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 (https://www.javascripttutorial.net/javascript-function/) is an instance of the `Function` (https://www.javascripttutorial.net/javascript-function-type/) type. For example:

```
function show(){
    //...
}

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

The `Function` type has a method named `call()` with the following syntax:

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

The `call()` method calls a function `functionName` with a given `this` value and arguments.

The first argument of the `call()` method `thisArg` is the `this` value. It allows you to set the `this` value to any given object.

The remaining arguments of the `call()` method `arg1`, `arg2`,... are the arguments of the function.

When you invoke a function, the JavaScript engine invokes the `call()` method of that function object.

Suppose that you have the `show()` function as follows:

```
function show() {
   console.log('Show function');
```

```
}
```

And invoke the `show()` function:

```
show();
```

It is equivalent to invoke the `call()` method on the `show` function object:

```
show.call();
```

By default, the `this` (https://www.javascripttutorial.net/javascript-this/) value inside the function is set to the global object i.e., `window` on web browsers and `global` on node.js:

```
function show() {
    console.log(this);
}

show();
```

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

See the following example:

```
function add(a, b) {
    return a + b;
}

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

Output:

```
30
```

In this example, instead of calling the `add()` function directly, we use the `call()` method to invoke the `add()` function. The `this` value is set to the global object.

See the following code:

```
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 will show the following result:

```
"Hi John"
```

However, when you invoke the `call()` method of `say()` function 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

The `call()` method can be used for chaining constructors for 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);
```

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 defines two objects: `car` and `aircraft`:

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

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

```
    }
};
```

The `aircraft` object has the `fly()` method.

The following code uses the `call()` method to invoke the `start()` method of the `car` object on the `aircraft` object:

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

Here is the output:

```
Start the aircraft
```

Technically, the `aircraft` object has borrowed the `start()` method of the `car` object for the `aircraft`.

When an object uses a method of another object is called the function borrowing.

The typical applications of function borrowing are to use the built-in methods of the Array (https://www.javascripttutorial.net/javascript-array/) type.

For example, the `arguments` object inside a function is an array-like object, not an array object. To use the `slice()` (https://www.javascripttutorial.net/javascript-array-slice/) method of the Array object, you need to use the `call()` method:

```
function getOddNumbers() {
    const args = Array.prototype.slice.call(arguments);
    return args.filter(num => num % 2);
}

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

Output:

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

In this example, we passed any number of numbers into the function. The function returns an array of odd numbers.

The following statement uses the `call()` function to set the `this` inside the `slice()` method to the `arguments` object and execute the `slice()` method:

```
const args = Array.prototype.slice.call(arguments);
```

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