call() apply() bind() Methods in JavaScript

Introduction

JavaScript is a flexible language that offers different methods to alter the context (this) of a function. Among these, the call(), apply(), and bind() techniques are crucial instruments for all JavaScript programmers. These techniques empower you to manage the execution context of functions, facilitating greater flexibility and reusability in your code. Now, let's explore each of these methods with in-depth explanations and illustrations!

The `call()` Method

The `call()` method invokes a function with a specified `this` value and arguments provided individually.

Syntax :

```
functionName.call(thisArg, arg1,arf2,...);
```

Example :

```javascript
function greet(greeting, punctuation) {
    console.log(greeting + ', ' + this.name + punctuation);
}

const person = {
    name: 'Alice'
};

greet.call(person, 'Hello', '!');

```

Output:

```
Hello, Alice!
```

In this example, the `greet` function is called with `this` set to the `person` object, and the arguments `'Hello'` and `'!'` are passed individually.

The `apply()` Method

The `apply()` method is similar to `call()`, but it takes an array of arguments instead of listing them individually.

Syntax:

```javascript
functionName.apply(thisArg, [argsArray]);
```

Example:

```javascript
function greet(greeting, punctuation) {
    console.log(greeting + ', ' + this.name + punctuation);
}

const person = {
    name: 'Bob'
};

greet.apply(person, ['Hi', '?']);
```

Output

```
Hi, Bob?
```

Here, `greet` is called with `this` set to `person`, and the arguments are passed as an array.

The `bind()` Method

The **`bind()`** method of `Function` instances creates a new function that, when called, calls this function with its `this` keyword set to the provided value, and a given sequence of arguments preceding any provided when the new function is called.

Syntax:

```
const boundFunction = functionName.bind(thisArg, arg1, arg2, ...);
```

Example:

```
function greet(greeting, punctuation) {
    console.log(greeting + ', ' + this.name + punctuation);
}

const person = {
    name: 'Charlie'
};

const greetCharlie = greet.bind(person, 'Hey');
greetCharlie('!!!');
```

Output

```
Hey, Charlie!!!
```

When you look at this example, welcomeCharlie comes into play as a fresh feature with this linked to individual and the primary point arranged to 'Greetings'. Once triggered, it records the welcoming with the supplied punctuation.

Use Cases and Differences

Use Cases

- Utilization Instances and Variances Usage Instances call() and apply() prove to be beneficial for summoning functions in an alternate framework. To provide an example, you may opt for call() or apply() when adopting methods from another entity.
- bind() proves to be beneficial for establishing functions with an established this value. It is frequently employed in event handlers where you desire to guarantee that this pertains to a particular entity.

Differences

- Arguments: call() accepts arguments separately, whereas apply() accepts arguments as an array.
- Invocation: call() and apply() activate the function right away, meanwhile bind() yields a new function that can be activated at a later time.

Advanced Topics

Partial Application

With `bind()`, you can create a partially applied function, where some arguments are preset.

Example:

```
function multiply(a, b) {
    return a * b;
}

const double = multiply.bind(null, 2);
console.log(double(5)); // Output: 10
```

Here, `double` is a partially applied function where `a` is fixed at 2.

Borrowing Methods

You can borrow methods from other objects using `call()` or `apply()`.

Example:

```
const arrayLike = {
    0: 'a',
    1: 'b',
    2: 'c',
    length: 3
};

const array = Array.prototype.slice.call(arrayLike);
console.log(array); // Output: ['a', 'b', 'c']
```

In this example, the `slice` method is borrowed from `Array.prototype` to convert an array-like object into a real array.

Function Currying

Using `bind()`, you can create curried functions, which are functions that take arguments one at a time.

Example:

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

```

```
const addFive = add(5);
console.log(addFive(10)); // Output: 15
```

Here, `add` is a curried function that returns a new function with `a` fixed.

Conclusion

Grasping call(), apply(), and bind() is essential for excelling in manipulating function context in JavaScript. These techniques improve the adaptability and recyclability of your code by empowering you to manage the this value and preset arguments for functions. Whether you're borrowing techniques, generating partially employed functions, or guaranteeing uniform context in event handlers, these utilities are indispensable in a JavaScript programmer's arsenal.

Frequently Asked Questions

1. What is the primary purpose of the call() method?

2. How does the apply() method differ from call()?

3. When should I use bind() instead of call() or apply()?

Geekster