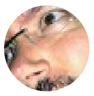
What are some advanced JavaScript techniques that you don't see often but should?

15 Answers



Mike Mikowski, SPA (UI/UX/server) architect and author
9.2k Views • Upvoted by Mattias Petter Johansson, Developer at Spotify

Create a *real* prototype-based object instead trying to create a psuedo-classical "Java-like" object.

Probably biggest mistake of JavaScript's design was the inclusion of the **new**keyword. Don't use **new**, use **Object.create()** instead. In other words,**Don't** do this:

```
function MyClass() {
    MyClass.prototype.method1 = function () { ... };
    MyClass.prototype.method2 = function () { ... };
    // ...
    MyClass.prototype.methodN = function () { ... };
    // ...
}
var instance = new MyClass();
```

Instead, create a prototype object and avoid any silly reference to MyClass.prototype.* at all:

```
var myProto, makeInstance, instance;

myProto = {
  method1 : function () { ... };
  method2 : function () { ... };
};

makeInstance = function () { return Object.create( myProto ); };

instance = makeInstance();
```

Now I realize this doesn't seem to provide significant advantage is in this trivial example. However, in larger projects this results in much more readable code because the word 'prototype' isn't 33% of your code. Take a look at some Google Closure Library "classes" if you want a feel for how awful it can get.

The **makeInstance** function can not only create an instance, but do other interesting and wonderful things, like keeping track of all instances in memory, and setting initial properties based on the current state of the application. One can also use a function closure (instead of an object literal) to create truly private methods for the prototype. And, perhaps most importantly, people won't confuse this with a Class-based object system. Which is good - because JavaScript's object *are not class based*.

Check out TypeBomb . Every element on screen has a corresponding prototype-based factory object with *makeInstance* and *destroyInstance* routines. Because of this approach, the model can easily answer questions about groups of objects (How many words on screen? How many are close to the bottom?). This is code that takes advantage of JavaScript's prototype-based objects instead of using misleading and verbose kludges to make them*seem* like Java objects.

Updated 15 Sep • View Upvotes

More Answers Below. **Related Questions**

What are some advanced PHP techniques that you don't see often but should?

Where can I learn about the more advanced techniques of Beautiful Soup?

Where can I learn advanced jQuery techniques?

Is "Pro Javascript Techniques" by John Resig still a recommended resource for advanced JavaScript education as of 2015?

JavaScript (programming language): What are some examples of things that aren't nodes?

John Babak, Software Engineer, expert in Web, JavaScript, Node.js
3.2k Views

This is not an advanced technique but is rarely seen. Deep property access can be compressed by using variables with references or strings.

Classes are often written in a very verbose way:

```
!function () {
    // ...
    function MyClass() {}
    MyClass.prototype.method1 = function () {};
    MyClass.prototype.method2 = function () {};
    // ...
    MyClass.prototype.methodN = function () {};
    // ...
}();
```

More concise and better for minification:

```
!function () {
    // ...
    // Note that the MyClass function definition is automatically hoisted.
    var _proto = MyClass.prototype;
    function MyClass() {}
    _proto.method1 = function () {};
    _proto.method2 = function () {};
    // ...
    _proto.methodN = function () {};
    // Some deprecated method could be aliased, too:
    _proto.methodX = _proto.methodDeprecated;
    // ...
}();
```

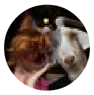
Even more concise if you have an `extend` function on your hands, though the following syntax requires keeping track of the trailing comma and limits the freedom of aliasing multiple names onto the same method:

```
var extend = require('util')._extend;
function MyClass() {}
extend(MyClass.prototype, {
    method1: function () {},
    method2: function () {},
    // ...
    methodN: function () {}
});
```

Another shortening for minification comes for long function names that are used frequently. See jQuery source for better examples. The following example is synthetic only to demonstrate the principle:

```
!function (global) {
    //< minifies to `!function(a){`

    var someObject = global.someObject;
    //< minifies to `var b=a.someObject;`

    var someLongMethodOrFunctionName_str = 'someLongMethodOrFunctionName';
    //< minifies to `var c='someLongMethodOrFunctionName';`

    someObject[someLongMethodOrFunctionName_str]();
    //< minifies to `b[c]();`

    // Below a lot of calls to `someLongMethodOrFunctionName`
    // should follow, otherwise the optimization is useless.
}(this);
```

Written 9 May • View Upvotes



Dustin Swan • Request Bio
1.5k Views

Observables. They're like Promises, but more powerful. Using the observable pattern (also called Functional reactive programming [1]) can really clean up your async JavaScript code. The most popular libraries seem to be RxJS , and Bacon.JS , but there are many out there. I happen to like RxJS.

