JS JavaScript TUTORIAL

JavaScript apply() method

Summary: in this tutorial, you'll learn about the JavaScript `apply()` method of the `Function` type and how to use it effectively.

Introduction to the JavaScript apply() method

The `Function.prototype.apply()` method allows you to call a function with a given `this` value and arguments provided as an array. Here is the syntax of the `apply()` method:

```
fn.apply(thisArg, [args]);
```

The `apply()` method accepts two arguments:

- The `thisArg` is the value of this provided for the call to the function `fn`.
- The `args` argument is an array that specifies the arguments of the function `fn`. Since the ES5, the `args` argument can be an array-like object or an array object.

The `apply()` method is similar to the `call()` method except that it takes the arguments of the function as an array instead of the individual arguments.

JavaScript apply() method examples

Let's take some examples of using the `apply()` method.

1) Simple JavaScript apply() method example

Suppose that you have a `person` object:

```
const person = {
    firstName: 'John',
    lastName: 'Doe'
}
```

...and a function named `greet()` as follows:

```
function greet(greeting, message) {
    return `${greeting} ${this.firstName}. ${message}`;
}
```

The `greet()` function accepts two parameters: `greeting` and `message`. Inside the `greet()` function, we reference an object that has the `firstName` property.

The following example shows how to use the `apply()` method to call the `greet()` function with the `this` set to the `person` object:

```
let result = greet.apply(person, ['Hello', 'How are you?']);

console.log(result);
```

Output:

```
Hello John. How are you?
```

In this example, we set the `this` value inside the function to the `person` object. The arguments of the `greet()` function was passed into the `apply()` method as an array.

The `apply()` method invoked the `greet()` function with the `this` value set to the `person` object and arguments as an array `['Hello', 'How are you?']`.

If you use the `call()` method, you need to pass the arguments of the `greet()` function separately as follows:

```
let result = greet.call(person, Hello', 'How are you?');
```

2) Function borrowing

The `apply()` method allows an object to borrow the method of another object without duplicating the code.

Suppose that you have the following `computer` object:

```
const computer = {
    name: 'MacBook',
    isOn: false,
    turnOn() {
        this.isOn = true;
        return `The ${this.name} is On`;
    },
    turnOff() {
        this.isOn = false;
        return `The ${this.name} is Off`;
    }
};
```

... and the following `server` object:

```
const server = {
    name: 'Dell PowerEdge T30',
    isOn: false
};
```

The `server` object doesn't have the `turnOn()` and `turnOff()` methods.

To execute the `turnOn()` method of the `computer` object on the `server` object, you can use the `apply()` method as follows:

```
let result = computer.turnOn.apply(server);

console.log(result);
```

Output:

```
The Dell PowerEdge T30 is On
```

In this example, the `server` object borrows the `turnOn()` method of the `computer` object.

Similarly, you can call the `turnOff()` method of the computer object on the server object:

```
let result = computer.turnOff.apply(server);
console.log(result);
```

Output:

```
The Dell PowerEdge T30 is Off
```

3) Using the apply() method to append an array to another

The `apply()` method allows you to append elements of an array to another:

```
let arr = [1, 2, 3];
let numbers = [4, 5, 6];

arr.push.apply(arr, numbers);

console.log(arr);
```

In this example, the `apply()` method modifies the original array `arr`. Note that the `Array.prototype.concat()` method also provides the same result except that it returns the new array instead of modifying the original array.

Summary

- The `apply()` method invokes a function with a given `this` value and arguments provided as an array.
- The `apply()` method is similar to the `call()` method excepts that it accepts the arguments of the function as an array instead of individual arguments.