Clearing your Front End Job Interview — JavaScript

Yangshun Tay
Jul 5, 2017 · 14 min read

Photo credits: https://unsplash.com/photos/F_EfOSXh0sI

Unlike typical software engineer job interviews, front end job interviews have less emphasis on algorithms and have more questions on intricate knowledge and expertise about the domain — HTML, CSS, JavaScript, just to name a few areas.

While there are some existing resources to help front end developers in preparing for interviews, they aren't as abundant as materials for a software engineer interview. Among the existing resources, probably the most helpful question bank would be Front End Job Interview Questions. Unfortunately, I couldn't find many complete and satisfactory answers for these questions online, hence here is my attempt at answering them. Being an open source repository, the project can live on with the support of the community as the state of web evolves. This is the JavaScript section of my answers. The HTML section can be found in this Medium article while the CSS section can be found in this Medium article.

The repository for my answers on the JavaScript questions can be found on GitHub, and future improvements will be made there. Pull requests for suggestions and corrections are welcome!

yangshun/front-end-interview-handbook
front-end-interview-handbook - 🕸 Almost complete answers to "Front-end Job Interview...
github.com

Disclaimer: *I do not own any of the content in the answers. Some of them have been extracted from the references word-for-word, and only rephrased for clarity if necessary.*

. . .

Explain event delegation

Event delegation is a technique involving adding event listeners to a parent element instead of adding them to the descendant elements. The listener will fire whenever the event is triggered on the descendant elements due to event bubbling up the DOM. The benefits of this technique are:

- Memory footprint goes down because only one single handler is needed on the parent element, rather than having to attach event handlers on each descendant.
- There is no need to unbind the handler from elements that are removed and to bind the event for new elements.

References

- https://davidwalsh.name/event-delegate
- https://stackoverflow.com/questions/1687296/what-is-dom-event-delegation

Explain how `this` works in JavaScript

There's no simple explanation for `this`; it is one of the most confusing concepts in JavaScript. A hand-wavey explanation is that the value of `this` depends on how the function is called. I have read many explanations on `this` online, and I found Arnav Aggrawal's explanation to be the clearest. The following rules are applied:

1. If the `new` keyword is used when calling the function, `this` inside the function is a brand new object.
2. If `apply`, `call`, or `bind` are used to call/create a function, `this` inside the function is the object that is passed in as the argument.
3. If a function is called as a method, such as `obj.method()` — `this` is the object that the function is a property of.
4. If a function is invoked as a free function invocation, meaning it was invoked without any of the conditions present above, `this` is the global object. In a browser, it is the `window` object. If in strict mode (`'use strict'`), `this` will be `undefined` instead of the global object.
5. If multiple of the above rules apply, the rule that is higher wins and will set the `this` value.
6. If the function is an ES2015 arrow function, it ignores all the rules above and receives the `this` value of its surrounding scope at the time it is created.

For an in-depth explanation, do check out his article on Medium.

References

- https://codeburst.io/the-simple-rules-to-this-in-javascript-35d97f31bde3
- https://stackoverflow.com/a/3127440/1751946

Explain how prototypal inheritance works

This is an extremely common JavaScript interview question. All JavaScript objects have a `prototype` property, that is a reference to another object. When a property is accessed on an object and if the property is not found on that object, the JavaScript engine looks at the object's `prototype`. and the `prototype`'s `prototype` and so on, until it finds the property defined on one of the `prototype`s or until it reaches the end of the prototype chain. This behaviour simulates classical inheritance, but it is really more of delegation than inheritance.

References

- https://www.quora.com/What-is-prototypal-inheritance/answer/Kyle-Simpson
- https://davidwalsh.name/javascript-objects

What do you think of AMD vs CommonJS?

Both are ways to implement a module system, which was not natively present in JavaScript until ES2015 came along. CommonJS is synchronous while AMD (Asynchronous Module Definition) is obviously asynchronous. CommonJS is designed with server-side development in mind while AMD, with its support for asynchronous loading of modules, is more intended for browsers.

