Understanding bind and bindAll in Backbone.js

August 18, 2011 - Neeraj Singh

Backbone.js users use bind and bindAll methods provide by underscore.js a lot. In this blog I am going to discuss why these methods are needed and how it all works.

It all starts with apply

Function `bindAll` internally uses `bind` . And `bind` internally uses `apply` . So it is important to understand what `apply` does.

```
var func = function beautiful(){
  alert(this + ' is beautiful');
};
func();
```

If I execute above code then I get `[object window] is beautiful` . I am getting that message because when function is invoked then `this` is `window` , the default global object.

In order to change the value of `this` we can make use of method `apply` as given below.

```
var func = function beautiful(){
  alert(this + ' is beautiful');
};
func.apply('Internet');
```

In the above case the alert message will be `Internet is beautiful` . Similarly following code will produce `Beach is beautiful` .

```
var func = function beautiful(){
  alert(this + ' is beautiful');
};
func.apply('Beach'); //Beach is beautiful
```

In short, `apply` lets us control the value of `this` when the function is invoked.

Why bind is needed

In order to understand why `bind` method is needed first let's look at following example.

```
function Developer(skill) {
  this.skill = skill;
  this.says = function(){
    alert(this.skill + ' rocks!');
  }
}
var john = new Developer('Ruby');
john.says(); //Ruby rocks!
```

Above example is pretty straight forward. `john` is an instance of `Developer` and when `says` function is invoked then we get the right alert message.

Notice that when we invoked `says` we invoked like this `john.says()` . If we just want to get hold of the function that is returned by `says` then we need to do `john.says` . So the above code could be broken down to following code.

```
  function Developer(skill) {
    this.skill = skill;
    this.says = function(){
      alert(this.skill + ' rocks!');
    }
  }
  var john = new Developer('Ruby');
  var func = john.says;
  func();// undefined rocks!
```

Above code is similar to the code above it. All we have done is to store the function in a variable called `func` . If we invoke this function then we should get the alert message we expected. However if we run this code then the alert message will be `undefined rocks!` .

We are getting `undefined rocks!` because in this case `func` is being invoked in the global context. `this` is pointing to global object called `window` when the function is executed. And `window` does not have any attribute called `skill` . Hence the output of `this.skill` is `undefined` .

Earlier we saw that using `apply` we can fix the problem arising out of `this` . So lets try to use `apply` to fix it.

```
function Developer(skill) {
  this.skill = skill;
  this.says = function(){
    alert(this.skill + ' rocks!');
  }
}
var john = new Developer('Ruby');
var func = john.says;
func.apply(john);
```

Above code fixes our problem. This time the alert message we got was `Ruby rocks!` . However there is an issue and it is a big one.

In JavaScript world functions are first class citizens. The reason why we create function is so that we can easily pass it around. In the above case we created a function called `func` . However along with the function `func` now we need to keep passing `john` around. That is not a good thing. Secondly the responsibility of rightly invoking this function has been shifted from the function creator to the function consumer. That's not a good API.

We should try to create functions which can easily be called by the consumers of the function. This is where `bind` comes in.

How bind solves the problem

First lets see how using `bind` solves the problem.

```
function Developer(skill) {
  this.skill = skill;
  this.says = function(){
    alert(this.skill + ' rocks!');
  }
}
var john = new Developer('Ruby');
var func = _.bind(john.says, john);
func();// Ruby rocks!
```

To solve the problem regarding `this` issue we need a function that is already mapped to `john` so that we do not need to keep carrying `john` around. That's precisly what `bind` does. It returns a new function and this new function has `this` bound to the value that we provide.

Here is a snippet of code from `bind` method

```
  return function() {
    return func.apply(obj, args.concat(slice.call(arguments)));
  };
```

As you can see `bind` internally uses `apply` to set `this` to the second parameter we passed while invoking `bind` .

Notice that `bind` does not change existing function. It returns a new function and that new function should be used.

How bindAll solves the problem

Instead of `bind` we can also use `bindAll` . Here is solution with `bindAll` .

```
function Developer(skill) {
  this.skill = skill;
  this.says = function(){
    alert(this.skill + ' rocks!');
  }
}
var john = new Developer('Ruby');
_.bindAll(john, 'says');
var func = john.says;
func(); //Ruby rocks!
```

Above code is similar to `bind` solution but there are some big differences.

The first big difference is that we do not have to worry about the returned value of `bindAll` . In case of `bind` we must use the returned function. In `bindAll` we do not have to worry about the returned value but it comes with a price. `bindAll` actually mutates the function. What does that mean.

See `john` object has an attribute called `says` which returns a function . `bindAll` goes and changes the attribute `says` so that when it returns a function, that function is already bound to `john` .

Here is a snippet of code from `bindAll` method.

```
function(f) { obj[f] = _.bind(obj[f], obj); }
```

Notice that `bindAll` internally calls `bind` and it overrides the existing attribute with the function returned by `bind` .

The other difference between `bind` and `bindAll` is that in `bind` first paramter is a function `john.says` and the second parameter is the value of this `john` . In `bindAll` first paramter is value of this `john` and the second parameter is not a function but the attribute name.

Things to watch out for

While developing a Backbone.js application someone had code like this

```
window.ProductView = Backbone.View.extend({
  initialize: function() {
    _.bind(this.render, this);
    this.model.bind('change', this.render);
  }
});
```

Above code will not work because the returned value of `bind` is not being used. The correct usage will be

```
window.ProductView = Backbone.View.extend({
  initialize: function() {
    this.model.bind('change', _.bind(this.render, this));
  }
});
```

Or you can use `bindAll` as given below.

```
window.ProductView = Backbone.View.extend({
  initialize: function() {
    _.bindAll(this, this.render);
    this.model.bind('change', this.render);
  }
});
```

Recommended videos

If you like this blog then most likely you will also like the four videos series on “Understanding this in JavaScript” at Learn JavaScript .

