# Write Better JavaScript with Promises



*By Landon Schropp on April 14, 2014* 40

You've probably heard the talk around the water cooler about how promises are the future. All of the cool kids are using them, but you don't see what makes them so special. Can't you just use a callback? What's the big deal? In this article, we'll look at what promises are and how you can use them to write better JavaScript.



## Promises are Easier to Read

Let's say we want to grab some data from the HipsterJesus API and add it to our page. This API responds with data that looks like this:

```
{
  "text": "<p>Lorem ipsum...</p>",
  "params": {
    "paras": 4,
    "type": "hipster-latin"
  }
}
```

Using a callback, we'd write something like this:

```
$.getJSON('http://hipsterjesus.com/api/', function(data) {
  $('body').append(data.text);
});
```

If you're experienced with jQuery, you'll recognize we're making a `GET` request and expecting JSON in the response body. We're also passing in a callback function that takes the response JSON and adds it to the document.

Another way to write this is to use the promise object returned by the `getJSON` method. You can attach a callback to this object directly.

```
var promise = $.getJSON('http://hipsterjesus.com/api/');

promise.done(function(data) {
  $('body').append(data.text);
});
```

Like the callback example, this appends the result of the API request to the document when the request is successful. But what happens if the request fails? We can also attach a `fail` handler to our promise.

```
var promise = $.getJSON('http://hipsterjesus.com/api/');

promise.done(function(data) {
  $('body').append(data.text);
});

promise.fail(function() {
  $('body').append('<p>Oh no, something went wrong!</p>');
});
```

Most people remove the `promise` variable, which makes it a little easier to tell what the code does at a glance.

```
$.getJSON('http://hipsterjesus.com/api/')

.done(function(data) {
  $('body').append(data.text);
})
```

```
.fail(function() {
  $('body').append('<p>Oh no, something went wrong!</p>');
});
```

jQuery also includes an `always` event handler that's called regardless if the request succeed or fails.

```
$.getJSON('http://hipsterjesus.com/api/')

.done(function(data) {
  $('body').append(data.text);
})
.fail(function() {
  $('body').append('<p>Oh no, something went wrong!</p>');
})
.always(function() {
  $('body').append('<p>I promise this will always be added!.</p>');
});
```

With promises, the order of the callbacks is respected. We're guaranteed to have our `done` callback called first, then our `fail` callback, and finally our `always` callback.

# Better APIs

Let's say we want to create a wrapper object for the HipsterJesus API. We'll add a method, `html`, to return the HTML data that comes down from the API. Rather than having this method take in a handler that's called when the request is resolved, we can just have the method return a promise object.

```
var hipsterJesus = {
  html: function() {
    return $.getJSON('http://hipsterjesus.com/api/').then(function(data) {
      return data.text;
    });
  }
};
```

The cool thing about this is we can pass around our promise object without worrying about when or how it resolves its value. Any code that needs the return value of the promise can just register a callback with `done`.

The `then` method allows us to modify the result of a promise and pass it to the next handler in the chain. This means we can now use our new API like this:

```
hipsterJesus.html().done(function(html) {
  $("body").append(html);
});
```

Until recently, one of the killer features of AngularJS was that templates could bind directly to promises. In an Angular controller, this looked like:

```
$scope.hipsterIpsum = $http.get('http://hipsterjesus.com/api/');
```

Then, it was as simple as writing `{{ hipsterIpsum.text }}` in a template. When the promise resolved, Angular would automatically update the view. Unfortunately, the Angular team has deprecated this feature. For now, it can be enabled by calling `$parseProvider.unwrapPromises(true)`. I hope Angular and other frameworks include this feature going forward (I'm looking at you Ember).

# Chaining

The best part about promises is you can chain them! Let's say we want to add a method to our API that returns an array of paragraphs.

```
var hipsterJesus = {

  html: function() {
    return $.getJSON('http://hipsterjesus.com/api/').then(function(data) {
      return data.text;
    });
  },
```

```
  paragraphs: function() {
    return this.html().then(function(html) {
      return html.replace(/<[^>]+>/g, "").split("");
    });
  }
};
```

---

We've left our HTML method the same, and we're using it in the `paragraphs` method. Because the return value of a promise's callback is passed to the next callback in the chain, we're free to create small, functional methods that change the data as it's passed through them.

We can chain promises as many times as we want. Let's add a method for sentences.