I find AMD syntax to be quite verbose and CommonJS is closer to the style you would write import statements in other languages. Most of the time, I find AMD unnecessary, because if you served all your JavaScript into one concatenated bundle file, you wouldn't benefit from the async loading properties. Also, CommonJS syntax is closer to Node style of writing modules and there is less context-switching overhead when switching between client side and server side JavaScript development.

I'm glad that with ES2015 modules, that has support for both synchronous and asynchronous loading, we can finally just stick to one approach. Although it hasn't been fully rolled out in browsers and in Node, we can always use transpilers to convert our code.

References

- https://auth0.com/blog/javascript-module-systems-showdown/
- https://stackoverflow.com/questions/16521471/relation-between-commonjs-amd-and-requirejs

Explain why the following doesn't work as an IIFE: `function foo(){ }();`. What needs to be changed to properly make it an IIFE?

IIFE stands for Immediately Invoked Function Expressions. The JavaScript parser reads `function foo(){ }();` as `function foo(){ }` and `();`, where the former is a function declaration and the latter (a pair of brackets) is an attempt at calling a function but there is no name specified, hence it throws `Uncaught SyntaxError: Unexpected token )`.

Here are two ways to fix it that involves adding more brackets: `(function foo(){ })()` and `(function foo(){ }())`. These functions are not exposed in the global scope and you can even omit its name if you do not need to reference itself within the body.

References

- http://lucybain.com/blog/2014/immediately-invoked-function-expression/

What's the difference between a variable that is: `null`, `undefined` or undeclared? How would you go about checking for any of these states?

Undeclared variables are created when you assign to a value to an identifier that is not previously created using `var`, `let` or `const`. Undeclared variables will be defined globally, outside of the current scope. In strict mode, a `ReferenceError` will be thrown when you try to assign to an undeclared variable. Undeclared variables are bad just like how global variables are bad. Avoid them at all cost! To check for them, wrap its usage in a `try` / `catch` block.

```
function foo() {
  x = 1;   // Throws a ReferenceError in strict mode
}

foo()
console.log(x) // 1
```

A variable that is `undefined` is a variable that has been declared, but not assigned a value. It is of type `undefined`. If a function does not return any value as the result of executing it is assigned to a variable, the variable also has the value of `undefined`. To check for it, compare using the strict equality (`===`) operator or `typeof` which will give the `'undefined'` string. Note that you should not be using the abstract equality operator to check, as it will also return `true` if the value is `null`.

```
var foo;
console.log(foo); // undefined
console.log(foo === undefined); // true
console.log(typeof foo === 'undefined'); // true

console.log(foo == null); // true. Wrong, don't use this to check!
```

```
function bar() {}
var baz = bar();
console.log(baz); // undefined
```

A variable that is `null` will have been explicitly assigned to the `null` value. It represents no value and is different from `undefined` in the sense that it has been explicitly assigned. To check for `null,` simply compare using the strict equality operator. Note that like the above, you should not be using the abstract equality operator (`==`) to check, as it will also return `true` if the value is `undefined` .

```
var foo = null;
console.log(foo === null); // true

console.log(foo == undefined); // true. Wrong, don't use this to
check!
```

As a personal habit, I never leave my variables undeclared or unassigned. I will explicitly assign `null` to them after declaring, if I don't intend to use it yet.

References

- https://stackoverflow.com/questions/15985875/effect-of-declared-and-undeclared-variables
- https://developer.mozilla.org/en/docs/Web/JavaScript/Reference/Global_Objects/undefined

Difference between: `function Person(){}`, `var person = Person()`, and `var person = new Person()`?

This question is pretty vague. My best guess at its intention is that it is asking about constructors in JavaScript. Technically speaking, `function Person(){}` is just a normal function declaration. The convention is use PascalCase for functions that are intended to be used as constructors.

