What are the

Call, Apply & Bind methods in JavaScript

M. SHAHZEB RAZA

REACT WEB DEVELOPER

Imagine that we have 02 person objects.
person_A & **person_B**.
Both contain a **firstName** property and a **greet** method.

call_method.js

```
const person_A = {
    firstName: "Shahzeb",
    greet: (friendName)=>{
      console.log(`Hey ${friendName}! I'm ${this.firstName}.`)
    }
}
const person_B = {
    firstName: "Junaid",
    greet: (friendName)=>{
      console.log(`Hey ${friendName}! I'm ${this.firstName}.`)
    }
}

person_A.greet('Ali') // Output: "Hey Ali! I'm Shahzeb"
person_B.greet('Santiago') // Output: "Hey Santiago! I'm Junaid."
```

Notice that **person_A** & **person_B** have the identical **greet** methods.
This means we're repeating our code. This may not look like a big problem but imagine if need to create 05 more persons.

Doesn't look very good. Does it? 😖

call_method.js

```
const person_A = {
    firstName: "Shahzeb",
    greet: (friendName)=>console.log(`Hey ${friendName}! I'm ${this.firstName}.`)
}
const person_B = {
    firstName: "Junaid",
    greet: (friendName)=>console.log(`Hey ${friendName}! I'm ${this.firstName}.`)
}

const person_C = {
    firstName: "Nouman",
    greet: (friendName)=>console.log(`Hey ${friendName}! I'm ${this.firstName}.`)
}
const person_D = {
    firstName: "Geralt",
    greet: (friendName)=>console.log(`Hey ${friendName}! I'm ${this.firstName}.`)
}
const person_E = {
    firstName: "Siri",
    greet: (friendName)=>console.log(`Hey ${friendName}! I'm ${this.firstName}.`)
}
```

Hmm! There should be a better way!

And there is !

Call Method

"The `call()` method calls the function with a given `this` value and arguments provided individually" - MDN

which means,

it lets you call a function or object method for another object without specifying it for the the other object.

First argument **`thisArg`** is used to set the **`this`** variable of the function.
Think of it as setting the context of the function from where certain values can be pulled into the called function.

But wait !

*How can we use **`call`** to optimize our code?*

Look at the optimized code below.

call_method.js

```
const person_A = {
    firstName: "Shahzeb",
    greet: (friendName)=>{
        console.log(`Hey ${friendName}! I'm ${this.firstName}.`)
    }
}
const person_B = {
    firstName: "Junaid",
}

person_A.greet('Ali') // Output: "Hey Ali! I'm Shahzeb"
person_A.greet.call(person_B, 'Santiago') // Output: "Hey Santiago! I'm Junaid"
```

In the code above, we have used the **`greet`** method of **`person_A`** without redefining it in **`person_B`**.

This act of using a method of one object on a different object without copying it is called ***function borrowing***.

call method can also be used with an independent function.

call_method.js

```
const new_greet = (friendName)=>{
        console.log(`Hey ${friendName}! I'm ${this.firstName}.`)
}

const person_A = {
    firstName: "Shahzeb",
}
const person_B = {
    firstName: "Junaid",
}

new_greet.call(person_A, 'Ali') // Output: "Hey Ali! I'm Shahzeb"
new_greet.call(person_B, 'Santiago') // Output: "Hey Santiago! I'm Junaid"
```

In the code above, we are calling the **new_greet** function twice with a **different context** (**this** variable) each time, and hence a different output is shown each time.

*Let's take a look at **apply** method now*

Apply Method

"The `apply()` method calls the function with a given `this` value, and arguments provided as an array" - MDN

which means,

it works exactly like the `call` method, except that it accepts the arguments (other than `this` variable) as an array.

apply_method.js

```
yourFunction.apply(thisArg, [...args] )
```

Didn't get it? No problem!

Let's take a look at an example !

In the example below, we use **`call`** & **`apply`** methods to run our **`getFavFoods`** function.

apply_method.js

```
const getFavFoods = (food_1, food_2)=>{
        console.log(`I'm ${this.name}! I like ${food_1} & ${food_2}.`)
}

const person = { name: 'Shahzeb' }

getFavFoods.call(
  person,
  'Pizza','Shawarma'
) // Output: "I'm Shahzeb! I like Pizza & Shawarma"

getFavSnacks.apply(
  person,
  ['Chocolate', 'Burger']
) // Output: "I'm Shahzeb! I like Chocolate & Burger"
```

Note that a similar ouput is received both times. However, in the with **`apply`** method, we wrap all the arguments, except **`thisArg`** in an array.

