article

Demystifying JavaScript Closures, Callbacks and IIFEs

kensho Feb '15 #1

Originally published at: http://www.sitepoint.com/demystifying-javascript-closures-callbacks-iifes/

We've already taken a close look at **variable scope and hoisting**, so today we'll finish our exploration by examining three o the most important and heavily-used concepts in modern JavaScript development — closures, callbacks and IIFEs.

Closures

In JavaScript, a [closure](**http://en.wikipedia.org/wiki/Closure_(computer_programming)**) is any function that keeps reference to variables from its parent's scope *even after the parent has returned*.

This means practically any function can be considered a closure, because, as we learned in the **variable scope** section from the first part of this tutorial, a function can refer to, or have access to –

- any variables and parameters in its own function scope
- any variables and parameters of outer (parent) functions
- any variables from the global scope.

So, chances are you've already used closures without even knowing it. But our aim is not just to use them – it is to understand them. If we don't understand how they work, we can't use them *properly*. For that reason, we are going to split the above closure definition into three easy-to-comprehend points.

Point 1: *You can refer to variables defined outside of the current function.*

```
function setLocation(city) {
  var country = "France";

  function printLocation() {
    console.log("You are in " + city + ", " + country);
  }

  printLocation();
}

setLocation ("Paris");  // output: You are in Paris, France
```

Try out the example in JS Bin

In this code example, the `printLocation()` function refers to the `country` variable and the `city` parameter of the enclosing (parent) `setLocation()` function. And the result is that, when `setLocation()` is called, `printLocation()` successfully uses the variables and parameters of the former to output "You are in Paris, France".

Continue reading this article on SitePoint

Show Full Post...

On Our Radar: Closures, Copyright and the Best Apps of 2014

felgall Feb '15 #2
Former Member of the Month

> kensho:

> In JavaScript, a [closure] is any function that keeps reference to variables from its parent's scope even after the parent has returned.

That statement makes no sense any you appear to have a completely wrong idea about closures - since none of your supposed closure examples actually use a closure.

A closure occurs when a variable or function defined inide of the inner function's scope continues to be accessible to the outer function even after the inner function has returned. Just about the exact opposite of what you said.

See **https://developer.mozilla.org/en-US/docs/Web/JavaScript/Closures** where tyour examples match the first example that illustrates lexical scope (not closures) The next example on that page illustrates the creation of a closure by returning the function from the inn er scope without executing it and then executing it later after the scope it was created in has closed. There are further example further down that page.

To create a closure you must return something that references elements with inner scope.

Jeff_Mott **Feb '15 #3**
SitePoint Wizard

> felgall:
>
> since none of your supposed closure examples actually use a closure.

> felgall:
>
> See **https://developer.mozilla.org/en-US/docs/Web/JavaScript/Closures** where tyour examples match the first example that illustrates lexical scope (not closures) The next example on that page illustrates the creation of a closure by returning the function from the inn er scope without executing it and then executing it later after the scope it was created in has closed. There are further example further down that page.

Felgall, the MDN example you're referencing is virtually identical to the article's "Point 2" example.

---

felgall **Feb '15 #4**
Former Member of the Month

> Jeff_Mott:

> Felgall, the MDN example you're referencing is virtually identical to the article's "Point 2" example.

Okay - I was misled by the completely incorrect description of what a closure actually is and missed that after the first supposed closure example that didn't include a closure that the subsequent examples started to use one. A very confusing article about closures.

There are basically three ways to create a closure

```
example1 = function(x) {
var myvar1 = x;
return function() {return myvar1;};
}

var aa;
example2 = function(y) {
var myvar2 = y;
aa = function() {return myvar2;};
}

example3 = function(z) {
var bb, myvar3;
bb = function() {return myvar3;};
myvar3 = z;
return bb;
}
```

Jeff_Mott **Feb '15 #5**
SitePoint Wizard

> felgall:
>
> A closure occurs when a variable or function defined inide of the inner function's scope continues to be accessible to the outer function even after the inner function has returned. Just about the exact opposite of what you said.

Actually the article got it right. A closure is an inner function that has access to an outer function's variables. And in fact the MDN article you referenced says the same.

Jeff_Mott Feb '15 #18
SitePoint Wizard

> daz4126:
>
> I still think that the first example in the article and the example I gave are important in building up to demonstrating closures as they demonstrate the scope of a function...

This part sounds fine. Yes, the first example in the article and the example you gave are a fine demonstration of *scope*.

> daz4126:
>
> ...and the fact that by using the return statement you can access variables created inside another function.

But this isn't quite right. In this discussion, remember it's important to distinguish *values* and *variables*. A return statement passes a *value*. You're not creating a closure just by returning some value.

daz4126 Feb '15 #19
SitePoint Member

> Jeff_Mott:
>
> But this isn't quite right. In this discussion, remember it's important to distinguish values and variables. A return statement passes a value. You're not creating a closure just by returning some value.

Yes I agree with you on this - it's important that if you just return a variable then it's the value of the variable that is returned and not the variable itself, which is what happens if it is returned inside a function. My example wasn't a great example, sorry if it confused anybody.

DAZ

CLOSED MAY 25, '15

This topic was automatically closed 91 days after the last reply. New replies are no longer allowed.