Single Page Applications - The failure of the web

Nathan Jervis

January 9, 2014

Contents

1 Background

The web in the current age is hailed as the platform that exists everywhere. If you need an application to work everywhere, and not worry about rewriting your application to support new platforms, you generally choose the browser as your targetted platform. This is enhanced even more by mobile frameworks which execute web based code within the application, and provide access to mobile phone features from within the app[1].

The only problem is navigating to new pages requires loading the page from the server, constructing a DOM[2] and downloading styles and images. While web browsers are becoming faster, this still takes a noticeable amount of time, especially on a slow internet connection.

Single Page Applications solve this problem by downloading the entire website all in one go. Clicking on a link no longer requires fetching any html or styles, and the DOM is already built. Applications such as G-Mail[**?**] are built completely upon this concept, and it gives them the feel of a native application.

[1] Phonegap is one of the many frameworks which allow you to do this

[2] Document Object Model - A tree which represents the HTML document and allows access to it

h!

Action	Time Taken	Notes
DNS	50ms	
TCP handshake	80ms	1 round-trip
SSL handshake	160ms	2 round-trips
Request to Server	40ms	
Server Processing	100ms	
Reponse from Server	40ms	
Total	470ms	

Table 1: Time taken for a typical web request

2 Why is SPA needed?

If we look at Table 1[**?**] we can see that a typical request takes about 470ms. This is just to load the content, and doesn't include the time the client needs to parse the page, construct the DOM and determine the styling.

Obviously this is a problem for applications since half a second is a long time to wait when the user clicks a button. SPA's avoid this in 2 ways.

2.1 Remove overhead

If you look at any web application you will usually see quite a bit of code that exists on every page. The navigation bar is common on almost every web application, and exists on each page. The there's often sidebars, headers and footers that exist on each page. With regular web sites, this data gets sent with each request, which is unneccessary and slows down both the connection and the construction.

Any web programmer should be familiar with how to include html files, but this processing is only done server-side. This means that all the data is loaded each time.

Many sites have a layout file where they include all the data that doesn't change across pages, and then have a single content section that does change. If we look at a typical informational website for a business, we'll see a layout file that includes a nav bar, header, footer, several scripts and some stylesheets (often with a framework or 2). Looking at the code that was delivered to the client just for this basic layout that doesn't change, it's often much more than the code for the actual content.

With Single Page Applicationthis layout file is only sent once because the page is only loaded once. Where the duplicated content is a signficant amount of the page, or a slow connection is used, this means a very significant speed improvement.

2.2 Pre-loaded Structure

The structure and style of the application is loaded all at once on the initial page load. This means that once the page has loaded it doesn't need to load any more structure or style. This means that the user has to wait longer to get the application started, but once it's started there won't be as much waiting. This increased startup time isn't a problem for web sites that stay open all the time, as e-mail and social media sites do, so these are ideal targets for Single Page Applicationwhereas a page that's opened and then closed without any interaction (for instance a blog page) performs worse as a Single Page Application.

3 SPDY - Protocol

4 SPDY - Multi-Page Applications

In this section we can see how SPDY can be used to increase the performance of web sites so that the performance of a Single Page Applicationcan be achieved with idiomatic HTML, not requiring the programmer to use frameworks that distort html and javascript to get what they want.

4.1 HTML include

To address the first problem where nav bars, headers and footers are sent along with every request, a simple HTML include command can be created. Now there are many frameworks which do in fact do this, but they create a very significant overhead for including files, and it can be done with almost zero overhead.

The substantial overhead doesn't come from the concept, because there should be no overhead to downloading 2 smaller sizes instead of one large one[3]. The overhead rather comes from the fact that the client only knows about it after it has parsed and executed that section of code. HTTP only allows one request per connection, so when it comes across the line to download the new piece it needs to open a new connection.

Even if the programmer is willing to put up with this overhead, it is very complex to deal with waiting for a proper onload event, since the onload event is fired once the page has loaded, but doesn't wait for all the client side includes to run.

By adding the HTML include, we can solve the fisrt problem and the onload issue. The browser can deal with making sure all the pieces have loaded before firing onload, and it doesn't need to parse and execute javascript before it realizes it needs the file.

The syntax is not important, but here's a suggestion that I think works well:

[3]Or at the very least the overhead should be like the overhead of having 2 files on a harddrive instead of one large one

```
<link href="navBar.html" rel="html" type="text/html" />
```

Another benefit of having the HTML include tag is that it's possible for the browser to not only cache the source, but also the DOM and computed styles. If it sees that a second page includes all the same files, it can just re-use the dom from that page.

Splitting your code among multiple files is such a critical feature in a language that it almost seems sloppy for HTML to not already support this. Indeed naively it seems like a very simple and useful feature to add.

The reason it hasn't been implemented however is because of HTTP. HTTP as mentioned only allows a single request per connection, so it requires a new connection for each request, and each included html file would add a significant amount of time to the page load. Rather programmers are encouraged to do it on the server side, so only request is needed.

This is where SPDY comes into play. SPDY allows mutliple requests per connection, even simulataneous ones. This means the client can download the pieces immediately with no overhead.

This still leaves the problem that the client has to wait until it finds the html include tag before it can download it, but that can be solved with another feature of SPDY. With SPDY, the server can actually iniate a request, and so it can send all the files that are included in the document within that connection. That way the client will recieve all the data at once, with no delay, and very minimal overhead.

It may be better for the server to just use the hint function though, to tell the client that it will need those files, that way the client can decide whether or not to fetch them (perhaps it has them in cache already).

4.2 Multi-Page Applications

Now that we can seperate our html file client side, we can create an application that doesn't have all the overhead we had initially. This is because the client can cache the pieces of the HTML from previous pages, and re-use those.

We are however still left with the fact that when a client navigates to a new page they will need to download the content for that page. It should be noted however that this is also a problem in Single Page Applicationsince the content for the page must be loaded. The different is loading the content is much faster than loading the entire page, but with the optimizations we've made, this is no longer the case.

Single Page Applications often pre-load content for other pages, or cache this content in order to improve performance. This is an action we are also able to do however, as caching is a problem the browser already deals with, and pre-loading can be done either be the client, or through hints from the server.