---

```
var hipsterJesus = {

  html: function() {
    return $.getJSON('http://hipsterjesus.com/api/').then(function(data) {
      return data.text;
    });
  },

  paragraphs: function() {
    return this.html().then(function(html) {
      return html.replace(/<[^>]+>/g, "").split("");
    });
  },

  sentences: function() {
    return this.paragraphs().then(function(paragraphs) {
      return [].concat.apply([], paragraphs.map(function(paragraph) {
        return paragraph.split(/. /);
      }));
    });
  }
};
```

---

# Multiple calls

Probably the most notable feature of promises is the ability to combine multiple API calls. When using callbacks, what happens if you need to make two API calls at once? You'll probably end up writing something like this:

```
var firstData = null;
var secondData = null;

var responseCallback = function() {

  if (!firstData || !secondData)
    return;

  // do something
}

$.get("http://example.com/first", function(data) {
  firstData = data;
  responseCallback();
});

$.get("http://example.com/second", function(data) {
  secondData = data;
  responseCallback();
});
```

With promises, this becomes much easier:

```
var firstPromise = $.get("http://example.com/first");
var secondPromise = $.get("http://example.com/second");

$.when(firstPromise, secondPromise).done(function(firstData, secondData) {
  // do something
});
```

Here, we're using the `when` method to attach a handler that's called when both requests are done.

# Conclusion

That's it! Hopefully you have a sense of some of the awesome things you can accomplish with promises. What's your favorite way to use them? Let me know in the comments!

*Note: For simplicity, this article is using jQuery's deferred implementation. There are subtle differences between jQuery's `Deferred` object and the Promises/A+ specification, which is a more canonical standard. For more information, check out Q's Coming from jQuery wiki.


## About Landon Schropp

Landon is a developer and entrepreneur based in Seattle. He's the author of the Free Flexbox Starter Course and Unraveling Flexbox, a book on how to create modern, responsive layouts in CSS. He's passionate about building simple apps people love to use.

landonschropp posts




## 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...

### Create Spinning Rays with CSS3: Revisited

Last December I wrote a blog post titled Create Spinning Rays with CSS3 Animations & JavaScript where I explained how easy it was to create a spinning rays animation with a bit of CSS and JavaScript. The post became quite popular so I...

## Incredible Demos

### Downloadify: Client-Side File Generation Using JavaScript and Flash

The following tools is in its very beta stages and works intermittently. Its so damn useful that I had to show it off now though! I recently stumbled upon Downloadify, a client-side file generation tool based on JavaScript and Flash ActionScript code. A...

### Image Data URIs with PHP

If you troll page markup like me, you've no doubt seen the use of data URI's within image src attributes. Instead of providing a traditional address to the image, the image file data is base64-encoded and stuffed within the src attribute. Doing so saves...

## Discussion

#### Steve

Great article... Promises/Deferred Objects do make for great readable code and better organization.

I thought I should point out though that there are some scenarios where Promises are not easy to implement (e.g. where JavaScript is available, but `setTimeout()` and `setInterval()` are not)

Chaining and waiting for multiple handlers to return is still highly desired and thus with "Necessity being the Mother of Invention" I created a JavaScript tool called "PseudoPromise" to fill in such a gap.

https://github.com/scunliffe/PseudoPromise

It isn't a complete (nor complying) implementation (and doesn't pretend to be) but it has proven very useful to me and the folks that have tried it out. Should anyone read your article and be looking for a very simple "wait until multiple async events have completed then do X" utility without having promises available this may be just the thing for them.

#### Landon Schropp

That's interesting. Can you give and example of a situation when you'd use your library? I'm having a bit of trouble understanding how `setTimeout` and `setInterval` relate to promises.

#### Shamari Feaster

I deal with asynchronous stuff all the time with my bootstrap program structureJS (https://github.com/ShamariFeaster/structureJS). I looked at your PsedoPromise class at frankly it will not do what you advertise it to. IF I pass it an async function, you don't pass my async a callback to signal your PsuedoPromise object that I have actually completed. You just run them all and then mark them as completed, which in the case of async functions will most likely not be true.

### Bob Rogan

Sadly "Promises" seems to be a collection of disjointed code snippets that attempts to blindly deal with Asynchronous Java-script object creation and management with little understanding or use of the inherent object oriented nature of the Java-script language or the goal of asynchronous client-side web-site application design. Check out the AJOMS.COM web-site

### Alex

Also, I thought I should point out, since some one is, jQuery's promises are a little broken (or diff), https://thewayofcode.wordpress.com/tag/jquery-deferred-broken/

### Volker Rose

Domenic Denicolas "You're Missing the Point of Promises" is a very good read all around Promises, the according Specs and the different implementations within the popular frameworks.

Another pattern is often saw and use myself is naming the promise in an "active" way:

```
var gettingExampleData = $.get("http://example.com/first");
var gettingAnotherExampleData = $.get("http://example.com/second");

$.when(gettingExampleData, gettingAnotherExampleData).done(function(firstData, secondData)
  // do something
});
```

Makes up a good read(able code) :) I probably saw it in the Alex McPhersons brillant "I .promise() to show you .when() to use Deferreds" talk from the jQuery Conference San Francisco 2012, http://www.confreaks.com/videos/993-jqcon2012-i-promise-to-show-you-when-to-use-deferreds.

