Closures in JavaScript

16 Jul 2009

A closure, in JavaScript, can simply be described as a retained scope; at least, this is how I think of it. The benefit of a closure is in the fact that it retains the scope (the “scope chain”) of the outer (or “parent”) execution context. This behaviour can be used in a number of different ways and has become a useful remedy for quite a few JavaScript gotchas; one of the most common being the “looping problem”.

The looping problem occurs when you create a function within a loop and expect the current value of a variable to retain itself within that new function even though it changes in the loops context before your new function is called. Here’s en example:

```
var myElements = [ /* DOM Collection */ ];

for (var i = 0; i < 100; ++i) {
    myElements[i].onclick = function() {
        alert( 'You clicked on: ' + i );
    };
}
```

By the time `myElements[0].onclick` is triggered the value of `i` will be 99, meaning that whatever element you click on you’ll always get the message, “You clicked on: 99”.

This problem can be solved by creating a function and calling it on each iteration of the loop, while passing `i`; calling the function will form a brand new execution context where the value of `i` is retained and can be used in any way within that context, including within the returned function.

```
function getHandler(n) {
    return function() {
        alert( 'You clicked on: ' + n );
    };
}

for (var i = 0; i < 100; ++i) {
    myElements[i].onclick = getHandler(i);
}
```

Now it works perfectly! A common shortcut used is to create and call the function at the same time, using what is known as a “self invoking anonymous function”:

```
for (var i = 0; i < 100; ++i) {
    myElements[i].onclick = (function(n) {
        return function() {
            alert( 'You clicked on: ' + n );
        };
    })(i);
}
```

This way is more common since it’s more concise and heck, it looks cooler! But, the former technique is probably faster since you’re only creating the `getHandler` function once; not on every iteration.

Note: You don’t have to use a different identifier (`n`) when passing `i` to the function but, generally, it’s good practice not to overwrite variables of an inherited scope.

Another common usage of the closure is Yahoo’s (or Crockfords?) Module Pattern:

```
var someCoolModule = (function() {

    var privateVariable = 12345;

    return {
        doSomething: function() {
            alert( privateVariable );
        },
        doSomethingElse: function() {
            // ...
        }
    }

})();

/* The first set of parentheses (around "function(){}") isn't required but is
    used to make it obvious that the function is immediately invoked, thus
    making it obvious that the expression does not necessarily return that
    function; but instead the return value of that function */
```

By using the module pattern we can create both private *and* public variables/methods. Since the returned functions inherit the scope of the parent function they have access to all variables and arguments within that context.

The closure is also useful in a situation where a function uses the same resource on every call yet *also* creates that resource on every call, thus making it inefficient, e.g.

```
function doSomething() {
    var regex = /[^[](.+?).$/;
    /* Do other stuff ... */
}
```

Since `regex` doesn't change between function calls we can put it elsewhere; and what better place to hide it than in an unexposed private scope:

```
var doSomething = (function(){
    var regex = /[^[](.+?).$/;
    return function() {
        /* Do someting with "regex"... */
    };
})();
```

Have any other useful applications of the closure? – please share…

Thanks for reading! Please share your thoughts with me on Twitter. Have a great day!

So far there's been 16 Responses to "Closures in JavaScript"

1. *Mark* July 16th, 2009 at 11:54 pm

   I think of closures as unique little environments. If I want to use a variable but don't want it to conflict with something else, I wrap the code in a self-invoking anonymous function.

   Similarly, when I use jQuery and other frameworks, I use the noConflict() for interoperability. I then wrap my jQuery in a closure and pass in jQuery for the `$' argument so that my code in the closure can still use the $:

   ```
   (function($){
   ```

```
  // use $ in here
})(jQuery);
```

2. *John.H* [July 17th, 2009 at 6:10 am]

   Similar to Yahoo's module pattern, I use closures to construct a javascript class.

```
function People(){
        //private properties
        var _name = '';
        var _age  = 0;

        //private method
        function doSomeThing(){}


        //public method
        this.setName = function(name){
                doSomeThing();
                _name = name;
        };

        this.setAge = function(age){
                _age = age
        };
}
```

3. [*Dusan*] [July 17th, 2009 at 9:29 am]

   I am really puzzled. Please do not think I am being rude etc …
   But I am simply wondering: Why are people reading this blog?
   Why am I reading a blog of an 18 year old "scripter" ?
   And on top of that, all is about one poorly made scripting language … Amazing.

   All I can think of is "packaging over content".
   Well written posts, and well researched too.
   What are they about is less important it seems ? After all I am here voluntarily ;o)

   I also do use (a lot) javascript and after 15+ years of "real" languages porffesional usage, I came to a conclusion that javascript has its appeal *only* inside HTML. It allows for very quick development of "awesome stuff", although not always relevant. And of course browser made this combination realy ubiquitous.

Keep up a good blog …

Regards: Dusan (London,UK)

4. *Matt* July 17th, 2009 at 5:41 pm

   Hi James,

   Can u explain, step by step, why "i" will alert 99 every time in your first example? I know it works like that, but I still don't get it.

5. *Georg* July 17th, 2009 at 11:51 pm

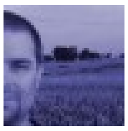
   @Mark: If you encapsulate each of your script files with this technique you get also the bonus of higher script performance (yes really) in addition to the fact that you don't pollute the global namespace.

   Closures are making the Javascript language really sexy. (And yes, I know programming in other higher languages.)

6. *Dean Edwards* July 18th, 2009 at 12:34 am

   @Dusan, I think you are right. I would never write a blog telling people how to do things at the age of 18. What do I know at that age? It's symptomatic of the web in general, lots of young know-alls telling you how to do things.

   Having said that, I can find no technical fault with this article. It's well-written and explains things with good examples.

