JavaScript: Passing by Value or by Reference

In JavaScript, we have functions and we have arguments that we pass into those functions. But how JavaScript handles what you're passing in is not always clear. When you start getting into object-oriented development, you may find yourself perplexed over why you have access to values sometimes but not other times.

When passing in a primitive type variable like a string or a number, the value is passed in by value. This means that any changes to that variable while in the function are completely separate from anything that happens outside the function. Let's take a look at the following example:

```
function myfunction(x)
{
      // x is equal to 4
      x = 5;
      // x is now equal to 5
}

var x = 4;
alert(x); // x is equal to 4
myfunction(x);
alert(x); // x is still equal to 4
```

Passing in an object, however, passes it in by reference. In this case, any property of that object is accessible within the function. Let's take a look at another example:

```
function myobject()
{
        this.value = 5;
}
var o = new myobject();
alert(o.value); // o.value = 5
function objectchanger(fnc)
{
        fnc.value = 6;
}
objectchanger(o);
alert(o.value); // o.value is now equal to 6
```

So, what happens when you pass in a method of an object? Most would expect (or at least I did) that it would be passed by reference allowing the method to access other parts of the object it is apart of. Unfortunately, that's not the case. Check out this example:

```
function myobject()
{
        this.value = 5;
}
myobject.prototype.add = function()
{
        this.value++;
}
var o = new myobject();
alert(o.value); // o.value = 5
o.add();
alert(o.value); // o.value = 6
function objectchanger(fnc)
{
        fnc(); // runs the function being passed in
}
objectchanger(o.add);
alert(o.value); // sorry, still just 6
```

The problem here is the use of the '`this`' keyword. It's a handy short-hand for referring to the current object context. When passing a function as a parameter, though, the context is lost. More accurately, `this` now refers to the context of the object making the call instead of the object's function we just passed in. For standalone functions, this would be the `window` object and for functions called from an event, this would be the `event` object.

Solving the problem

There are two possible ways to get around this.

Option 1: When you know the method

If you know the method of the object that will be called then it's fairly easy. Just pass in the object instead of the function and call that instead. Using the `objectchanger` from the last example you'd get the following:

```
function objectchanger(obj)
{
        obj.add(); // runs the method of the object being passed in
}
objectchanger(o);
alert(o.value); // the value is now 7
```

Option 2: When you don't know the method

If you don't know the method of the object being passed in then you need to pass both the method and the object as parameters and use the `call` method. `call` is part of the JavaScript specification and allows a function to run in the context of another object. As a result, the `this` keyword will reference the right object: the object we passed in.

Here's our `objectchanger` function one more time:

```
function objectchanger(fnc, obj)
{
        fnc.call(obj); // runs the method of the object being passed in
}
objectchanger(o.add, o);
alert(o.value); // the value is now 7
```

Happy Scripting!

PUBLISHED JANUARY 18, 2006 · UPDATED SEPTEMBER 14, 2006
CATEGORIZED AS JAVASCRIPT
SHORT URL: https://snook.ca/s/503

Conversation

52 COMMENTS · RSS FEED

SAM MINNEE said on January 19, 2006 1

The bind method from prototype is helpful here. Let me rephrase that - I nearly fell off my chair when I realised how many problems that I had faced in my JS coding career could have been solved much more elegantly with it.

Bind effectively packages what you've done in option 2 into a reusable format.

```
objectchanger(o.add.bind(o));
alert(o.value); // this'll be 7 now
```

Some terse, inexplicable examples

```
obj = document.getElementById('wow');
myFunc = function() {
this.somethingCool();
}
setTimeout(myFunc.bind(obj), 500);

function myAjaxThing(request, onSuccess) { ... }
MyClass = function() {
...
}
MyClass.prototype = {
sendRequest : function() {
myAjaxThing(someURL, this.afterRequest.bind(this))
},
afterRequest : function(response) {
this.something = response.responseText;
}
```

COLIN D. DEVROE said on January 19, 2006 2

Great job at explaining this. I've run into this a few times, but never had the desire to write it up.

Thanks for sharing it.

VERACON said on January 19, 2006 3

I'm not too much into the JavaScript thing, but that call() method looks to work sort of like Python's getattr() -- is that what it does? Call a method from an object and the NAME of a method rather than the object of a method?