### Alex McPherson

My ears are burning! Thanks for the nice shout out :-)

### Fagner Brack

Totally agree with the naming stuff, it helps to prevent lots of confusions, see https://medium.com/@fagnerbrack/promises-are-not-proxies-fd00751eb980

### Volker Rose

Doh, missed the link to Domenics article: http://domenic.me/2012/10/14/youre-missing-the-point-of-promises/.

### Amy

Cool stuff! Great article

### Charles

If you enjoyed the taste of promises here you may want to check out Matt Greer's article that gets down to the nitty gritty details of promise usage.

http://mattgreer.org/articles/promises-in-wicked-detail/

### Collin

Most articles about promises use the very simple example of a $.get request. This is all fine and good, but why is there no explanation on how to actually create the promise object you're consuming? This is only half the story about promises and how to actually use them.

With a title like "Write Better JavaScript with Promises" I think people would expect way more. Maybe change it to "Write Cleaner $.get requests with Promises".

### Nitij

If anyone is wondering how promise and deferred objects work internally then here is a novice attempt to make such deferred object in JavaScript.

https://github.com/Nitij/NoviceJsPromise

### Landon Schropp

I think your point is fair. Writing a post like this is always challenging because if I stray into the details too much, people lose interest. For this article, my main goal was getting people used to the idea of using promises instead of callbacks. I chose to use jQuery's Ajax functions as an example because I thought people would be familiar with them and there are several common applications I could point out.

### Will

The "older" button overlapping the text on the this page is super annoying.

### Javier Callico

It there a naming convention for functions that return promises? For example, just by looking at the html() function I can't tell that a promise is been returned. It would be nice to have something similar to the Ruby convention of appending a ? at the end of boolean methods.

#### Ed Hasting-Evans

TypeScript brings typing to JS, as well as interfaces, classes and modules. Might give you some of what you're after....

#### Tiberiu

I use 'Async' suffix for these functions, htmlAsync looks pretty self-explanatory.

### Shahar

Great article! Another, very inspiring, article about promises is:
https://blog.jcoglan.com/2013/03/30/callbacks-are-imperative-promises-are-functional-nodes-biggest-missed-opportunity/

### Darryl Young

Great article! I've been meaning to read up on Promises and this is just what I've been looking for. Thanks.

### Aram Koukia

Nice, I think I can reconsider many parts of our JS codes using promises.
There are many instances of stupid `setTimeout` stuff to fix some minor racing condition

issues ...
Very useful ...

### Drew

Speaking of setTimeout....

```
$.wait = function(time) {
      return $.Deferred(function(dfd) {
        setTimeout(dfd.resolve, time);
      });
    };
```

I ran into that trick here: http://www.intridea.com/blog/2011/2/8/fun-with-jquery-deferred

That's just the simplest iteration of it (you could also modify it to handle midway notifications, pass arguments through, etc.) but it's a neat, expressive way to handle timeouts, including all the the benefits of Deferreds. For instance, consider this:

```
function maybeGetApi(){
  return $.wait(500).then(function(){
    $.getJSON();
  });
}

var api  = maybeGetApi();

api.done(useAPI);
```

That basically waits 500ms before making a request. But note that the external api doesn't have to know or care about that: it just gets a Deferred. But not a promise. Why? Because one of the things we can do with it now is reject "api" as long as it's still in the timeout phase, thus preventing the request from ever being made. Once you transition various bit of javascript over into promises/Deferreds you start to see ways to chain them that never quite made sense before.

But the ultimate win, imho, is simply that your application logic becomes far more readable. You can hide all the complicated bits away from the actual main execution functions, and they just work.

### Landon Schropp

That's awesome! I wasn't aware of the Deferred syntax where you pass in a function.

### Jan Hesse

I encourage you to digg more into Ember.js Promises are great UseCases for this framework. What you describe in your example is also possible out of the box in Ember, though you should use a model instead of binding to the result directly.
Skip to “Second Thing: Ember Loves Promises” here:
http://www.wekeroad.com/2014/05/28/the-frustratingly-lovable-crazy-making-huggable-ball-of-whack-that-is-ember-js/

#### Landon Schropp

That’s cool! I didn’t know Ember was so promise friendly. I’m actually doing a project in Ember now and I’ll definitely be making use of this.

### Tony

Promises are the way to go

### Olga Yasovsky

Alright!
I might do some code re-factoring after reading this article.
Thanks for the post!