7.  *Bjoern Wibben* July 18th, 2009 at 12:15 pm

   Hi James,

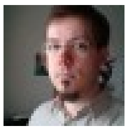
   your blog is great, keep up the good work.

   I'm using closures for iteration (simplified example):

```
var myarr = ["a", "b", "c"];

function iterator(arr) {
        var idx = 0;
        return function () {
                return arr[idx++];
        }
}

var getNext = iterator(myarr);

getNext(); // a
getNext(); // b
getNext(); // c
```

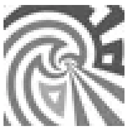
Best Regards,

Björn

8.  *[Jani Hartikainen](#)* [July 20th, 2009 at 12:34 am](#)

I’m not 100% sure but the looping example with the anonymous function may end up leaking memory. The array has references to the elements, and the closure is going to keep a reference to the array, thus creating a circular reference when you assign the function as a handler in an element.

@Dusan why are you reading this blog if you think it’s pointless? 😛

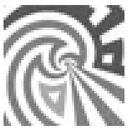
9.  *zorg* [July 20th, 2009 at 1:05 pm](#)

@Dusan

> I also do use (a lot) javascript and after 15+ years of “real” languages porffesional usage, I came to a
> conclusion that javascript has its appeal *only* inside HTML. It allows for very quick development of
> “awesome stuff”, although not always relevant. And of course browser made this combination realy
> ubiquitous.

Seriously, don’t people ever tire of that “real professional languages” rant ?
Javascript is a cool mix of Scheme and Self (two languages way cooler than most “real” languages.
Javascript is the web lingua franca.
Javascript implementation have upped their games tremendously in the past year, V8 is faster than PHP, CPython, MRI, etc…
Javscript is indeed making inroads outside the browser

(http://en.wikipedia.org/wiki/Server-side_JavaScript), coding your whole web app in one language does makes sense, goodbye impedance mismatch

10.  *zorg* July 20th, 2009 at 2:17 pm

@James

> "This behaviour can be used in a number of different ways and has become a useful remedy for quite a few
> JavaScript gotchas; one of the most common being the "looping problem".

the Javascript gotcha you mention happens the onclick anonymous function closes over a binding to a mutable variable, it is non intuitive but not incorrect! Intuitively, one expects a new binding to a fresh variable at every iteration, that's what would happen in a language where immutability is the default.

See http://math.andrej.com/2009/04/09/pythons-lambda-is-broken/
and http://lambda-the-ultimate.org/node/2648

11. *Louis* July 21st, 2009 at 3:25 pm

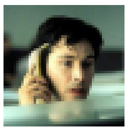
Nice article, James.

I just wanted to point out, so as not to confuse beginners, that a closure is not always necessray in the initial example you gave. The closure is only required if you want to access the value of "i" directly. Also, just because "i" is equal to 99 when the object is clicked, does not mean you are clicking on object 99. So if you just need to access the current object, and don't care about outputting the value of "i", then you can just use the `this` keyword instead:

```
var myElements = [ /* DOM Collection */ ];

for (var i = 0; i &lt; 100; ++i) {
    myElements[i].onclick = function() {
        alert( 'You clicked on: ' this.id );

    };
}
```

So, with the above script, if you had set your elements to have incremental IDs in the HTML, then the alerted value would be equal to the ID of the object clicked. Using `this` can also be used to access a number of properties of the current object like `this.style` or `this.className`, etc. Only if you're dealing with the variable value itself are you required to use a closure.

Anyhow, thanks for a nice topic of discussion.

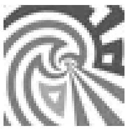
12.  *Martin Kirk* July 27th, 2009 at 1:03 pm

you might find that using closures will hunt you after some time…

one major problem with event-handling in JS is 'this', you never can be 100% sure what 'this' is 🙂

ie. you scroll or click on something and want to get the target element – some browsers give you the element by the 'this' keyword, others give you the page…

be careful 🙂

13.  *Nicolaj Kirkgaard Nielsen* August 19th, 2009 at 2:31 pm

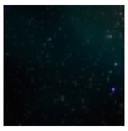
When trying to encapsulate the jQuery function like @mark suggested, JSLint gives me an error:

> Problem at line 53 character 3: Wrap the entire immediate function invocation in parens.
> })(jQuery);

The code looks like:

```
(function($){
        $(document).ready(function(){
                // Code...
        });
})(jQuery);
```

Any idea what JSLint is referring to here?

14.  *Guillermo Rauch* November 15th, 2009 at 10:55 pm

I don't understand Dean Edwards' or Dusan's comments. How does the guy age play a role in the analysis of the article? How can you question whether he should be blogging or not when you're both saying that the information is technically correct?

15. *James Rourke* November 16th, 2009 at 12:29 am

    It does seem extremely arrogant of them. Initially I thought Dean Edwards was being sarcastic, and the first paragraph would still seem sarcastic if not for the second one. Quite surprising really. Why shouldn't an 18 year-old kid write about what he thinks closures are about, inviting others to contribute to a discussion? After all, the way we imagine certain things in a programming language (like Objects) is worth sharing with each other.

16. *Johan* February 19th, 2010 at 9:07 pm

    I'm sorry, I don't usually flame people on the internet. I'll make an exception this time 🙂

    Dusan:

    "But I am simply wondering: Why are people reading this blog?"
    They find it informing, and as you said, it's well written. Perhaps you in your great experience could produce something of more interest.

    "Why am I reading a blog of an 18 year old "scripter" ?"
    Since when did age become of importance. I have a long time M.sc. in computer science, still I learned something reading this article. Your arrogance is your greatest flaw.

    "And on top of that, all is about one poorly made scripting language … Amazing."
    You said yourself you're using it alot. What does that make you?

Contact me // Terms