Neeraj Singh on August 18, 2011

Category: Javascript

Share this article

Subscribe to our monthly newsletter.

Enter your email here

Subscribe

Follow @BigBinary

Comments for this thread are now closed.

×

47 Comments BigBinary Blog

1 Login

Recommend 33 Share

Sort by Best

Tim • 4 years ago

Thanks for this, I now finally understand what bindAll does :)

13 • Share ›

Matt Styles • 4 years ago

Good explanation, really well written

10 • Share ›

TheHungryCoder • 3 years ago

loved this article.

2 • Share ›

moo • 3 years ago

very clear and concise. nice work!

2 • Share ›

damfa • 3 years ago

Thanks !

2 • Share ›

meandre • 2 years ago

Everybody knows `bind` is an anti-pattern that may lead to memory leaks. Oh I might have said everybody who knows how to program.
`This` is just an argument and there is nothing good in pairing functions with data. Use the `EventHandler` interface (which the guys from W3C have invented for you) and you won't lose your context.

2 • Share ›

gillbates • 2 years ago

in the final example, why is it _.bindAll(this, this.render); and not _.bindAll(this, 'render'); ?

1 • Share ›

Sam • 3 years ago

Really excellent explanation!

1 • Share ›

Adam Musial-Bright • 3 years ago

Amazing explanation - thank you Neeraj

1 • Share ›

StephanWallentin • 4 years ago

This is a great article. And the code example with binding model changes to the view's render method was an cool pattern. Job well done!

1 • Share ›

Bhargav • a year ago

Thanks Neeraj!

• Share ›

Randy • 2 years ago

best article i've found that explains this..thanks!

• Share ›

vijay krishna • 2 years ago

good explanation thanks :)

• Share ›

Phil • 2 years ago

Great explanation.

• Share ›

momoprobs • 2 years ago

This is awesome, even two years later. Thanks!

• Share ›

Barak • 2 years ago

Thanks
Very helpful

• Share ›

Danush • 2 years ago
Thank you very much!! Very good explanation!!

• Share ›

Mikael • 2 years ago
Thanks, Neeraj! This is exactly the explanation that I needed.

• Share ›

Reza • 2 years ago
Thanks a lot, very useful

• Share ›

Johanes Mai • 2 years ago
Good explanation and clearly. Thankju

• Share ›

kalyanrajiv • 3 years ago
Very Well Explained. Thanks :)

• Share ›

Sudheer A • 3 years ago
Awesome article !!!!!. Anyone didn't get idea about bind and bindAll after reading this article. please Quit from programming.

• Share ›

Anand Itagi • 3 years ago
Thank you

• Share ›

rockneverdies • 3 years ago

Great work. Now I know the difference between the 2.

• Share ›

Pranay Soni • 3 years ago

thanks! loved this article

• Share ›

Andrew Martin • 3 years ago

THANK YOU.

• Share ›

xReWxpilau • 3 years ago

Great article. I'll be following you.

• Share ›

Emile Baizel • 3 years ago

Great write up. Thanks for taking the time to explain it so thoroughly.

• Share ›

Ryan W. • 3 years ago

Excellent tutorial. Thanks!

• Share ›

Alexis Bellido • 3 years ago

I was reading Developing Backbone.js Applications and the _.bind method was mentioned but it wasn't that clear. Your explanation complements that part of the book and now I get it. Thanks!

• Share ›

Erich Schulz • 3 years ago

Extrend? Typo?

• Share ›

Guest • 3 years ago
loved this article
• Share ›

Victor Lam • 3 years ago
Thanks! Details explanation from javascript to underscore to backbone...very helpful :)
• Share ›

Jayant Bhat • 3 years ago
Very clean,neat and awesomest explanation. Thanks a lot.. :) It helped me a lot ..:)
• Share ›

deid4r4 • 3 years ago
Thank's for your remarkable explanation..... :)
• Share ›

Joe • 3 years ago
Excellent article, well written - thanks!
• Share ›

Brian • 3 years ago
Fantastic explanation. I am learning Backbone and was curious as to the bind call and this explained it incredibly thoroughly! Great information.
• Share ›

Nico Beta • 3 years ago
Thanks for explaining this!
• Share ›

Rodrigo Dominguez • 4 years ago

Great article, until now I didn't understand how bindAll worked at all...

• Share ›

Ivan Juarez • 4 years ago

Nice article from the beginning to the end, thanks!!

• Share ›

RS • 4 years ago

Really Help full. But i have one question if i call "this.curretnobj.bind("logout:success", this.logoutSuccessful)" what that means

• Share ›

Attila SATAN • 4 years ago

This is a wonderful article. It really clears the clouds.

• Share ›

Erwann Mest • 4 years ago

Thanks for your explanation. Regards.

• Share ›

Ravi Gadhia • 4 years ago

Thanks, good explanation with example :)

• Share ›

Omer • 4 years ago

Thanks dear . very helpful

• Share ›

ffleandro • 4 years ago

Thank you very much. I was struggling with this when using backbone.

• Share ›

Loyiso • 4 years ago

Very insightful. Thank you very much.

• Share ›

Subscribe Add Disqus to your site Add Disqus Add Privacy

« Previous Next »

Home

Videos

Blog

About

Ebooks

Contact

786 - 763 - 1059

hello@BigBinary.com

Twitter

GitHub