### Raymond Camden

Query, in this part: “With promises, the order of the callbacks is respected. We’re guaranteed to have our done callback called first, then our fail callback, and finally our always callback.” You are saying, as I read it, that *all* the callbacks are run, but that isn’t true. fail will only run is the call fails. Also, is order really important? If I did `.always().fail().done()` won’t that work properly as well? (Calling done then always on a good call and fail then always on a bad call?)

#### Landon Schropp

That’s not quite what I mean. When I said the order is respected, I meant the order of the callbacks that are actually fired. Take this contrived example:

```
$.getJSON("http://example.com")
.always(function() {
  $(".message").empty();
})
.done(function() {
  $(".message").html("Done!");
});
```

Were the always and done callbacks reversed, the code would behave very differently. Also, keep in mind you can attach as many callbacks as you want to a promise and promises can be passed around anywhere, which would get very complicated in larger applications if the order weren't respected.

### Drew

To add on: while that example doesn't use or change the result of getJSON call, you could imagine it doing so if it used .then(). These orderings give very different results:

```
function log(){console.log.apply(console,arguments);}
function addOne(number){return number+1;}

$.getJSON("http://example.com/return5")
.always( log )
.then(addOne)
.then(addOne)
.done( log );
//logs: 5
//logs: 7


$.getJSON("http://example.com/return5")
.always( log )
.then(addOne)
.done( log )
.then(addOne);
//logs: 5
//logs: 6
```

In the second function, the final return is still a Promise that wraps the value "7" we saw in the first case, but since we logged the case at a different point, the value logged was different. That's because with .then, you're composing functions over the wrapped result here.

jQuery's .done, .fail are basically just syntactic sugar for a usage of .then() & .catch() that will always simply return a new Promise wrapping the same result that was originally passed in, regardless of what the functions they include happen to do. But the entire chain proceeds by composing together the callbacks over the first wrapped result, then returning a Promise that wraps the accumulated result of that composition.

Landon's example is about two side effects potentially happening in the wrong order. My example is about the order of composition (with a side effect to peek into what's happened at what point).

### Erick Jones

Good article, I need to recode things with this approach.
Hipster Jesus :D

### Fatih Toprak

@Hipster Jesus :)

What a great article. Thanks London.

### Brian Vaughn

Great introduction to Promises, Landon!

I'd love to hear your thoughts on a Promise-like library I've been working on recently- Task Runner (github.com/bvaughn/task-runner/). I'd be happy to incorporate any feedback you have, if you could spare a few minutes to look. :)

#### Landon Schropp

It looks interesting. You code is clean and the documentation is great. However, I'm having trouble imagining the use case for using it. I could see a series of small, independent chunks of work that need to be executed in a complex combination and sequence. However, I generally wouldn't do something like that in a client web browser, and on the server I'd probably use a functional language or a queueing system. Who are you building your library for? What problem are you solving?

#### Brian Vaughn

I originally wrote it in hopes that one of the other feature-teams at Google (where I work) would be interested in using it for managing some of their complicated animation sequences. (Whether or not they will still remains to be seen.)

I'm continuing the work on it partly as an experiment- I think it's unique, and I wonder if other JavaScript engineers will find its uniqueness compelling.

I think the real selling feature will be the app-engine portion that I'm still developing. I've been tentatively planning on loosely pairing it with a framework like React or Riot- or even Angular, to be honest. In my experience, these frameworks are fantastic for DOM-manipulation/generation, but they kind of fall down when it comes to answering the question of "how do I structure a large application?". I think Task Runner *could* provide enough structure to make answering that question simple.

But this might all just be an unrealistic ambition on my part.

Thanks for taking a look and for sharing your thoughts. If any more ideas come to you- particularly ones like "I could see myself using a framework like that IF..."- I'd love to hear them. :)

### Bob Rogan

If you want to understand Asynchronous Java-script Object Management visit the ajoms.com web-site and peruse the AJOMS Framework documentation and then click on the example to see a sample of power of the AJOMS Framework in action.
Promises "Promises" Programming headaches with its unstructured and unreadable code.

### Bob Rogan

Come on now, the AJOMS Framework is the answer to all Client/Server Asynchronous Object Management, presentation, and communication website design needs.

### Hamabama

Such an elegant solution for multiple requests. Was looking for one. Thank you!

### Jabran Rafique

This makes it confusing now. Jack Archibald has explained here how jQuery deferred methods are not Promise/A+ compliant.

**Drew**

Update on this: as of 3.0, they are/will be. All the proper error and timing behavior should be supported now as well as the .catch() method to replace the semi-confusing .then(null, fn)

http://blog.jquery.com/2015/07/13/jquery-3-0-and-jquery-compat-3-0-alpha-versions-released/

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