Beautiful JavaScript: Easily Create Chainable (Cascading) Methods for Expressiveness

AUG. 13 2013 28

(Part of the "12 Powerful JavaScript Tips" Series)

menu

Prerequisites:
— Understand JavaScript's "this" With Ease
— JavaScript Objects in Detail

Chaining Methods, also known as **Cascading**, refers to repeatedly calling one method after another on an object, in one continuous line of code. This technique abounds in jQuery and other JavaScript libraries and it is even common in some JavaScript native methods.

Writing code like this:

```
$("#wrapper").fadeOut().html("Welcome, Sir").fadeIn();
```

or this:

```
str.replace("k", "R").toUpperCase().substr(0,4);
```

is not just pleasurable and convenient but also succinct and intelligible. It allows us to read code like a sentence, flowing gracefully across the page. It also frees us from the monotonous, blocky structures we usually construct.

Receive Updates

Your Email: Go

We will spend the next 20 minutes learning to create expressive code using this cascading technique. To use cascading, we have to return *this* (the object we want subsequent m to operate on) in each method. Let's quickly learn the details and get back to eating, or watching YouTube videos, or reading Hacker News, or working and browsing, or working and focusing.

Let's create all of our "chainable" code within an object, along with a local data store. Note that in a real-world app we will likely store the data in a database, but here we are just saving it in a variable.

```
 // The data store:
var usersData = [
    {firstName: "tommy", lastName: "MalCom", email: "test@test.com", id: 102},
    {firstName: "Peter", lastName: "breCht", email: "test2@test2.com", id: 103},
    {firstName: "RoHan", lastName: "sahu", email: "test3@test3.com", id: 104}
];


// A quick utility function that does what it says:
function titleCaseName(str) {
    return str.replace(/\w\S*/g, function (txt) {
        return txt.charAt(0).toUpperCase() + txt.substr(1).toLowerCase();
    });
}


// Our object with the chainable methods
var userController = {

    currentUser: "",

    findUser: function (userEmail) {
        var arrayLength = usersData.length, i;
        for (i = arrayLength - 1; i >= 0; i--) {
            if (usersData[i].email === userEmail) {
                this.currentUser = usersData[i];
                break;
            }
        }
        return this;
    },

    formatName: function () {
        if (this.currentUser) {
            this.currentUser.fullName = titleCaseName(this.currentUser.firstName) + " " + titleCaseName
        }
        return this;

    },

    createLayout: function () {
        if (this.currentUser) {
            this.currentUser.viewData = "<h2>Member: " + this.currentUser.fullName + "</h2>"
            + "<p>ID: " + this.currentUser.id + "</p>" + "<p>Email: " + this.currentUser.email + "</p>"
        }
        return this;
    },

    displayUser: function () {
        if (!this.currentUser) return;

        $(".members-wrapper").append(this.currentUser.viewData);

    }
};
```

With our chainable methods defined, we can now execute our expressive code like this (just like it is done in jQuery):

```
 userController.findUser("test2@test2.com").formatName()
.createLayout().displayUser();
```

Here is the result on JSBin:
http://jsbin.com/erewat/1/edit

Why Use Cascading in JavaScript?

- There is no need to create temporary variables to save each step of the process. For example, without chaining, our code will look like this:

```
var aUser = userController.findUser("test@test.com");
var formatUserName =  aUser.formatName();
var layoutHTML =  formatUserName.createLayout();
userController.displayUser(layoutHTML);

```

- Now, every line of code clearly and succinctly expresses what it is doing, particularly when the name of each method is defined using verbs.
- Our code is more maintainable because we have simple, lean, specialized methods.
- Overall, one can easily read the “chainable” code, effortlessly type it, and comfortably understand it.

How Does Chaining Methods Work in JavaScript?

When each method returns *this,* the entire object that called the method is returned. The execution proceeds thus:

```
// Use the userController object to execute the findUser method
userController.findUser("test@testdd.com")
```

Because we are executing the *findUser* method on the *userController* object, and because the findUser method returns “this” (the object that invoked it), the entire userController object is returned and passed to the next method in the chain, since the “this” keyword in findUser holds the value of the object that invoked it.