`var person = Person()` invokes the `Person` as a function, and not as a constructor. Invoking as such is a common mistake if it the function is intended to be used as a constructor. Typically, the constructor does not return anything, hence invoking the constructor like a normal function will return `undefined` and that gets assigned to the variable intended as the instance.

`var person = new Person()` creates an instance of the `Person` object using the `new` operator, which inherits from `Person.prototype`. An alterative would be to use `Object.create`, such as: `Object.create(Person.prototype)`.

```
function Person(name) {
  this.name = name;
}

var person = Person('John');
console.log(person); // undefined
console.log(person.name); // Uncaught TypeError: Cannot read
property 'name' of undefined

var person = new Person('John');
console.log(person); // Person { name: "John" }
console.log(person.name); // "john"
```

References

- https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators/new

What's the difference between `.call` and `.apply`?

Both `.call` and `.apply` are used to invoke functions and the first parameter will be used as the value of `this` within the function. However, `.call` takes in a comma-separated arguments as the next arguments while `.apply` takes in an array of arguments as the next argument. An easy way to remember this is C for `call` and comma-separated and A for `apply` and array of arguments.

```
function add(a, b) {
  return a + b;
}

console.log(add.call(null, 1, 2)) // 3
console.log(add.apply(null, [1, 2])) // 3
```

Explain `Function.prototype.bind`.

Taken word-for-word from MDN:

> The `bind()` method creates a new function that, when called, has its this keyword set to the provided value, with a given sequence of arguments preceding any provided

> when the new function is called.

In my experience, it is most useful for binding the value of `this` in methods of classes that you want to pass into other functions. This is frequently done in React components.

References

- https://developer.mozilla.org/en/docs/Web/JavaScript/Reference/Global_objects/Function/bind

When would you use `document.write()`?

`document.write()` writes a string of text to a document stream opened by `document.open()`. When `document.write()` is executed after the page has loaded, it will call `document.open` which clears the whole document (`<head>` and `<body>` removed!) and replaces the contents with the given parameter value in string. Hence it is usually considered dangerous and prone to misuse.

There are some answers online that explain `document.write()` is being used in analytics code or when you want to include styles that should only work if JavaScript is enabled. It is even being used in HTML5 boilerplate to load scripts in parallel and preserve execution order! However, I suspect those reasons might be outdated and in the modern day, they can be achieved without using `document.write()`. Please do correct me if I'm wrong about this.

References

- https://www.quirksmode.org/blog/archives/2005/06/three_javascrip_1.html
- https://github.com/h5bp/html5-boilerplate/wiki/Script-Loading-Techniques#documentwrite-script-tag

Have you ever used JavaScript templating? If so, what libraries have you used?

Yes. Handlebars, Underscore, Lodash, AngularJS and JSX. I disliked templating in AngularJS because it made heavy use of strings in the directives and typos would go uncaught. JSX is my new favourite as it is closer to JavaScript and there is barely and syntax to be learnt. Nowadays, you can even use ES2015 template string literals as a quick way for creating templates without relying on third-party code.

```
const template = `<div>My name is: ${name}</div>`;
```

However, do beware of a potential XSS in the above approach as the contents are not escaped for you, unlike in templating libraries.

What is the difference between `==` and `===`?

`==` is the abstract equality operator while `===` is the strict equality operator. The `==` operator will compare for equality after doing any necessary type conversions. The `===` operator will not do type conversion, so if two values are not the same type `===` will simply return `false`. When using `==`, funky things can happen, such as:

```
1 == '1' // true
1 == [1] // true
1 == true // true
0 == '' // true
0 == '0' // true
0 == false // true
```

My advice is never to use the `==` operator, except for convenience when comparing against `null` or `undefined`, where `a == null` will return `true` if `a` is `null` or `undefined`.

```
var a = null;
console.log(a == null); // true
console.log(a == undefined); // true
```

References

- https://stackoverflow.com/questions/359494/which-equals-operator-vs-should-be-used-in-javascript-comparisons

Make this work:

```
duplicate([1,2,3,4,5]); // [1,2,3,4,5,1,2,3,4,5]

function duplicate(arr) {
  return arr.concat(arr);
}
```

Why is it called a Ternary expression, what does the word "Ternary" indicate?

