Calling a class function in forEach: how Javascript handles “this” keyword

Asked 4 years, 6 months ago Active 11 months ago Viewed 2k times

4

2

I'm new to Javascript and just want to make sure I'm understanding how it handles the `this` keyword, since... well, it seems like it's pretty messy. I've checked out similar questions on StackOverflow and want to make sure I'm not moving forward with the wrong idea.

So I'm defining a class like so, and want to process every point received in the constructor.

```
function Curve(ptList) {
  this.pts = [];

  if(ptList.length > 2) {
    // I want to call "addPoint" for every item in ptList right here
  }
}

Curve.prototype.addPoint = function(p) {
  this.pts.push(p);
  /* some more processing here */
}
```

So, initially I thought I could just do:

```
ptList.forEach(this.addPoint);
```

but I can't, because that is simply passing a pointer to the prototype function, which means `this` in `addPoint` refers to the global object.

Then I tried:

```
ptList.forEach(function(p) { this.addPoint(p); });
```

but I can't, because `this` refers to the global scope as soon as I enter that internal function (it no longer refers to the `Curve` object being constructed), so `addPoint` is undefined.

The way to get around that is to make a variable that refers to `this` in a scope I can still talk to in the subfunction:

```
var _this = this;
ptList.forEach(function(p) { _this.addPoint(p); });
```

And this finally works (but looks really weird to someone without JS experience). But then I found out about the `bind` function, which lets me not define a trivial function wrapper:

```
ptList.forEach(this.addPoint.bind(this));
```

and it seems like, finally, this is the best and most concise way to do it, even thought it looks silly. Without the `bind` function, `addPoint` has no understanding of which object it was called on, and without the first `this` , I can't even find my `addPoint` function.

Is there a better way to do this seemingly simple thing, or is that last line of code the recommended way?

javascript foreach this bind

asked Jan 22 '15 at 0:06
XenoScholar
82 1 10

1 For complete illumination on this in Javascript, check out this free book: github.com/getify/You-Dont-Know-JS/tree/master/… It will answer all of your questions. – Tad Donaghe Jan 22 '15 at 0:09

2 This question is asked at least once a day. The solution is really understanding how `this` works, then you'll have an aha moment and realize why it is not silly. But there's another way `.forEach(this.addPoint, this)` – elclanrs Jan 22 '15 at 0:09

Depends on your viewpoint. For instance, in knockoutjs it's a common idiom to capture a copy of `this` with the line `var self=this` . Lately, `.bind` is more commonplace. – spender Jan 22 '15 at 0:10

If `forEach` did not provide the convenience then yes, one of your solutions would be the way to go. – Jon Jan 22 '15 at 0:10

2 Answers

7

The [`forEach()`]() method and other similar array methods like `map()` and `filter()` provide an optional parameter called `thisArg` specifically for this purpose.

This serves as the `this` "argument" that is provided to the function when it is called:

```
ptList.forEach(this.addPoint, this);
```

When such a parameter isn't available, then choosing between using a closure variable (the variable name `self` is often used for this) or `.bind()` is mostly a matter of preference. Note that another option is to define functions inside your constructor and capture everything in a closure:

```
function Curve(ptList) {
  var pts = [];
  function addPoint(p) {
      pts.push(p);
  }

  if(ptList.length > 2) {
    ptList.forEach(addPoint);
  }

  this.addPoint = addPoint;
}
```

This requires a bit more memory than putting the functions on a prototype since every instance gets its own copy (and also rules out some of the things a prototype allows you to do), but Douglas Crockford (the inventor of JSON and JSLint) recently said that he doesn't even use `this` anymore, with the rationale that memory is cheap, but CPU cycles are not (traversing the prototype chain requires a fair amount of processing).

edited Aug 17 '18 at 14:05

answered Jan 22 '15 at 0:10
JLRishe
79.2k 11 84 122

2 IMO `this` is a dangerous thing in JS. Given that it can be very difficult to identify, at-a-glance, what `this` is, I think Crockford has got it right. – spender Jan 22 '15 at 0:34

1 @spender I would pretty much agree with that. Lately I've been trying to use `this` as little as possible. The arrow functions in ES6 seem like an attempt to remedy `this`'s weird behavior, but I think they're just going to cause a whole new wave of confusion. – JLRishe Jan 22 '15 at 0:39

Thanks for your comments, makes me think that I should probably redesign my (as of now, short) program to do the closure idea, rather than make a mess of it with `this`. – XenoScholar Jan 22 '15 at 0:59

The way it's normally done is `var that = this`. Inheritance in JavaScript seems weird at first, but you should stick with what's commonly used. Doing something like `ptList.forEach(this.addPoint.bind(this))` is going to confuse the next person to use your code.

0

[What does 'var that = this;' mean in JavaScript?]

edited May 23 '17 at 10:32
Community ♦
1 1

answered Jan 22 '15 at 0:16
apostl3pol
365 2 13

1 I think that not using a very clearly defined language feature because it might be a barrier to someone who isn't familiar with it is misguided. To me the `.bind` is crystal-clear and reduces very clear intent to a single point in the code. Closing over a copy defined in another place diffuses the intent over 2 locations, so increases the complexity of the code. IMO `.bind` will lead to code that is more easily identifiable as correct, which is far more desirable. – spender Jan 22 '15 at 0:38