OSWALD said on January 20, 2006 4

@Veracon

It's method of each function. Simple takes the first given parametr, and use it as "this" keyword. Example:

```
function greeting() {
alert(this.message);
}

// global variable == property of the window object
var message = 'Hello universe!';

// "this" is window: 'Hello universe!'
greeting();

// object with own property "message"
var o = {message: 'Hello world!'};

// "this" is o: 'Hello world!'
greeting.call(o);
```

MISLAV said on January 20, 2006 5

> *"When passing in a primitive type variable like a string or a number, the value is passed in by value. (...) Passing in an object, however, passes it in by reference."*

It's wrong to say it that way. In JavaScript, **everything** is an object, even strings and numeric types.

JONATHAN SNOOK said on January 20, 2006 6

Mislav: the ECMAScript specification [PDF] actually makes somewhat of a separation between primitive types and objects:

A primitive value is a member of one of the following built-in types: Undefined, Null, Boolean, Number, and String; an object is a member of the remaining built-in type Object; and a method is a function associated with an object via a property.

MISLAV said on January 23, 2006 7

I am well aware of that. However, since even primitive types are objects I felt the "primitive type vs. objects" classification is wrong. Now I realize that the word 'object' is mostly used to indicate the member of Object (and derived classes) and to distinguish it from mentioned primitive types.

This (by value vs. by reference) behaviour is seen in PHP5 too, where objects are passed by reference by default - in contrast with other types.

LECOCHIEN said on February 21, 2006 8

is call() closed to eval() ?
eg. :

```
function objectchanger(fnc, obj)
{
//fnc.call(obj);
eval(fnc +'('+ obj +')');// Beurk !
}
objectchanger(o.add, o);
```

SOrry it's very dirty..

RODD SNOOK said on February 22, 2006 9

No lecochien, I don't think call() is like eval().

However, I believe Veracon can achieve something similar to the getettr() function like so:

```
function objectchanger(fname, obj) {
obj[fname]();
}

objectchanger('add', o);
```

This is a bit like the PHP paradigm of passing an array with an object and a method name in it to a built-in function.

(P.S. I am not related to Jonathan)

TDD said on March 10, 2006 10

The bind-related comment sure is useful. I love it.

However, provided that we can define the add method in the same code that defines the constructor, why not use closures instead, making usage even simpler:

```
function MyObj() {
this.value = 5;
o = this;
this.add = function() {
++o.value;
}
}

function myFunc(f) {
f();
}

var x = new MyObj();
alert(x.value);
x.add();
alert(x.value);
myFunc(x.add);
alert(x.value);
```

You'll get 5,6,7 (at least that's what I get on FF1.5).

The key here is to have add() *not* use "this", which is context-bound, but another name, which is closure-bound, such as "o", which here is a reference to the object bound to "this" in the constructor.

Of course, you can't use that with functions added elsewhere through the prototype mechanism, since they don't have the original object context.

TDD said on March 10, 2006 11

BTW, this bind thing is non-standard. It's not in the JS spec, nor even in the MSDN's JScript ref (worth a shot), and doesn't work on FF 1.5 for instance. Only call and apply (which are roughly

synonymous) seem to be standard, but they don't yield a context-adapted method pointer, they RUN the method, which makes them less useful.

JONATHAN SNOOK said on March 10, 2006 12

TDD: bind() is a feature of Prototype and is essentially a synonym for call(). (which was the trigger for this article... I left out closures and Prototype as I really wanted the article to just explain the topic at hand)

DANIEL LALIBERTE said on March 31, 2006 13

> *"When passing a function as a parameter, though, the context is lost. More accurately, this now refers to the context of the object making the call instead of the object's function we just passed in. "*

There is a better (i.e. correct) way to look at this. No context is lost when you pass a function as a parameter. When you call a method, you are *always* providing the context at that time, and if you call it as a function rather than a method of some object, it uses the default context, as you point out. (Hmm, what happens if you call it within some other method invocation, not the global context?)

Getting a method via an instance, e.g. o.add, never does the binding to the object you might assume, whether or not the method (as a function) is passed anywhere else. It might be nice if it did that binding, but it can also be handy to apply a method to a different object.

DAFIN said on June 06, 2006 14

Thanks, informative article

OWEN said on June 27, 2006 15

In your second option, what if the add function took a parameter? how would that change the syntax of the call function?

JOHANNES KNAUPP said on June 27, 2006 16

The header of this topic does not seem to me to reflect the problem.