"Ternary" indicates three, and a ternary expression accepts three operands, the test condition, the "then" expression and the "else" expression. Ternary expressions are not specific to JavaScript and I'm not sure why it is even in this list.

References

- https://developer.mozilla.org/en/docs/Web/JavaScript/Reference/Operators/Conditional_Operator

What is `"use strict"`; ? What are the advantages and disadvantages to using it?

'use strict' is a statement used to enable strict mode to entire scripts or individual functions. Strict mode is a way to opt in to a restricted variant of JavaScript.

Advantages:

- Makes it impossible to accidentally create global variables.
- Makes assignments which would otherwise silently fail to throw an exception.
- Makes attempts to delete undeletable properties throw (where before the attempt would simply have no effect).
- Requires that function parameter names be unique.
- `this` is undefined in the global context.
- It catches some common coding bloopers, throwing exceptions.
- It disables features that are confusing or poorly thought out.

Disadvantages:

- Many missing features that some developers might be used to.
- No more access to `function.caller` and `function.arguments` .
- Concatenation of scripts written in different strict modes might cause issues.

Overall, I think the benefits outweigh the disadvantages, and I never had to rely on the features that strict mode blocks. I would recommend using strict mode.

References

- http://2ality.com/2011/10/strict-mode-hatred.html
- http://lucybain.com/blog/2014/js-use-strict/

Create a for loop that iterates up to `100` while outputting "fizz" at multiples of `3`, "buzz" at multiples of `5` and "fizzbuzz" at multiples of `3` and `5`.

Check out this version of FizzBuzz by Paul Irish.

```
for (let i = 1; i <= 100; i++) {
  let f = i % 3 == 0, b = i % 5 == 0;
  console.log(f ? (b ? 'FizzBuzz' : 'Fizz') : (b ? 'Buzz' : i));
}
```

I would not advise you to write the above during interviews though. Just stick with the long but clear approach. For more wacky versions of FizzBuzz, check out the reference link below.

References

- https://gist.github.com/jaysonrowe/1592432

Explain what a single page app is and how to make one SEO-friendly.

The below is taken from the awesome Grab Front End Guide, which coincidentally, is written by me!

Web developers these days refer to the products they build as web apps, rather than websites. While there is no strict difference between the two terms, web apps tend to be highly interactive and dynamic, allowing the user to perform actions and receive a response for their action. Traditionally, the browser receives HTML from the server and renders it. When the user navigates to another URL, a full-page refresh is required and the server sends fresh new HTML for the new page. This is called server-side rendering.

However in modern SPAs, client-side rendering is used instead. The browser loads the initial page from the server, along with the scripts (frameworks, libraries, app code) and stylesheets required for the whole app. When the user navigates to other pages, a page refresh is not triggered. The URL of the page is updated via the HTML5 History API. New data required for the new page, usually in JSON format, is retrieved by the browser via AJAX requests to the server. The SPA then dynamically updates the page with the data via JavaScript, which it has already downloaded in the initial page load. This model is similar to how native mobile apps work.

The benefits:

- The app feels more responsive and users do not see the flash between page navigations due to full-page refreshes.
- Fewer HTTP requests are made to the server, as the same assets do not have to be downloaded again for each page load.
- Clear separation of the concerns between the client and the server; you can easily build new clients for different platforms (e.g. mobile, chatbots, smart watches) without having to modify the server code. You can also modify the technology stack on the client and server independently, as long as the API contract is not broken.