Basically, Observables allow you to use good old functional programming (e.g. map, reduce, filter, scan) on *asynchronous data streams* (e.g. AJAX requests, click events). Then you can *subscribe* functions to these streams that will render your UI, update your models, generate more streams, etc. This allows for a more *declarative* programming style.

For a little *amuse-bouche*, watch this YouTube video by this Netflix Cross Team Tech Lead person about how they do async frontend development

And for a nice beginner tutorial, check out this gist by André Staltz.

[1] It is debated whether the FRP title actually applies to what libraries like RxJS are doing.

Written 14 May • View Upvotes

Spen Taylor, Web Developer
3.1k Views

I'm not sure if you'd consider this advanced, but it's a technique I've probably been abusing since I learned of it...

Using ternaries to call functions or variables conditionally.
For instance if you might be using Velocity.js and so instead of using "jQuery.animate()" you'll be using "jQuery.velocity()".

Now you might want to do this conditionally, perhaps using animate for certain devices, so rather than using an if/else statement every time you could write:

```
jQuery[(theCondition) ? 'velocity' : 'animate']({settings...});
```

or better still move the ternary out in to an app-wide scope to be reused...

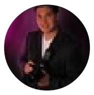
```
var animationMethod = (theCondition) ? 'velocity' : 'animate';
```

and just call like so:

```
jQuery[animationMethod]({settings...});
```

The idea of using a variable to hold an object's key reference as a string is something I've generally found quite useful!

Written 4 May • View Upvotes



Brennan Cheung, Programmer, Photographer, Entrepreneur
2.1k Views

To cast to boolean you can use:
`!!(expression).`

Any expression that is truthy becomes true. Any expression expression that is falsy (empty string, 0, null, undefined, etc) becomes false.

Another technique involves returning early for error conditions.

For example, if you have code that looks like:
```
if (condition) {
    // handle the success case
} else {
    // handle the error
}
```

You can convert it to:
```
if (!condition) return;
// handle the success case
```

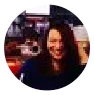
The benefit is that the code is a lot cleaner, shorter, and there is less indentation. It also separates errors from the normal logic. This is especially beneficial when there is multiple nested conditions.

These are used frequently but are neat tricks you can use.

Cast a signed integer to an unsigned integer:
`num >>> 0`

Set all bits (useful for creating masks):
`~0`

Written 12 May • View Upvotes



Sarah Federman, Web Designer, Developer
4.8k Views

You can use `bind()` for currying, which has been around for a while but you don't see it used much. It's clean and no libraries needed :)

Source: Some conversation I had with getify once upon a time, Currying in JavaScript using bind()

Written 9 May • View Upvotes

Rob Brown, been using Javascript for 18 years
4.8k Views • Upvoted by Mattias Petter Johansson, Developer at Spotify

Sharing a lot of code between client and server, also known as "Isomorphic Javascript." (thanks, Sarah Federman!) It's an extremely powerful thing that is allowed by node.js, but it is rarely used by most people, or only used for the most trivial things.

Updated 12 May • View Upvotes

Geoffrey Abdallah • Request Bio
1.9k Views

Utilizing more of the functional paradigm, taking advantage of .map and .reduce, and introducing immutable data structures into your program:facebook/immutable-js .

Written 11 May • View Upvotes

Dvid Silva, burrito.
2.6k Views • Upvoted by Simon Gardner, 15 years developing websites and content management apps.

Promises are not advanced, they're not very hard to learn, and they're extremely useful; and somehow people are afraid of them and are not using them.

I would say, if you're not using promises, you should, now!