> *So, what happens when you pass in a method of an object? Most would expect (or at least I did) that it would be passed by reference allowing the method to access other parts of the object it is apart of. Unfortunately, thatâ€™s not the case.*

I think you mixed up these two different things: the function/method, and its object.

The function is a non-primitive object, and therefore it is, of course, passed by reference. But the object refered to by the 'this' keyword is a different thing, and has nothing to do with the type of parameter passing.

Consider this modified sample:

```
function Myobject() {
        this.value = 5;
}
Myobject.prototype.add = function() {
        this.value++;
};

var o = new Myobject();
alert(o.value); // 5
o.add();
alert(o.value); // 6
// that's what we expect:
alert(o.add.h ? 'h() exists.'
              : 'No h(), sorry!');

function objectchanger(fnc) {
        fnc(); // runs the function being passed in
        // adds a static method:
        fnc.h = function() {
                alert('Hello!');
        };
}
objectchanger(o.add);
alert(o.value); // still just 6
alert(o.add.h ? 'h() exists.'
              : 'No h(), sorry!');      // :-)

// now let's call the new method:
o.add.h();      // :-)))
```

This clearly is no "standard" way of programming. It is only meant to demonstrate that the function was passed by reference: the change still exists after having returned form objectchanger().

JONATHAN SNOOK said on June 27, 2006 17

Johannes: thanks for helping to clear that up. Yes, the title really applies to the first half of the article where the discussion is by value or by reference whereas the second half really talks of the use of the 'this' keyword and possible ways to solve this.

PAUL said on August 22, 2006 18

Thanks, explained very well.

SURYAKANT GUPTA said on August 25, 2006 19

Very very thanx ,

u described it very well.

MIDNITE said on August 25, 2006 20

i think the different is the way of JS accessing the variable passed in. Try this:

```
function incArray(a){
a[0]++;
}
var testArray = new Array(1);
testArray[0] = 4;
document.writeln('testArray[0]: '+testArray[0]+'
'); // 4
incArray(testArray);
document.writeln('testArray[0]: '+testArray[0]+'
'); // 5
```

accessing array is by its address, while variable is by hash table. Try this:

```
function incVar(v){
v++;
}
var testVar = 4;
document.writeln('testVar: '+testVar+'
'); // 4
incVar(testVar);
document.writeln('testVar: '+testVar+'
'); // 4
```

this logic holds in every language.
adding * or & to a variable result in an error. so JS can't change the method of accessing its variables, i suppose.

any wiser solution ?

JONATHAN SNOOK said on August 25, 2006 21

midnite: it works as you might expect for arrays because the array is an object and gets passed by reference. For your second example, you'd have to have incVar return the modifed value back.

```
function incVar(v){
return ++v; //pre-increment
}
var testVar = 4;
// need to assign the result back
testVar = incVar(testVar);
```

MIDNITE said on August 25, 2006 22

won't it be too clumsy if many variables are required to be modified within the function?
why JS don't develope something like * or & like all others do ?
i thought it has. Or i'd better say i choose to believe that it has. But just not * or & like others, just i don't know what they are and how to use it.

RYAN BROOKS said on January 23, 2007 23

Super useful as a reference! Thanks!

GARY STEPHENSON said on April 03, 2007 24

The statement:

> *"Passing in an object, however, passes it in by reference."*

is simply wrong. It simply passes in **a** reference - by value. The reference itself **cannot** be altered - only the contents of the thing it refers to. There is a **big** difference between "passing-by-reference" and "passing-a-reference-by-value".

Javascript does **not** support the passing of parameters by reference. I find it amazing the number of Javascript writers that get this wrong.

If Javascript supported pass-by-reference, and supposing we could pass a parameter by reference by preceding it with an "@" modifier (a la Clipper), then we would have

```
function objectchanger(fnc)
{
    fnc = null;
}
```

```
f = new function() { ... }
objectchanger( @f );
alert( f ? "True", "False"); // -> "False"
```

pedantically yours, gary

OSAMU said on April 18, 2007 25

Dir Sir.

I am a translator. I am translating a programming document and encounterd the term ""pass in".

You use "pass in" many times. What means "pass in"

Thnak you.

GABE said on May 25, 2007 26

I believe the problem is a scope issue. If you pass a method to a function, the method will be called in the scope of that function. Because the parent object does not exist in the private scope of the function, its properties cannot be accessed from within the function. If you pass an object, its properties are available through the scope of that object.

