;
  return year;
}

const myCar = new Car("BMW", 2001);

console.log(myCar); // Car { brand: 'BMW', year: 2001 }
```

As a result, we will see `Car { brand: 'BMW', year: 2001 }`.

§3. Methods in constructor

You can create methods as well as properties inside regular objects. The same is true for constructor functions.

```
function Car(model, speed) {
  this.model = model;
  this.speed = speed;

  this.getSpeed = function() {
 console.log("The speed of " + this.model + " is: " + this.speed + " km per hou
r");
  };
}
```

As you can see, we used `this` to refer to an object in its method.

Then we can call `getSpeed` method:

```
const carKIA = new Car("KIA Stinger", 209);
carKIA.getSpeed();
```

The console displays "The speed of KIA Stinger is 209 km per hour".

§4. Conclusion

Constructors are a common way to create similar objects. They are regular functions, but based on the Convention, we should name them with a capital letter. To create a new object, we need to call the constructor function with the keyword `new` and the parameters for the object. You can also manipulate object variables inside the function constructor method.

Report a typo

55 users liked this theory. **0** didn't like it. **What about you?**

Start practicing

Comments (2) Hints (0) Useful links (0) Show discussion