Written 11 May • View Upvotes

Dennis Babkin, S/w developer, Investor, Aspiring scientist
34.8k Views

This is not really an advanced technique but a short *trick* you can use in JavaScript. As you are probably well aware, JavaScript internally uses floating-point registers to store numbers. So if you need a quick way to ensure that a small number is an *integer*, you can round it down with double *bitwise complement operator*, as such:

```
v = ~~v;
```

It will always return an *integer* (never a *NaN*) and will set v to 0 if it was not a number.

PS. Something to watch out with this approach is for the number to lie within the boundaries of a signed 32-

bit integer (or from -2147483648 to 2147483647, inclusively.) Otherwise use your usual Math.floor(). This limitation stems from the fact that we're technically doing a bitwise operation on the number (that's what implicitly converts it to an integer) and thus we have to "fit" within the internal JavaScript engine's bitness.

Written 11 May • View Upvotes

Dominik Guzei • Request Bio
1.5k Views • Upvoted by Mattias Petter Johansson, Developer at Spotify

One technique that I discovered recently is what I call "runtime-checked messages". So instead of calling a method on another object directly, or sending string-based events/commands that have no defined structure, you send messages that check their structure/payload/params themselves.

There are multiple benefits:

1. You decouple sender / receiver (nothing new -> events / event bus etc.)
2. Well defined contract between sender / receiver (messages are the contract)
3. Unit tests that are testing the real thing (correctness of messages) but dont require the receiving or sending part during the test.
4. Messages are class instances, not just anonymous object literals
5. Messages can check for anything from simple String, Number up to business ValueObjects. This simplifies your code, because you don't have to spread the checks around your codebase (DRY)

Basically this is what you get with statical type-checking compilers in other languages and it fixed one of the biggest headaches I experienced in big Javascript applications: refactoring!

Written 13 May • View Upvotes

Rohit Kumar • Request Bio
1.7k Views

I don't want to explain every key point as you can find better explanation on google. So here it goes :
1. scope of veriable global, function and closures
closures scope is very confusing one and you should be very clear of this as it helps making code a little tricky and can be managed to avoid memory leak

2. Prototypical inheritance :
its very tricky and different from classical inheritance. you must know this for using javascript's OOP's behavior.

3. AJAX :
I know that every one knows this but at the same time, it is one of the most important feature of javascript. You must know this in order to make responsive and one page website.

4. Memoization :
okay. this may not be the key point according to many one but caching data is one of the thing you must learn to make your website work on slow internet connection also.

5. Framework :
Frameworks are now a days everywhere. You gotta find a hell lot of time saved, when you switch to a framework based website from website written in plain JavaScript.

Written 4 May • View Upvotes

Torgeir Helgevold, Software Engineer
2k Views • Torgeir has 190+ answers in JavaScript (programming language).

My best advice for writing "advanced" JavaScript is as follows:
My take on the subject may not be exactly what you expect sine I am always trying to promote better modeling practices and less messy code. Key to achieving this, in my opinion, is through properly decoupled JavaScript where you keep your view (html) separate from your JavaScript objects. This will not only make your code reusable, but also very unit testable.

Written 4 May • View Upvotes

Sayed Ahad Abbas, Don't believe what you see; a lot more is happening in the background!
722 Views

Use of ternary operators is something I don't see in most of the libraries and hence I would say this is something one should use more often. Instead of using single line if-else statements or a chain of single line statements, you can use chained ternary operators. The best part with them is that you can call 'null' in case there is no else part!
This is something I do on a regular basis:

> condition_met ? call_function() : null;

I use it even in echo statements:

> console.log(my_condition ? 'Hi!' :(your_condition ?'Hello' :'Ñamaste'));

Another thing which people usually ignore is the use of 'use strict'. This is a new directive available in JavaScript 1.8.5 and later. This executes script in a strict mode and allows you to catch any loose errors. By loose I mean usage of anything undeclared previously, which could be undeclared variables, objects, functions, etc. This I specially helpful when you're developing the code and it ensures that when you would be delivering the code, there won't be any errors. While you can follow this, it is not suggested.

Written 16 May • View Upvotes

Peter Roca, I make software and furniture, sometimes both.
994 Views

This is probably more advice for developers coming to JS from another language, but using type-checking comparison operators should be something that you use virtually all of the time, specifically `===` or `!==` instead of `==` or `!=`