Uddesh
Posted on May 30, 2021

Creating your own bind() (Polyfill of bind)

#beginners #javascript #node #tutorial

Strange JS (4 Part Series)

1. Introduction to Hoisting in JavaScript.
2. Coercion in JavaScript.
3. Spread Operator(...) in JavaScript.
4. **Creating your own bind() (Polyfill of bind)**

So, what is polyfill?

Polyfill is a fallback for a method that is not supported by the browser by default. You can find browser support for any function or method on the `mdn` website.

Now, what is `bind()` ?

According to `mdn`

> The `bind()` method creates a new function that, when called, has its this keyword

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

This definition sounds really fancy, but what does that mean?

In simple words,

> The `bind()` takes an object as an argument and returns a new function whose `this` refers to the object we passed as an argument.

Now, we got the idea about `polyfill` and `bind()`. So, let's try to implement it.

1). let's create an object that we are going to use as a call site.

```
let obj = {
  name: 'Jack',
};
```

2). Create a `function` that we need to bind with the object.

```
let myFunc = function () {
  console.log(`${this.name}`);
};
```

If you call this function now, It will print `undefined`

3). Add your bind() to the function prototype.

```
Function.prototype.myBind = function (obj) {
  let func = this;
  return function () {
    func.apply(obj);
  };
};
```

Putting it all together.

```
let obj = {
  name: 'Jack',
};

let myFunc = function () {
  console.log(`${this.name}`);
};
```

```
};

Function.prototype.myBind = function (obj) {
  let func = this;
  return function () {
    func.apply(obj);
  };
};

let newFunc = myFunc.myBind(obj)
newFunc() // Jack
```

This is the basic implementation of `bind()`, But it has few edge cases. Let's say you want to pass some arguments in `myBind()`. How will you do that? Currently, we are not accepting arguments other than the object itself. Let's implement this functionality.

The problem

```
let obj = {
  name: 'Jack',
};

let myFunc = function (id) {
  console.log(`${this.name}, ${id}`);  // id will be undefined
};

Function.prototype.myBind = function (obj) {
  let func = this;
  return function () {
    func.apply(obj);
  };
};

let newFunc = myFunc.myBind(obj, 'a_random_id')
newFunc() // Jack, undefined
```

We are trying to pass `id` in `myBind` but not able to access it.

The solution

```
let obj = {
  name: 'Jack',
};

let myFunc = function (id) {
  console.log(`${this.name}, ${id}`);  // id will be undefined
};
```

```
// Accepting any number of arguments passed to myBind
Function.prototype.myBind = function (obj, ...args) {
  let func = this;
  return function () {
    func.apply(obj, [...args]);
  };
};

let newFunc = myFunc.myBind(obj, 'a_random_id')
newFunc() // Jack, a_random_id
```

Now we solved an edge case. However, there is one more improvement we can make. What if we want to pass arguments to `newFunc()`?

The problem

```
let obj = {
  name: 'Jack',
};

let myFunc = function (id, city) {
  console.log(`${this.name}, ${id}, ${city}`);  // id will be undefined
};

// Accepting any number of arguments passed to myBind
Function.prototype.myBind = function (obj, ...args) {
  let func = this;
  return function () {
    func.apply(obj, [...args]);
  };
};

let newFunc = myFunc.myBind(obj, 'a_random_id')
newFunc('New York') // Jack, a_random_id, undefined
```

We are passing 'New York' but not able to access it inside `myFunc`

The solution

```
let obj = {
  name: 'Jack',
};

let myFunc = function (id, city) {
  console.log(`${this.name}, ${id}, ${city}`);  // id will be undefined
};
```

```
// Accepting any number of arguments passed to myBind
Function.prototype.myBind = function (obj, ...args) {
  let func = this;
  // Accepting arguments passed to newFunc
  return function (...newArgs) {
    func.apply(obj, [...args, ...newArgs]);
  };
};

let newFunc = myFunc.myBind(obj, 'a_random_id')
newFunc('New York') // Jack, a_random_id, New York
```

Now I think we have covered almost all the cases. But, still, if there is anything left, Let me know in the comment section.

Time to celebrate. You have implemented your own `bind()`.

I hope this will be helpful for you.

[redacted]

Strange JS (4 Part Series)

1 Introduction to Hoisting in JavaScript.

2 Coercion in JavaScript.

3 Spread Operator(...) in JavaScript.

4 **Creating your own bind() (Polyfill of bind)**

Discussion (11)

Muhammad Hasnain • May 31 '21

Great article, although, I don't think there is any need for this. It is 2021 and support for bind came between 2009 and 2011. Almost all operating systems for all types of devices comes with advanced built-in browsers that supports at least ES6 and almost all of them install and use modern browsers.

Uddesh • May 31 '21

Totally agree, But first of all it helps us to understand how bind() works behind the scene. Apart from that i was asked to implement polyfill of bind() in few interviews as well.

David Matos salvador • Jan 17

this example is to understand how the black box works

Adam Crockett • May 30 '21

I haven't used bind very often since arrow functions

Uddesh • May 30 '21

Arrow function are really helpful for adopting 'this' from their parent and in most of

Arrow function are really helpful for adopting 'this' from their parent and in most of the use cases we don't need 'bind()', But what if you want 'this' to refer to a particular

object then you need to use 'bind()'. I hope you got my point.

Adam Crockett • May 31 '21

To be honest, I only use bind for it's less well known second Param. I avoid using this and prefer pure functions, I have very few reasons to make this do anything, bimut that's just me ☺

prashanth • Jan 30

This is a pretty cool post.

I would like to make a small alteration in your code.
We can eliminate the need for storing 'this' in the parent scope and using it at the time by invocation by making use of arrow function.

```
Function.prototype.myBind = function (obj, ...args) {
  return (...newArgs) => this.apply(obj, [...args, ...newArgs]);
};
```

Hope this helps!!

Aleksei Berezkin • May 31 '21

Implementing `bind` with `apply` is kind of cheating 😉

Neeraj Sewani • Jan 27

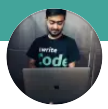
Nice article Uddesh.

Abhishek Jain • May 30 '21

Good one. 🌟🌟

Uddesh • May 30 '21

Thanks. 🙂



Uddesh

Full Stack Web Developer, JavaScript lover ❤️, Computer Science Student. Enthusiastic about the latest technologies.

LOCATION
India

EDUCATION
Engineering

JOINED
Sep 15, 2018

More from Uddesh

Zustand: Yet another state management library for react
#javascript #react #beginners #tutorial

The minimal server setup guide.
#security #aws #googlecloud #beginners

Getting familiar with Objects.
#javascript #beginners #node #writing