Therefore, this occurs next:

```
userController.formatName();
```

Similarly, the *formatName* method returns the userController object, so expectedly, this follows:

```
userController.createLayout();
```

Followed by:

```
<script>
userController.displayUser();
```

Each step of the way, we are returning the userController object and invoking method:

Final Words

This was probably the shortest post I have written so far. I am hopeful it is just as informative as the others and you will start using cascading to make your JavaScript code more expressive and beautiful.

As always, if you see any bugs, typos, or grammar mistakes, please notify me promptly.

Be good. Sleep well. And enjoy coding.

Posted in: 12 Powerful JavaScript Tips, JavaScript / Tagged: 12 Powerful JavaScript Tips, Intermediate Javascript

Richard
Thanks for your time; please come back soon. Email me here: javascriptissexy at gmail email, or use the contact form.

28 Comments

Mattheu

August 22, 2013 at 10:54 am / Reply

Thanks for this. I always wnat to know how jQuery did chaining. It is beautiful stuff :).

Richard Bovell (Author)

August 23, 2013 at 11:27 am / Reply

You are welcome, Mattheu. I am always happy to hear when the articles are helpful.

Morgan

October 14, 2013 at 9:34 pm / Reply

Thanks Richard, very clear explanation.

hidoos

October 16, 2013 at 11:58 pm / Reply

Thanks to share! This is very useful!

Steve

October 26, 2013 at 10:37 pm / Reply

Chaining is cool… but just make sure you understand how things will behave if one of the functions needs to throw an error…

Richard Of Stanley (Author)

October 29, 2013 at 2:42 pm / Reply

Thanks for your input, Steve. I guess I should add that bit to the post. I will do that.

Leslie

November 4, 2013 at 11:36 pm / Reply

Thanks a lot. But in fact, if you write to much chain, it will be hard to maintenance.

Vinod

November 6, 2013 at 2:34 am / Reply

Here's another brief article on how to achieve chaining.

http://goo.gl/z0qNB6

Karl pokus

November 11, 2013 at 2:07 pm / Reply

Thanks for sharing. Very well written.

Jabir

January 18, 2014 at 7:44 am / Reply

Really helpful

Jabir

January 18, 2014 at 7:45 am / Reply

It's really good

Luis

March 4, 2014 at 9:11 pm / Reply

Hi Richard,
how are you?
First, I'm obliged to say that you are a really good teacher, and I don't say this lightly: I've been to thousands of classes in my life and most teachers are

so you have a rare talent.
I'm a beginner at this game and I have a beginner's question that I hope you may answer.
I've build this simple object and it worked fine with a small external array, but when I tried to use it with an array generated by a DOM document method it failed.

Here is the dom method that should generate the array to be used as a variable in the object:

var eleArray = document.getElementById(div).getElementsByTagName(ele);

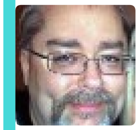
And here's the object:

```
var normalizeDots = {

dotArray: [ ],
stringDots: ".",

searchArrays: function () {

for (i = 0; i < eleArray.length; i++) {
this.dotArray.push(eleArray[i].match(/[.]/g));
}
return this;
},

displayResult: function () {
alert(this.dotArray[2].length);
}
}
```

What I've been able to see is that the array variable is generated but it doesn't seem to be available to the object. How can I solve this?
thanks in advance

Henrique Silvério
March 6, 2014 at 8:32 pm / Reply

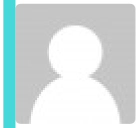
Very useful tricks. Thanks! =]



John McCormick

March 23, 2014 at 1:29 pm / Reply

I started using this technique in a C++ program I work on. When I created a new error container it made a lot of sense to do it this way as it allowed me to build the object in a clear and concise way in a single line of code.



