Interviewing a front-end developer

Dec 30th, 2013 by Alex MacCaw

Part of my role at Twitter and Stripe involved interviewing front-end engineering candidates. We were given a fair amount of discretion on how we went about interviewing, and I developed a few different sets of questions that I thought would be interesting to share.

I'd like to prefix all of this with the caveat that hiring is extremely hard, and figuring out if someone is a good fit within 45 minutes is a demanding task. The problem with interviews is that everyone tries to hire themselves. Anybody that aces my interview probably thinks a lot like me, and that's not necessarily a good thing. As it is, I've been pretty hit and miss in my decisions so far. However, I believe this approach to be a good start.

Ideally the candidate has a really full **GitHub** 'resume', and we can just go back through their open source projects together. I usually browse their code and ask them questions about particular design decisions. If the candidate has an excellent track record, then the interview largely moves into whether they'd be a good social fit for the team. Otherwise, I move on to the coding questions.

My interviews are very practical and are entirely coding. No abstract or algorithmic questions are asked — other interviewers can cover those if they choose to, but I think their relevance to front-end programming is debatable. The questions I ask may seem simple, but each set is designed to give me an insight into a particular aspect of JavaScript knowledge.

No whiteboards are used. If the candidate brings their own laptop they can type away on that. Otherwise they can just use mine. They can use whatever editor they'd like, and I often test the output of their programs directly in Chrome's console.

Section 1: Object Prototypes

We start out simple. I ask the candidate to define a `spacify` function which takes a string as an argument, and returns the same string but with each character separated by a space. For example:

```
spacify('hello world') // => 'h e l l o  w o r l d'
```

Although this question may seem rather simple, it turns out that it's a good place to start, especially with unvetted phone candidates — some of whom claimed to know JavaScript, but in actuality didn't know how to write a single function.

The correct answer is the following. Sometimes candidates would use a `for` loop, also an acceptable answer.

```
function spacify(str) {
  return str.split('').join(' ');
}
```

The follow-up question to this, is to ask candidates to place the `spacify` function directly on the `String` object, for example:

```
'hello world'.spacify();
```

Asking this question gives me an insight into the candidate's basic knowledge of function prototypes. Often this would result in an interesting discussion about the perils of defining properties directly on prototypes, especially on `Object`. The end result looks like this:

```
String.prototype.spacify = function(){
  return this.split('').join(' ');
};
```

At this point, I usually ask candidates to explain the difference between a function expression and a function declaration.

Section 2: Arguments

Next I'd ask a few simple questions designed to show me how well candidates understood the `arguments` object. I'd start off by calling the as-yet undefined `log` function.

```
log('hello world')
```

Then I'd ask the candidates to define `log` so that it proxies its string argument to `console.log()`. The correct answer is something along these lines, but better candidates will often skip directly to using `apply`.

```
function log(msg){
  console.log(msg);
}
```

Once that's defined I change the way I call `log`, passing in multiple arguments. I make clear that I expect `log` to take an arbitrary number of arguments, not just two. I also hint to the fact that `console.log` also takes multiple arguments.

```
log('hello', 'world');
```

Hopefully your candidate will jump straight to using `apply`. Sometimes they'll get tripped up on the difference between `apply` and `call`, and you can nudge them to the correct direction. Passing the `console` context is also important.

```
function log(){
  console.log.apply(console, arguments);
};
```

I next ask candidates to prefix all logged messages with `"(app)"`, for example:

```
'(app) hello world'
```

Now this is where it gets a bit trickier. Good candidates will know that `arguments` is a pseudo array, and to manipulate it we need to convert it into a standard array first. The common pattern for this is using `Array.prototype.slice`, like in the following:

```
function log(){
  var args = Array.prototype.slice.call(arguments);
  args.unshift('(app)');

  console.log.apply(console, args);
};
```

Section 3: Context

The next set of questions are designed to reveal a candidate's knowledge of JavaScript context and `this`. I first define the following example. Notice the `count` property is being read off the current context.

```
var User = {
  count: 1,

  getCount: function() {
    return this.count;
  }
};
```

I then define the following few lines, and ask the candidate what the output of the log will be.

```
console.log(User.getCount());

var func = User.getCount;
console.log(func());
```

In this case, the correct answer is `1` and `undefined`. You'd be amazed how many people trip up on basic context questions like this. `func` is called in the context of `window`, so loses the `count` property. I explain this to the candidate, and ask how we could ensure the context of `func` was always bound to `User`, so that it would correctly return `1`.

The correct answer is to use `Function.prototype.bind`, for example:

```
var func = User.getCount.bind(User);
console.log(func());
```

I usually then explain that this function isn't available in older browsers, and ask the candidate to shim it. A lot of weaker candidates will struggle with this, but it's important that anyone you hire has a comprehensive knowledge of `apply` and `call`.

```
Function.prototype.bind = Function.prototype.bind || function(context){
  var self = this;

  return function(){
    return self.apply(context, arguments);
  };
}
```

Extra points if the candidate shims `bind` so that it uses the browser's native version if available. At this point, if the candidate is doing really well, I'll ask them to implement currying arguments.

Section 4: Overlay library

In the last part of the interview I ask the candidates to build something practical, usually an 'overlay' library. I find this useful, as it demonstrates a full front-end stack: HTML, CSS and JavaScript. If a candidate excels at the previous part of the interview, I move to this question as soon as possible.

It's up to the candidate as to the exact implementation, but there are a couple of key things to look out for:

It's much better to use `position: fixed` instead of `position: absolute` for overlay covers, which will ensure that the overlay encompasses the entire window even if it's scrolled. I'd definitely prompt for this if the candidate misses it, and ask them the difference between `fixed` and `absolute` positioning.

```
.overlay {
  position: fixed;
  left: 0;
  right: 0;
  bottom: 0;
  top: 0;
  background: rgba(0,0,0,.8);
}
```

How they choose to center content inside the overlay is also revealing. Some of the candidates might choose to use CSS and absolute positioning, which is possible if the content is a fixed width and height. Otherwise may choose to use JavaScript for positioning.

```
.overlay article {
  position: absolute;
  left: 50%;
  top: 50%;
  margin: -200px 0 0 -200px;
  width: 400px;
  height: 400px;
}
```

I also ask them to ensure the overlay is closed if it's clicked, which serves as a good segway into a discussion about the different types of event propagation. Often candidates will slap a click event listener directly on the overlay, and call it a day.

```
$('.overlay').click(closeOverlay);
```

This works, until you realize that click events on the overlay's children will also close the overlay — behavior that's definitely not desirable. The solution is to check the event's targets, and make sure the event wasn't propagated, like so:

```
$('.overlay').click(function(e){
  if (e.target == e.currentTarget)
    closeOverlay();
});
```

Other ideas

Clearly these questions only cover a tiny slice of front-end knowledge, and there are a lot of other areas you could be asking about, such as performance, HTML5 APIs, AMD vs CommonJS modules, constructors, types and the box model. I often mix and match question topics, depending on the interviewees interests.

I also recommend looking at the excellent **Front-end Developer Interview Questions** repository for ideas, and also any of the JavaScript behavior documented in **JavaScript Garden**.