If you place an alert(this); inside your method "add":

```
Myobject.prototype.add = function() {
        alert('called in scope of: '+this);
        this.value++;
};
```

You'll notice that you've isolated your method to where it becomes a global object (method of window).

Using the method "call" allows you to call a method in the scope of the calling object, not the scope you are calling from.

As shown by Johannes Knaupp, methods are passed by reference since they are objects themselves, its just that its in a scope where its parent Object does not exist.

SCOTT said on July 03, 2007 27

```
var a = 0;

function pauseMedia () {
        if (a = 1) {
                a = 0;
                playSong();
        } else {
                a = 1;
                pauseSong();
}

function playSong() {
        mediaPlayer.Play();
}

function pauseSong(){
        mediaPlayer.Pause();
}
```

Dear Sir,

As you can see the source code above, the playSong() will play the song when i press play button on remote control. pauseSong() will pause the song when i press pause button on remote control. Since i want to create a new button & want this new button to have click 1st time to pause the song, then click 2nd time to play the song again, then click again to pause the song & so on...

So i write the playPauseButton() function. But i only can pause the song & cannot play the song with clicking the same button. Why?

JONATHAN SNOOK said on July 03, 2007 28

@Scott: the problem you're running into isn't a scoping issue but rather that your variable never changes in value. Change your pauseMedia function to:

```
function pauseMedia () {
   if (a == 1) {
      playSong();
   } else {
      pauseSong();
   }
   a = 1 - a;
}
```

The last line constantly flips between 0 and 1 every time the function runs.

SCOTT said on July 03, 2007 29

Dear Sir,

Using back the example coding i given, may i know about will the a become 1 after pauseSong() (in the else section)?

JONATHAN SNOOK said on July 03, 2007 30

Ah, I see what you were trying to do. In which case, the assignment in the if statement was likely messing things up. It needs to be two == and not just one =.

SCOTT said on July 03, 2007 31

```
var a = 0;

function pauseMedia () {
if (a == 1) {
a = 0;
playSong();
} else {
a = 1;
pauseSong();
}

function playSong() {
mediaPlayer.Play();
}

function pauseSong(){
mediaPlayer.Pause();
}
```

Dear Sir,

Very sorry about my typing error. The code should look like this. What i need to know is the variable a's number. Is it the variable will pass the number i need?

For example, if a==0, set the a=1 and do pauseSong(). Is it the a will become 1? Is it it will replace 0 with 1? Will it pass the 1 to pauseMedia() and make the a==1 work?

JONATHAN SNOOK said on July 03, 2007 32

Scott, you've also got a missing close bracket } after pauseSong();. Assuming that bracket is in the right place, I don't see anything wrong with the code.

SCOTT said on July 04, 2007 33

Thank you Sir, you helped me a lot on solving my problem.

SHARATH said on September 11, 2007 34

Thanks for the article. It helped me solve my problem with select list box options, where I was adding items to the list by assigning values of a particular object. When I changed the value of the object, even the option element got changed because it was reference. Now I created new option element by using string argument to it, and the problem is solved.

ABILLEI BASNAYAKE said on September 26, 2008 35

What a nice website design you have here. I just found out your website when I'm searching for javascript passing value. I'm interested in the website layout design. Perhaps you can give tutorial on this subject.

DAVID SPECTOR said on October 29, 2008 36

I wrote a simple Ajax implementation, but then wanted to add multiple request objects (one for simple requests, another for concurrent data refreshing). Here's my solution, using a one-element array for passing the request object by-reference:

```
In caller:

var Req=[]; // General request object (short lifespan)
var ReqRcv=[]; // Receive data request object (long lifespan or polling)

function Loaded()
{
// Alloc both request objects
AjaxInit(Req);
AjaxInit(ReqRcv);
}

// Send Ajax request
Post(ReqRcv,'Receive',NrChars);

// In Ajax file:

function AjaxInit(Req) // Req is a 1-element array
{
if (window.XMLHttpRequest)
```

```
// Standard
Req[0]=new XMLHttpRequest();
// Req[0] is the actual request object being set here
...
}
```

MAJORYE said on November 20, 2008 37

Great,thanks for you sharing ,sir
I would like to see your blog,
from there ,can get much thing
I want to get, thanks.