The downsides:

- Heavier initial page load due to loading of framework, app code, and assets required for multiple pages.
- There’s an additional step to be done on your server which is to configure it to route all requests to a single entry point and allow client-side routing to take over from there.
- SPAs are reliant on JavaScript to render content, but not all search engines execute JavaScript during crawling, and they may see empty content on your page. This inadvertently hurts the Search Engine Optimization (SEO) of your app. However, most of the time, when you are building apps, SEO is not the most important factor, as not all the content needs to be indexable by search engines. To overcome this, you can either server-side render your app or use services such as Prerender to “render your javascript in a browser, save the static HTML, and return that to the crawlers”.

References

- https://github.com/grab/front-end-guide#single-page-apps-spas
- http://stackoverflow.com/questions/21862054/single-page-app-advantages-and-disadvantages
- http://blog.isquaredsoftware.com/presentations/2016-10-revolution-of-web-dev/
- https://medium.freecodecamp.com/heres-why-client-side-rendering-won-46a349fadb52

What are some of the advantages/disadvantages of writing JavaScript code in a language that compiles to JavaScript?

Some examples of languages that compile to JavaScript include CoffeeScript, Elm, ClojureScript, PureScript and TypeScript.

Advantages:

- Fixes some of the longstanding problems in JavaScript and discourages JavaScript anti-patterns.
- Enables you to write shorter code, by providing some syntactic sugar on top of JavaScript, which I think ES5 lacks, but ES2015 is awesome.
- Static types are awesome (in the case of TypeScript) for large projects that need to be maintained over time.

Disadvantages:

- Require a build/compile process as browsers only run JavaScript and your code will need to be compiled into JavaScript before being served to browsers.
- Debugging can be a pain if your source maps do not map nicely to your pre-compiled source.
- Most developers are not familiar with these languages and will need to learn it. There's a ramp up cost involved for your team if you use it for your projects.
- Smaller community (depends on the language), which means resources, tutorials, libraries and tooling would be harder to find.

- IDE/editor support might be lacking.
- These languages will always be behind the latest JavaScript standard.

Practically, ES2015 has vastly improved JavaScript and made it much nicer to write. I don't really see the need for CoffeeScript these days.

References

- https://softwareengineering.stackexchange.com/questions/72569/what-are-the-pros-and-cons-of-coffeescript

What is event loop? What is the difference between call stack and task queue?

The event loop is a single-threaded loop that monitors the call stack and checks if there is any work to be done in the task queue. If the call stack is empty and there are callback functions in the task queue, a function is dequeued and pushed onto the call stack to be executed.

If you haven't already checked out Philip Robert's talk on the Event Loop, you should. It is one of the most viewed videos on JavaScript.

References

- https://2014.jsconf.eu/speakers/philip-roberts-what-the-heck-is-the-event-loop-anyway.html
- http://theproactiveprogrammer.com/javascript/the-javascript-event-loop-a-stack-and-a-queue/

Explain the differences on the usage of `foo` between `function foo() {}` and `var foo = function() {}`

The former is a function declaration while the latter is a function expression. The key difference is that function expressions are not hoisted (they have the same hoisting behaviour as variables). For more explanation on hoisting, refer to the question above on hoisting. If you try to invoke a function expression before it is defined, you will get an `Uncaught TypeError: XXX is not a function` error.

Function Declaration

```
foo(); // 'FOOOOO'
function foo() {
  console.log('FOOOOO');
}
```

Function Expression

```
foo(); // Uncaught TypeError: foo is not a function
var foo = function() {
  console.log('FOOOOO');
}
```

References

- https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Statements/function

. . .

That's it for now! The repository for the above content can be found on GitHub, and future improvements will be made there. Pull requests for suggestions and corrections are welcome!

If you enjoyed this article, please don't forget to tap ❤. You can also follow me on GitHub and Twitter.

yangshun/front-end-interview-handbook
front-end-interview-handbook - 🕸 Almost complete answers to "Front-end Job Interview…
github.com

Other Answers

- http://flowerszhong.github.io/2013/11/20/javascript-questions.html

JavaScript Web Development Technology Front End Development Interview