JamesP

March 24, 2014 at 11:05 am / Reply

but here we are just saving it a variable <——

and

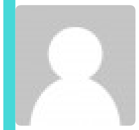
userController.displayUser();

No need for the orphaned tag

Vinayak

May 3, 2014 at 6:55 am / Reply

Thanks for the post..!!



rohan

May 8, 2014 at 8:33 am / Reply

Really very helpful. Did not thought that chaining implementation will be this much easy… very nice post

zaib ghaffar

June 4, 2014 at 2:42 am / Reply

This is very informative, clear and concise .

Peter

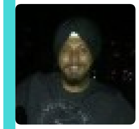
July 15, 2014 at 8:49 am / Reply

Hi,

As of now I got through almost all of your articles and have learned sooo much more than I used thourgh a single book.

I'm so greatful for that. But Its been almost a year I haven't seen any new articles… .Did you stop writing?

Regards



Jatinder

July 18, 2014 at 2:58 am / Reply

Thanks for sharing. Nice to know more about 'this' and JS cascading.

form

October 28, 2014 at 12:56 pm / Reply

Comment form not labeled – which is name, which is email. Test.

David

October 31, 2014 at 10:43 am / Reply

Thank you for a nice, clear exploration of this. I'm ambivalent about method chaining though. If you are the sole developer or part of a small team, masters of your domain, then it allows you to create succinct and tidy code. But it relies on the implicit assumption that each method in the chain returns "this" – the "intelligibility" of the code is reliant entirely on that assumption. If you're creating the code and maintaining it yourselves, then great.

The problem for a subsequent developer who may inherit your code in a support role a few years down the line is that they will have to inspect your code to see what is being returned by each method in the chain to ensure that indeed "this" is being returned and that the chain is doing what on value they would expect, ie chaining. A perverted or annoying developer could easily make a method at some point in the chain return something other than "this" so that subsequent function calls in the chain are no longer operating on the initial object.

Client side code needs to be bandwidth-light and in that environment chaining makes perfect sense. But if you are writing server-side JS which is likely to be maintained by other people, probably support people without in depth knowledge of the code, the golden rule is clarity, even at the expense of brevity. If I had a dollar for every time that someone who thought that terseness was the same as simplicity, and subsequently created code that looked nice but which was error-prone for the unwary to maintain, I'd be a wealthy man.

Matt Wood

November 18, 2014 at 1:36 pm / Reply

Thanks for a short and sweet post, Richard!

Christian Ruocco

January 29, 2015 at 3:27 am / Reply

Sorry to be *that* guy, but your code is FAAAAAAR from beautiful or clear!

$("#wrapper").fadeOut().html("Welcome, Sir").fadeIn();

I feel like i'm taking crazy pills! My brain has to work to read that. It's a novelty to pack so much action into one line, but it fails hopelessly on the clear and beautiful front. Chaining is great but certainly does not equate directly to more beautiful or clear code. Seperating chains by line helps a bit though as in:

```
var thing = new UIThing();
thing
.setParent(blah)
.moveTo(x, y)
.setSize(100, 100);
```

No! There is so much more to this. It's not gonna be a simple bunch of steps in the same way painting by numbers doesn't turn you into Picasso. It IS an art form in itself. My search continues..

Just my 2 cents. 🙂

Thomas

November 22, 2015 at 2:01 am / Reply

Here is another easier tutorial on chaining – for beginners

http://schier.co/blog/2013/11/14/method-chaining-in-javascript.html

Joseph Gringeri

March 18, 2016 at 8:49 pm / Reply

Really, really great article and content on the site overall. I've been digging deeper into JS and jQuery in recent weeks and this really helps me understand both a bit better and how I can use methods similar to jQuery in a more native way.

Trackbacks for this post

1. Beautiful JavaScript: Easily Create Chainable (...
2. javascript - understanding the return this in carasoul.js - javascript

Leave a Reply

Name	Email (hidden)	Website

Comment:

Submit Comment

☐ Notify me of follow-up comments by email.

☐ Notify me of new posts by email.