--Major

MAJORYE said on November 20, 2008 38

Great,thanks for you sharing ,sir
I would like to see your blog,
from there ,can get much thing
I want to get, thanks.

--Major

ERIC SMILING said on December 19, 2008 39

The two solutions you provide are interesting. How do you feel about calling the method on the object it belongs to inside of a function that, for example, handles an event. For example:

function SomeObject(element, instanceVar){
this.element = element;
this.instanceVar = instanceVar;

//attach an event handler to the element
var currentObj = this;
this.element.onclick = function(e){
currentObj.clickHandler(e);
}
}
SomeObject.prototype.clickHandler = function(event){
// This function is an event handler of an element and a method of some object

```
// 'this' refers to the current object. We have access to all of it's instance vars and
// methods as well as the typical stuff we'd want in an event handler like a reference
// to the target as well as the event object...
var e = event || window.event;
doSomethingWith(this.instanceVar);
doSomethingElseWith(this.element);
}
```

JONATHAN SNOOK said on December 19, 2008 40

@Eric Smiling: I don't normally use that approach just for event handling as it makes the code less clear as to what's happening. In project these days, I often use the Prototype bind or bindAsEventListener feature which essentially handles proper binding for event handling.

SIMPLYKID said on January 07, 2009 41

This site is really great. It gives me view about passing primitive type variable that you can just pass it by value. And the great thing is about the scope issue that it really gives me clearly thought about how javascript handles passing by reference issue.

But I got some problem regarding executing function or method(but I'm not working with this one) from a string like eval() or setTimeout()

On what Sam Minnee wrote ~ setTimeout(myFunc.bind(obj), 500); ~
Will it run after 500ms (i have tried the same case with alert() and it fires immediately)?
Because from what i know setTimeout is accepting string as the first argument.

My problem is somehow same like this one. Here's my example code:

```
//This case about make some dropdownmenu appear with animating sliding down
function test(obj) {
        obj.style.height = '0px';
        slideMenuDown(obj, 0, 40);
}

//obj = object to be slided down, 2nd param is current height, 3rd param is final height
function(obj, ch, fh) {
        if ((ch < fh)) {
                        ch += 5;
                        if (ch>fh) ch = fh;
                        obj.style.height = ch + 'px';
                        setTimeout('slideDownMenu(' + objID + ',' + ch + ',' + fh + ')', 10);  //this is my pr
                        //if the function accept the id of the object, it works when i get the referenct by g
                        // but in this case, I don't feels like to use it.
```

```
                }
        else return;
}

//Let's assume 'ob' is the menu object from box element like <div> tag
test(ob);
```

Some notes here are:
- First, because sometimes when I didn't get the correct reference with getElementById() function, and maybe because there's 2 element with the same id (don't really care right now why, my prediction the cause is iframe),
I don't feel like to use the id value of the object.

- Second, I don't want to rely on global variable as possible as I can, so please understand me.

Is there any way to solve my problem?
Sorry to write this long, and if I'm out of topic then i'm really sorry too.

Thanks in advanced
~ Simplykid ~

SIMPLYKID said on January 07, 2009 42

sorry there some mistype in my code, that objID should be obj:

function(obj, ch, fh) {
if ((ch < fh)) {
ch += 5;
if (ch>fh) ch = fh;
obj.style.height = ch + 'px';
setTimeout('slideDownMenu(' + obj + ',' + ch + ',' + fh + ')', 10); //this is my problem...
//if the function accept the id of the object, it works when i get the reference by getElementById(),
// but in this case, I don't feels like to use it and not global variable too.
}
else return;
}

JONATHAN SNOOK said on January 08, 2009 43

@Simplykid, it sounds like what you want to do is take advantage of closures to maintain scope on the variables. Like this:

```
function animateThis(id) {
   var el = document.getElementById(id);
   var ch, fh;

   function slide() {
      ch += 5;
      if (ch > fh) ch = fh;
      el.style.height = ch + 'px';
      if (ch < fh) setTimeout(slide, 10);
   }
}
```

I'm obviously missing a lot of detail in this example but hopefully it explains the concept of what I'm trying to do here.

SIMPLYKID said on January 08, 2009 44

Wow, I just surprised that you can do this thing that make function in a function (or maybe it's just me that there's already a way like that since long ago). And well, i think it is possible, because by creating a function, it means also creating an object of function type. So, you can also adding some method to the object, in this case is the slide() function

I found dificulty at the beginning, but after some trial and error, i must add a statement that execute the slide() once for it to succeed.
And..
IT WORKS!!! THANKS A LOT! IT SOLVES MY PROBLEM!!