Bind Method

"The `bind()` method creates a new function that, when called, has its `this` keyword set to the provided value, with a given sequence of arguments preceding any provided when the new function is called." - MDN

which means,

it lets you copy a *function* or *object method* while storing the **`this`** context and any provided *arguments* inside the copiedFunction.

bind_method.js

```
const funcCopy = functionToCopy.bind(thisArg, arg1, arg2)
```

Still Unclear ?

Hmm! Let's take at a look at an example!

Sometimes, we find ourselves in situation when we're calling the same function with mostly the same arguments.

For Instance,

bind_method.js

```
const getFavFood = (food_1,food_2,food_3)=>{
        console.log(`Hey I'm ${this.name}! I like eating ${food_1,${food_2 &
${food_3}.`)
}
const person_A = { name: 'Shahzeb'}
const person_B = { name: 'Junaid'}

getFavFood.call(person_A, 'Shawarma', 'Burger', 'Pasta')
// Output: "I'm Shahzeb! I like eating Shawarma, Burger & Pasta."
getFavFood.call(person_B,'Shawarma', 'Burger', 'Steak')
// Output: "I'm Shahzeb! I like eating Shawarma, Burger & Steak."
```

It's clear that first 02 arguments for both the persons is same. There may be a case when multiple functions may have multiple arguments common with each other.

In such cases, we can use **bind** method to *create a function copy bound to the given* **this** *context & the repeated/common arguments.*

The above code can be replaced with the following code!

bind_method.js

```
const getFavFood = (food_1,food_2,food_3)=>{
        console.log(`Hey I'm ${this.name}! I like eating ${food_1,${food_2 &
${food_3}.`)
}
const person_A = { name: 'Shahzeb'}
const person_B = { name: 'Junaid'}

const getFoodsForA = getFavFood.bind(person_A, 'Shawarma', 'Burger');
const getFoodsForB = getFavFood.bind(person_B, 'Shawarma', 'Burger');


getFoodsForA('Pasta')
// Output: "I'm Shahzeb! I like eating Shawarma, Noodles & Pasta."

getFoodsForB('Steak')
// Output: "I'm Shahzeb! I like eating Shawarma, Noodles & Steak."
```

Not a very good function name but it does the work !

The new `getFoodsForA` function is now bound with `this` context of `person_A` and *first 02 arguments*.

Let's take a look at another example!

In the following example, we are repeatedly calling the `add()` function with same 1st argument: 2

bind_method.js

```
const add= (num_A, num_B)=>{
        console.log(num_A + num_B)
}
console.log(add(2,3)) // Output: 5
console.log(add(2,8)) // Output: 10
console.log(add(2,2)) // Output: 4
console.log(add(2,1)) // Output: 3
```

`bind()` method can be used here to optimize this code.

Binding the **2** to the 1st argument of `add( )` saves us from writing it again each time we want to add something to ***5***

bind_method.js

```
const add= (num_A, num_B)=>{
        console.log(num_A + num_B)
}
// binds context: 'null' and 1st argument: '2' to
addToTwo
const addToTwo = add.bind(null,2)
console.log(addToTwo(3)) // Output: 5
console.log(addToTwo(8)) // Output: 10
console.log(addToTwo(2)) // Output: 4
console.log(addToTwo(1)) // Output: 3
```

Phew! That was a lot to take in!

Let's review what we have learnt !

Recap

Call Method

Let's us call a "*function*" / "*object method*" by replacing its **`this`** context with a provided **`this`** context.

Apply Method

Calls a "*function*" / "*object method*" with a given **`this`** context just like call method.
The only difference is that all the arguments except the **`this`** context are passed in an array.

Bind Method

Creates a copy of a function while storing the provided **`this`** *context as well as any arguments inside the copiedFunction.*

M. SHAHZEB RAZA

SWIPE 👉

Did you found it Helpful?

M. SHAHZEB RAZA

REACT WEB DEVELOPER

Hit Like ❤️

if you learnt something! 😺

FOLLOW FOR MORE!