Gmail Bikash Adhikari <virtual.bikash@gmail.com>

bind, call and apply in JavaScript

1 message

Bikash <bikash.caim@gmail.com>
To: virtual.bikash@gmail.com
Fri, Mar 7, 2025 at 8:17 PM

`bind`, `call`, and `apply` are JavaScript methods that allow you to explicitly set the `this` value of a function when it's invoked. They are crucial for controlling the context in which a function executes, especially in object-oriented JavaScript and when dealing with callbacks.

Here's a breakdown of each method:

1. `call()`

- **Purpose:** Invokes a function with a specified `this` value and arguments provided individually.
- **Syntax:** `function.call(thisArg, arg1, arg2, ...)`
  - `thisArg`: The value to be used as `this` when the function is called.
  - `arg1, arg2, ...`: The arguments to be passed to the function, separated by commas.
- **Example:**

JavaScript

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

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

const result = person.fullName.call(person1, "New York", "USA");
console.log(result); // Output: John Doe, New York, USA
```

In this example, `person.fullName` is called with `person1` as the `this` value.

2. `apply()`

- **Purpose:** Similar to `call()`, it invokes a function with a specified `this` value, but it takes arguments as an array (or array-like object).
- **Syntax:** `function.apply(thisArg, [argsArray])`

  - `thisArg`: The value to be used as `this`.
  - `[argsArray]`: An array (or array-like object) containing the arguments to be passed to the function.
- **Example:**

JavaScript

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

const person1 = {
  firstName: "Jane",
  lastName: "Smith"
};

const args = ["London", "UK"];
const result = person.fullName.apply(person1, args);
console.log(result); // Output: Jane Smith, London, UK
```

The key difference from `call()` is that `apply()` expects an array of arguments.

3. `bind()`

- **Purpose:** Creates a new function that, when called, has its `this` value set to the provided value. It does *not* immediately invoke the function.
- **Syntax:** `function.bind(thisArg, arg1, arg2, ...)`
  - `thisArg`: The value to be used as `this`.
  - `arg1, arg2, ...`: Optional arguments to be prepended to the arguments provided when the new function is called.
- **Example:**

JavaScript

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

const person1 = {
  firstName: "David",
      lastName: "Brown"
    };

    const myFunc = person.fullName.bind(person1, "Paris"); // returns a new function
    const result = myFunc("France"); // Now the function is called.
    console.log(result); // Output: David Brown, Paris, France
```

In this case, `bind()` creates a new function `myFunc` where `this` is permanently bound to `person1`, and the first argument is permanently set to "Paris". When `myFunc` is called later, it only requires the remaining argument ("France").

Key Differences Summarized:

- `call()` **and** `apply()`**:** Immediately invoke the function. `call()` takes arguments individually, while `apply()` takes them as an array.
- `bind()`**:** Creates a new function with the specified `this` value and optional pre-set arguments. It does not invoke the function immediately.

Use Cases:

- **Setting** `this` **in callbacks:** When passing a method as a callback, `bind()` is often used to ensure the method retains its original `this` context.
- **Inheritance:** `call()` and `apply()` are used to invoke parent constructor functions in child constructors.
- **Function currying:** `bind()` can be used to create new functions with pre-set arguments.
- **Event handlers:** `bind()` can be useful to bind a specific context to an event handler.

These methods provide powerful ways to manipulate the `this` context in JavaScript, leading to more flexible and robust code.

Sources
1. https://jianline.com/javascript-bind-call-apply/
2. https://github.com/A4lfr32/Javascript