Call, apply, and bind methods

Absolutely! Let's break down `call`, `apply`, and `bind` in JavaScript — three powerful methods used to control the `this` context of a function.

🧠 What is `this` in JavaScript?

In JavaScript, `this` refers to the **object that is executing the current function**.

But sometimes, you want to **explicitly set** what `this` refers to — and that's where `call`, `apply`, and `bind` come in.

🔧 The Setup Example

```
const person = {
  name: "Abhi",
  greet(greeting) {
    console.log(`${greeting}, my name is ${this.name}`);
  }
};
```

Now suppose we want to borrow `greet` and use it with a different object:

```
const user = { name: "Alex" };
```

🔷 1. `call()`

✅ Syntax:

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

✅ Description:

Invokes the function **immediately** and **sets** `this` **explicitly** to `thisArg`.

```
person.greet.call(user, "Hello");
// Output: Hello, my name is Alex
```

> call executes the function right away with the provided this and arguments.

🔷 2. `apply()`

✅ Syntax:

```
function.apply(thisArg, [argsArray])
```

✅ Description:

Same as `call`, but **takes arguments as an array**.

```
person.greet.apply(user, ["Hi"]);
// Output: Hi, my name is Alex
```

✅ Use `apply()` when you already have arguments in an array.

🔷 3. `bind()`

✅ Syntax:

```
const newFunction = function.bind(thisArg, arg1, arg2, ...)
```

✅ Description:

Returns a **new function** with `this` bound permanently to `thisArg`.

It **does not execute immediately** — you can call it later.

```
const greetAlex = person.greet.bind(user, "Hey");
greetAlex(); // Output: Hey, my name is Alex
```

> bind is useful when you want to delay execution but lock this.

🎯 Real-World Use Case: `setTimeout`

```
const obj = {
  name: "Abhi",
  sayName() {
    console.log(`My name is ${this.name}`);
  }
};

setTimeout(obj.sayName, 1000); // ❌ undefined (or global object)

setTimeout(obj.sayName.bind(obj), 1000); // ✅ My name is Abhi
```

> bind helps maintain context when functions lose this (e.g., in setTimeout, event listeners, etc.).

🔍 Comparison Table

Method	Executes Immediately?	Arguments Format	Returns a New Function?	Use Case
`call`	✅ Yes	Separate args: `arg1, arg2`	❌ No	Function borrowing
`apply`	✅ Yes	Array of args: `[arg1, arg2]`	❌ No	Useful when args are in an array
`bind`	❌ No	Separate args	✅ Yes	Delayed execution with fixed `this`

🧪 Example: Math `max` using `apply`

```
const nums = [1, 5, 3, 9, 2];

console.log(Math.max.apply(null, nums)); // 9
```

> You can't use Math.max(nums) directly — so apply() helps spread the array.

In modern JavaScript, you can also do:

```
console.log(Math.max(...nums)); // 9
```

📝 Summary

- `call()` → calls the function **with arguments**, setting `this`.
- `apply()` → same as `call()` but accepts **arguments as an array**.
- `bind()` → returns a **new function** with `this` permanently set.

Let me know if you'd like to try examples in the browser or move on to **Pure Functions and Side Effects** next!