But there are still something i don't get.

First, that settimeout can accept something other than string code to be execute. After i search in the internet, then i know it can pointer of function
But is it still work with what Sam Minnee wrote ~ setTimeout(myFunc.bind(obj), 500); ~ which I don't think that's a pointer to function?

Second, why if i write down this code:
setTimeout('slide();', 10);
instead of
setTimeout(slide, 10);
it can't work?

And how to detect what is the context that call a function?

Thank for the solution.

JONATHAN SNOOK said on January 08, 2009 45

When you have the code quoted, all it does is get eval'd internally. You almost never want to or need to use eval. So, setTimeout(slide,10) runs the slide function in 10ms and the context ends up being the global (window) object, which is why the use of closures helps you retain access to important variables.

The setTimeout(myFunc.bind(obj), 500) calls the bind function immediately but the bind function *returns* a function that gets called after 500ms. It's a little more confusing, I know. :) The bind function is part of PrototypeJS (and is often found in many of the other JavaScript libraries).

SIMPLYKID said on January 08, 2009 46

Well, that's really make sense since setTimeout is a method of window object, and slide function is just available through the animateThis function if you don't have the pointer to the slide function.

That really clears everything on my mind now.
Though i still need to know if there's a way how to detect the context that call a function within the function itself, because if i know this it will be very helpful to do testing in another occasion.

Thanks a lot for all that has given their thought in this site especially this site's owner.
This site is really great, very helpful

JEAN said on January 15, 2009 47

Can somebody explain the results of the following to me in light of this whole issue of passing methods by reference. Why do I get the results I did based on the main example of this post with some added changes.
I created 3 functions: incMyO1, incMyO2 and incMyO3 whick increments the value of o. The only difference is what gets send to them. The first to give the o.value as 6 but the last one gives o.value as 7, why is this? Note that only one should be run at a time, comment out the other two when testing:

```
function Myobject() {
this.value = 5;
this.add = function() {
++this.value;
}
}
//Myobject.prototype.add = function() {
// this.value++;
//};
```

```
function incMyO1(t) {
t();
}

function incMyO2(t) {
t;
}

function incMyO3(t) {
t;
}

var o = new Myobject();
alert(o.value); // result = 5

o.add();
alert(o.value); // result = 6

//incMyO1(o.add);
//alert(o.value); // result = 6

//incMyO2(o.add);
//alert(o.value); // result = 6

incMyO3(o.add());
alert(o.value); // result = 7
```

Thanks!

JONATHAN SNOOK said on January 15, 2009 48

Jean, your incMyO3(o.add()) line increments the value and then passes the return value (of which is undefined, since your value method doesn't have a "return" statement). incMyO1 doesn't increment because you're only passing the value function into the incMyO1 function which loses the o object context (and therefore incrementing its value does nothing).

I hope that paints a clearer picture.

JEAN said on January 15, 2009 49

Thanks snook. Thought it was something obvious. I'm just trying various tests to get my head around this topic. Great topic, very informative. By the way, great looking site!

JAIRO said on March 09, 2011 50

Thanks. This solved my problem, and it is also applicable in attributes. My problem was this, I had two objects, with an array attribute routes:

curr.routes
new.routes

when I pass them in a function (which compares 2 arrays and returns an array with the elements that exists on both array; non-destructive), the attributes of the objects are erased.

prev = compareArray(curr.routes, new.routes)
// curr.routes, new.routes is now erased

To solve this, I instantiated 2 variables for each routes array, then passed it to the function:

var currRoutes = []; currRoutes = curr.routes;
var newRoutes = []; newRoutes = new.routes;
prev = compareArray(currRoutes, newRoutes);
// curr.routes & new.routes are retained.

Thanks again!

SAMS CLUB CREDIT CARD said on March 31, 2011 51

I am glad to be one of several visitants on this outstanding website (:, thankyou for posting .

BGC2033 said on April 28, 2011 52

Indeed. Why pass variables to functions at all when you can return custom objects with all the data you could ever dream of.

Sorry, comments are closed for this post. If you have any further questions or comments, feel free to send them to me directly.

Hi. I'm Jonathan Snook and I write about web development.
I wrote SMACSS. I tweet. I speak. Want to learn more?