Based on how they are constructed, can callbacks also be defined as closures?

Asked 15 years, 11 months ago Modified 13 years, 10 months ago Viewed 269 times

4

In JavaScript, I know that a closure is can be defined as a nested function that has access to its containing function's variables. For example:

```
function outerFunction(x, y) {
    function innerFunction() {
        return x + y + 10;
    }
    return innerFunction;
}
```

Now, the following code is wiring up a callback for the `onreadystatechange` property of the request object; however, **I was wondering if, by definition, this is also considered to be a closure**:

```
/* This is a contrived example, I know.
 * Bear with me - it demonstrates the point I'm trying to convey. */

function submitHandler() {

    var oRequest = createRequest(); // assume I'm getting an instance of the
xhr
    var sUsername = 'Tom';          // assume this is needed for work in the
handler
    var This = this;
    oRequest.onreadystatechange = function() {
        This.handleResponse(oRequest, sUsername)
    }

}

function handleResponse(oResponse, sUsername) {
    if(oResponse.readyState === 4 && oResponse.status === 200) {
        // do work with the username
    } else {
        // we're not done yet...
    }
}
```

I realize that the `handleResponse` function could also just be written as an anonymous function in the context of `submitHandler`, but I find that more complex Ajax code can be more readable and easily maintained if callbacks are defined outside the scope of the function calling back to them. Again, this is a contrived example that I'm using in hopes of simply demonstrating the point of my question.

javascript closures callback

Share
Improve this question
Follow

edited Jan 19, 2009 at 14:11

asked Jan 19, 2009 at 13:16
Tom
15.9k ● 5 ● 50 ● 63

3 Answers

Sorted by: Highest score (default)

3

Yes, you are correct in assuming that it is a closure by definition.

It sounds like you know your stuff but here is a good, extensive article on javascript closures.

Share Improve this answer Follow

answered Jan 19, 2009 at 13:19
Andrew Hare
351k ● 75 ● 645 ● 641

3

I wholeheartedly agree that too many inline closure functions don't enhance readability. On the other hand I strongly dislike the `var self = this` style of writing closures, to which `this` is just a variant, as it's still too verbose in its declaration, and you introduce your own new 'keyword' whether that is `this` or `self`.

What you want is to curry/bind the method.

```
function $A(o)
{
    var a = [];
    for (var i = 0; i < o.length; ++i)
        a.push(o[i]);
    return a;
}

Function.prototype.bind = function()
{
   var _method = this;
   var a = $A(arguments)
   var o = a.shift();

   return function()
   {
       return _method.apply(o, a.concat($A(arguments)));
   }
}
```

```
Function.prototype.curry = function()
{
   var _method = this;
   var a = $A(arguments);

   return function()
   {
       return _method.apply(null, a.concat($A(arguments)));
   }
}
```

The methods are taken from the Prototype library. I use these even in projects that don't use the Prototype library as they are very useful!

In your case this means instead of writing:

```
var This = this;
oRequest.onreadystatechange = function() {
    This.handleResponse(oRequest, sUsername)
}
```

you can now write:

```
oRequest.onreadystatechange = this.handleResponse.curry(oRequest,sUsername);
```

However if you want to transfer the meaning of the this keyword you could do this

```
oRequest.onreadystatechange =
this.handleResponse.bind(this,oRequest,sUsername);
```

`handleResponse`, when called, will have the same `this` context as the `submitHandler`.

Share

Improve this answer

Follow

edited Feb 13, 2011 at 12:19

Peter Mortensen

31.6k ● 22 ● 109 ● 133

answered Jan 19, 2009 at 13:25

Martijn Laarman

13.5k ● 47 ● 64

1

Your first example is not a closure. It would have been had you returned innerFunction but you are executing the innerFunction and returning the result.

The second example works and is using a closure. One caveat with this specific technique is that you can end up with quite a severe memory leak in some browser implementations (IE certainly and may be others). You will have created a circular reference between the xhr held in oRequest, the onreadystatechange referencing a function and that functions scope being the scope where the xhr is held. Not only will the xhr remain in memory so will the response which if large (and especially if XML) can gobble memory quickly.

Set the onreadystatechange property to an empty global level function during the handling for the response. This will break the circle.

Share Improve this answer Follow

answered Jan 19, 2009 at 13:37

AnthonyWJones
189k ● 35 ● 235 ● 307

Thanks for the heads up - edited my initial example of a closure. Also, I like your discussion of the circular references on oRequest. I typically don't write my production-level code like this, but this seemed to be a quick example for purposes of this question. – Tom Jan 19, 2009 at 14:15