JavaScript call() Method

Summary: in this tutorial, you will learn about the JavaScript `call()` method and how to use it more effectively.

Introduction to the JavaScript call() method

In JavaScript, a function is an instance of the `Function` type. For example:

```
function add(x, y) {
  return x + y;
}

console.log(add instanceof Function); // true
```

The `Function.prototype` type has the `call()` method with the following syntax:

```
functionName.call(thisArg, arg1, arg2, ...);
```

In this syntax, the `call()` method calls a function `functionName` with the arguments (`arg1` , `arg2` , ...) and the `this` set to `thisArg` object inside the function.

- The `thisArg` is the object that the `this` object references inside the function `functionName` .
- The `arg1` , `arg2` , .. are the function arguments passed into the `functionName` .

The `call()` method returns the result of calling the `functionName()` .

The following example defines the `add()` function and calls it normally:

```
function add(x, y) {
  return x + y;
```

```
}

let result = add(10, 20);
console.log(result); // 30
```

The following calls the `add()` function but use the `call()` method instead:

```
function add(x, y) {
  return x + y;
}

let result = add.call(this, 10, 20);
console.log(result); // 30
```

By default, the `this` inside the function is set to the global object i.e., `window` in the web browsers and `global` in Node.js.

> Note that in the strict mode, the `this` inside the function is set to `undefined` instead of the global object.

Consider the following example:

```
var greeting = 'Hi';

var messenger = {
    greeting: 'Hello'
}

function say(name) {
    console.log(this.greeting + ' ' + name);
}
```

Inside the `say()` function, we reference the `greeting` via the `this` value. If you just invoke the `say()` function via the `call()` method as follows:

```
say.call(this,'John');
```

It'll show the following output to the console:

```
"Hi John"
```

However, when you invoke the `call()` method of `say` function object and pass the `messenger` object as the `this` value:

```
say.call(messenger,'John');
```

The output will be:

```
"Hello John"
```

In this case, the `this` value inside the `say()` function references the `messenger` object, not the global object.

Using the JavaScript call() method to chain constructors for an object

You can use the `call()` method for chaining constructors of an object. Consider the following example:

```
function Box(height, width) {
  this.height = height;
  this.width = width;
}

function Widget(height, width, color) {
  Box.call(this, height, width);
  this.color = color;
}
```

```
let widget = new Widget('red', 100, 200);

console.log(widget);
```

Output:

```
Widget { height: 'red', width: 100, color: 200 }
```

In this example:

- First, initialize the `Box` object with two properties: `height` and `width`.
- Second, invoke the `call()` method of the `Box` object inside the `Widget` object, set the `this` value to the `Widget` object.

Using the JavaScript call() method for function borrowing

The following example illustrates how to use the call() method for borrowing functions:

```
const car = {
  name: 'car',
  start() {
    console.log('Start the ' + this.name);
  },
  speedUp() {
    console.log('Speed up the ' + this.name);
  },
  stop() {
    console.log('Stop the ' + this.name);
  },
};

const aircraft = {
  name: 'aircraft',
  fly() {
    console.log('Fly');
  },
};
```

```
car.start.call(aircraft);
car.speedUp.call(aircraft);
aircraft.fly();
```

Output:

```
Start the aircraft
Speed up the aircraft
Fly
```

How it works.

First, define a car object with one property name and three methods `start`, `speedUp`, and `stop`:

```
const car = {
  name: 'car',
  start() {
    console.log('Start the ' + this.name);
  },
  speedUp() {
    console.log('Speed up the ' + this.name);
  },
  stop() {
    console.log('Stop the ' + this.name);
  },
};
```

Second, define the aircraft object with one property name and a method:

```
const aircraft = {
  name: 'aircraft',
  fly() {
    console.log('Fly');
  },
};
```

Third, call the `start()` and `speedUp()` method of the `car` object and the `fly()` method of the `aircraft` object. However, passing the `aircraft` as the first argument into the `start()` and `speedUp()` methods:

```
car.start.call(aircraft);
car.speedUp.call(aircraft);
aircraft.fly();
```

Inside the `start()` and `speedUp()` methods, the `this` references the `aircraft` object, not the `car` object. Therefore, the `this.name` returns the `'aircraf'` string. Hence, the methods output the following message:

```
Start the aircraft
Speed up the aircraft
```

Technically, the `aircraft` object borrows the `start()` and `speedUp()` method of the `car` object. And function borrowing refers to an object that uses a method of another object.

The following example illustrates how the `arguments` object borrows the `filter()` method of the `Array.prototype` via the `call()` function:

```
function isOdd(number) {
  return number % 2;
}

function getOddNumbers() {
  return Array.prototype.filter.call(arguments, isOdd);
}

let results = getOddNumbers(10, 1, 3, 4, 8, 9);
console.log(results);
```

Output:

```
[ 1, 3, 9 ]
```

How it works.

First, define the `isOdd()` function that returns true if the number is an odd number:

```
function isOdd(number) {
  return number % 2;
}
```

Second, define the `getOddNumbers()` function that accepts any number of arguments and returns an array that contains only odd numbers:

```
function getOddNumbers() {
  return Array.prototype.filter.call(arguments, isOdd);
}
```

In this example, the `arguments` object borrows the filter() method of the `Array.prototype` object.

Third, call the `getOddNumbers()` function:

```
let results = getOddNumbers(10, 1, 3, 4, 8, 9);
console.log(results);
```

In this tutorial, you have learned about the JavaScript `call()` method and how to use it more effectively.