JavaScript call(), apply(), and bind() Methods

These methods are used to manipulate the **this** context in JavaScript functions and to invoke functions in specific ways. They are particularly useful for function borrowing, function currying, and creating reusable function templates.

1. call() Method

The call() method calls a function with a given **this** value and arguments provided individually.

Syntax

function.call(thisArg, arg1, arg2, ...)

- **thisArg**: The value to use as **this** when calling the function.
- **arg1, arg2**, ...: Arguments for the function.

Example

```
const person = {
  fullName: function (city, country) {
    return `${this.firstName} ${this.lastName}, ${city}, ${country}`;
  },
};

const person1 = {
  firstName: "John",
  lastName: "Doe",
};

console.log(person.fullName.call(person1, "New York", "USA"));

// Output: "John Doe, New York, USA"
```

In this example, we're using call() to borrow the fullName method from person and use it with person1's data.

2. apply() Method

The apply() method is similar to call(), but it accepts arguments as an array (or an array-like object).

Syntax

function.apply(thisArg, [argsArray])

- **thisArg**: The value to use as this when calling the function.

- **argsArray**: An array or array-like object specifying the arguments for the function.

Example

```
const numbers = [5, 6, 2, 3, 7];

const max = Math.max.apply(null, numbers);
console.log(max); // Output: 7

const min = Math.min.apply(null, numbers);
console.log(min); // Output: 2
```

Here, we're using apply() to find the maximum and minimum values in an array. The null is used as thisArg because Math.max() and Math.min() don't use this.

3. bind() Method

The bind() method creates a new function that, when called, has its this keyword set to the provided value. It allows you to fix the value of this for a function, regardless of how it's called.

Syntax

javascript

Copy

function.bind(thisArg, arg1, arg2, ...)

- **thisArg**: The value to be passed as the this parameter to the target function.
- **arg1, arg2**, ...: Arguments to prepend to arguments provided to the bound function.

Example

```
const module = {
  x: 42,
  getX: function () {
    return this.x;
  },
};

const unboundGetX = module.getX;
console.log(unboundGetX()); // Output: undefined (this is not bound)

const boundGetX = unboundGetX.bind(module);
console.log(boundGetX()); // Output: 42
```

In this example, **bind()** is used to create a new function boundGetX with this permanently set to module.

Partial Application with bind()

bind() can also be used for partial application of functions:

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

const double = multiply.bind(null, 2);
console.log(double(4)); // Output: 8
```

Here, we create a new function double that always multiplies its argument by 2.

Key Differences

1. call() and apply() invoke the function immediately, while bind() returns a new function.
2. call() accepts an argument list, apply() accepts a single array of arguments.
3. bind() can be used for partial application of functions.