Prototypal Inheritance

3rd Jun, 2020 12:00 PM

Albert Sebastian

Week # 7 | Day # 3

JavaScript prototype

Prototypes are the mechanism by which JavaScript objects inherit features from one another. In this session, we explain how prototype chains work and look at how the prototype property can be used to add methods to existing constructors.

Javascript a prototype based language

JavaScript is often described as a prototype-based language — to provide inheritance, objects can have a prototype object, which acts as a template object that it inherits methods and properties from.

An object's prototype object may also have a prototype object, which it inherits methods and properties from, and so on. This is often referred to as a prototype chain, and explains why different objects have properties and methods defined on other objects available to them.

- If we had to create a person object
- Lets say we keep name and energy of that person

```
var person = {}
person.name = 'Jay'
person.energy = 10

person.eat = function (amount) {
  console.log(this.name,'is eating.')
  this.energy += amount
}

person.sleep = function (time) {
  console.log(this.name,'is sleeping.')
  this.energy += time
}

person.code = function (time) {
  console.log(this.name,'is coding.')
  this.energy -= time
}
```

If we need multiple instances, we could wrap this in a function and use as a construtor

```
function Person(name,energy){
  var person = {}
  person.name = name
  person.energy = energy

 person.eat = function (amount) {
    console.log(this.name,'is eating.')
    this.energy += amount
  }

  person.sleep = function (time) {
    console.log(this.name,'is sleeping.')
    this.energy += time
  }

  person.code = function (time) {
    console.log(this.name,'is coding.')
    this.energy -= time
  }
  return person
}

var sai = Person('Sai',15)
var jay = Person('Jay',12)
```

By using a constructor, now whenever we want to create a new instance or person, we can invoke the Person function, and passing the necessary arguments into it.

Now this is simple and works.

But if you think about it, the three methods that are in the Person object are being created again with every instance.

In this case, all the methods are common/general. Is there a way to save memory?

Instead of creating new methods on the go, we can group them together for the Person object.

```
var personMethods = {
 eat(amount) {
    console.log(this.name,'is eating.')
    this.energy += amount
  },
  sleep(time) {
    console.log(this.name,'is sleeping.')
    this.energy += time
  },
  code(time) {
    console.log(this.name,'is coding.')
    this.energy -= time
  }
}

function Person(name,energy){
  var person = {}
  person.name = name
  person.energy = energy
  person.eat = personMethods.eat
  person.sleep = personMethods.sleep
  person.code = personMethods.code

  return person
}

var sai = Person('Sai',15)
var jay = Person('Jay',12)
```

Now we have made it better by using a shared function.

But we can make it better by using Object.create. it allows you to create an objects, using an existing object, and allows you to look into that object if there is a property which isnt available in child object.

Object.create MDN

example

```
var parent = {
  name:'parent',
  age:40,
  country:'India'
}

var child = Object.create(parent)
child.name = 'child'
child.age = 10
// Its quite similar to how you use new Person()
```

```
var personMethods = {
 eat(amount) {
    console.log(this.name,'is eating.')
    this.energy += amount
  },
  sleep(time) {
    console.log(this.name,'is sleeping.')
    this.energy += time
  },
  code(time) {
    console.log(this.name,'is coding.')
    this.energy -= time
  }
}

function Person(name,energy){
  var person = Object.create(personMethods)
  person.name = name
  person.energy = energy

  return person
}

var sai = Person('Sai',15)
var jay = Person('Jay',12)
```

Now instead of storing this information inside a seperate object.

Javascript has a feature, and its called `prototype`. Every function in JavaScript has a prototype property that references an object.

This is called a `prototypal inheritance`


```
function Person(name,energy){
  var person = Object.create(Person.prototype)
  person.name = name
  person.energy = energy

  return person
}

Person.prototype.eat = function (amount) {
  console.log(this.name,'is eating.')
  this.energy += amount
}

Person.prototype.sleep = function (time) {
  console.log(this.name,'is sleeping.')
  this.energy += time
}

Person.prototype.code = function (time) {
  console.log(this.name,'is coding.')
  this.energy -= time
}

var sai = Person('Sai',15)
var jay = Person('Jay',12)
```

All JavaScript objects inherit properties and methods from a prototype: `prototype` is a property that every function in JavaScript has. it allows us to share the methods across all instances of a function.

Now you have made a blueprint of your own object.

Note:

`Only modify your own prototypes. Never modify the prototypes of standard JavaScript objects.`

We already know how to use the new keyword

we can change the syntax like this

```
function Person(name,energy){
  this.name = name
  this.energy = energy
}

Person.prototype.eat = function (amount) {
  console.log(this.name,'is eating.')
  this.energy += amount
}

Person.prototype.sleep = function (time) {
  console.log(this.name,'is sleeping.')
  this.energy += time
}

Person.prototype.code = function (time) {
  console.log(this.name,'is coding.')
  this.energy -= time
}

var sai = new Person('Sai',15)
var jay = new Person('Jay',12)
```

Object.create builds an object that inherits directly from the one passed as its first argument.

With constructor functions, the newly created object inherits from the constructor's prototype,

By the end of this you should have a better understanding of how object oriented javascript works

Sources:

MDN

3rd Jun, 2020 01:15 PM