Limiting Variable Scope Using (function(){})();

By David Walsh on December 14, 2009 9

Simply put, a scope limiter is a self-executing function that defines a variables, performs a task, and clears those variables so that their scope is limited to the function itself. Take the following JavaScript code, for example:

```
/* do task 1:   */
var lynx = $$('a');
var divs = $$('div');
//(do stuff with links and divs)

/* do task 2 */
var lynx = $$('a'); //error:  lynx already defined!
var lis = $$('li');
//(do stuff with links and list items)
```

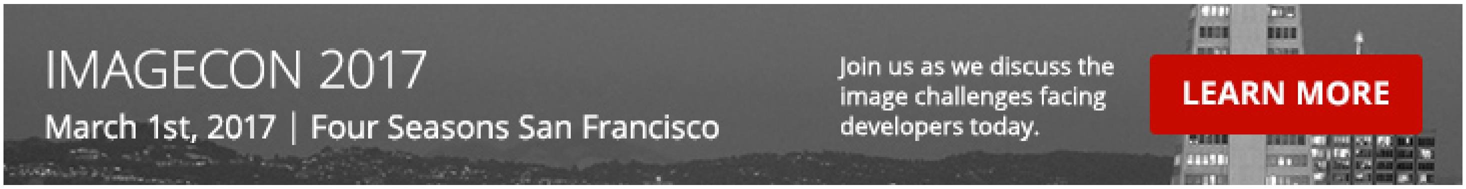
Everything above works but the second task can "see" the variables used to complete task one. This is undesirable as those variables from the first task could cause problems with later tasks. The better way to complete the two tasks is to use scope limiters for each:

```
/* do task 1:   */
(function() {
    var lynx = $$('a');
    var divs = $$('div');
    //(do stuff with links and divs)
})();
```

```
/* do task 2 */
(function() {
    var lynx = $$('a');
    var lis = $$('li');
    //(do stuff with links and list items)
})();
```

Sweet! Now the variables from the first task have scope only within their executed function scope and cannot affect other JavaScript within the "parent" scope.

Clean code FTW! Keep these techniques in mind when you're writing code that will be repurposed!



Recent Features

LightFace: Facebook Lightbox for MooTools

One of the web components I've always loved has been Facebook's modal dialog. This "lightbox" isn't like others: no dark overlay, no obnoxious animating to size, and it doesn't try to do "too much." With Facebook's dialog in mind, I've created LightFace: a Facebook lightbox...

Designing for Simplicity

Before we get started, it's worth me spending a brief moment introducing myself to you. My name is Mark (or @integralist if Twitter happens to be your communication tool of choice) and I currently work for BBC News in London England as a principal engineer/tech...

Incredible Demos

Add Styles to Console Statements

I was recently checking out Google Plus because they implement some awesome effects. I opened the console and same the following message: WARNING! Using this console may allow attackers to impersonate you and steal your information using an attack called Self-XSS. Do not enter or paste code that you...

CSS Scoped Styles

There are plenty of awesome new attributes we've gotten during the HTML5 revolution: placeholder, download, hidden, and more. Each of these attributes provides us a different level of control over an element on the page, but there's a new element attribute that allows...

Discussion

Richard

Proper programming technique is to define variables before use, and to use as few (preferably no) global variables as possible. What you have done in the examples are a work-around.

I love all your other articles though!

David Walsh

In this case you could define `$$('a')` globally but my example was more trying to display the two closures working separately.

Will

Doesn't this sort of miss the point of closures? I always thought that 'closure' meant it 'closed' over variables available to it when being declared, so those variables were available even when they went out of scope. It's hard to explain, but this example might help:

```
function makeClosure() {
    var t = "foo";
    return function() {
      alert(t + "bar");
    }
  }
```

So, you'd call makeClosure, which would return a function. Whenever the returned function was called, the value of t ("foo") would still be available to it (thus returning "foobar") even though t's scope had expired when makeClosure ended.

TheBigBabou

Sorry David, I don’t want to be offensive, but your code examples are no closures. What you are showing and describing is limiting variables to function scope.

A closure is a function or codepointer which accesses variables outside its scope. Like here:

```
(function () {

var out = 1;

window.setTimeout(function () {
  alert(out);
}, 100);

})()
```

The closure here would be the function, which is the first parameter to `setTimeout` . It doesn’t declare a variable “out”, but is able to access “out” from its outer function scope. As long as a variable stores a reference to this function, “out” is not freed from memory as well.

David Walsh

Ugh, updated my post as the terminology was off. This should make more sense and be more direct.

Jeremy Martin

Good post. Function scoping is a great way to keep your code clean. I often use a model like this:

```
(function() {
    // all my private stuff (vars and funcs) go here
    var foo = 'I am scoped, no one else sees me';

    var apiMethods = {
        // publicly exposed methods here
        // these methods still have access to privately scoped members
        bar : function() { alert(foo); }
    };

    // expose api methods
    window.MyNameSpace = apiMethods;
})();
```

```
// alerts value of foo
MyNameSpace.bar();
```

antonio torres

hi David!

i hope you can help me.
is it possible to implement your “load more posts” widget (http://net.tutsplus.com/tutorials/javascript-ajax/create-a-twitter-like-load-more-widget/) on a blogger blog? i guess the problem is the php implementation..
i like very much the script but i supose is not possible to implement it on a blogger thing…

any idea on that, David? maybe a go around trick…
thank you very much in advance
antónio torres
http://www.vaiumagasosa.com

Devi

Thanks David! Cleared up some stuff I never really understood in Javascript. This is a good page I came across on function scope in Javascript as well:

http://www.programmerinterview.com/index.php/javascript/javascript-function-scope/

hiho

Hi, ty, this is recommended to do today? or the pattern changed?
PD: sorry for my english :p

Name	Email	Website

Wrap your code in `<pre class="{language}"></pre>` *tags, link to a GitHub gist, JSFiddle fiddle, or CodePen pen to embed!*

□ **Continue this conversation via email**

Post Comment! Use Code Editor