Evergreen Googlebot with Chromium rendering engine: What technical SEOs need to know

It’s been an exciting week with important announcements from the stage at the 2019 Google I/O event. Probably the most impactful announcement is that Google has now committed to regularly updating its Googlebot crawl service to begin using the most recent stable version of their headless Chromium rendering engine. This is a significant leap forward with more than 1,000 features now supported over the previous version.

Nearly all the new feature support is modern JavaScript syntax officially called ECMAScript (ES6). If you are a JavaScript developer, you really want to use the latest version of the language for access to syntactic sugar that continually appears as the language matures. It’s true that if you’re a vanilla JavaScript user, or if you favor one of the modern reactive frameworks, many neat new features come from developers who recommend better patterns for blocks of commonly written code.

One basic example is to add a value to an array, a very common thing to do using push():

```
<script> names = [ 'Amy', 'Bruce', 'Chris' ];
names.push('David');
</script>
```

Reactivity in a Nutshell

In the example above, an array of names is defined and assigned 3 values: Amy, Bruce, and Chris. Then David is added to the list using the push() method. With modern reactive

frameworks mutation of values can trigger ‘diff’ evaluations of a page DOM against a newer ‘virtual DOM’ by the framework, and since the array values differ, page values can be updated by JavaScript without reloading the browser window.

Reactivity in web-facing applications is where JavaScript has really added to our capabilities, and where our capabilities continue to advance as modern JavaScript further evolves on the server and in the browser. It gets tricky to keep track of JavaScript written for the server versus JavaScript that gets shipped to the browser. For example, with ES6 you can do the following, including the ability to use ‘let’ (and ‘const’) in definition statements:

```
<script> let names = [ 'Amy', 'Bruce', 'Chris' ]; names = [...names, 'David'];
</script>
```

Backward Compatibility

The names array mutation above uses a newer ‘spread operator’ syntax [...names] to represent current values of the names array, and then adds David using an assignment operation instead of the push() method. The newer syntax is not compatible with Chrome 41, and therefore would not work prior to Googlebot’s update to Chrome 74. For developers it is like death by a thousand cuts to have to write or transpile ES6 down for backward compatibility.

Now modern JavaScript syntax will largely start to work straight out of the box with Googlebot and there are literally dozens of new features available such as the one above. Just be aware that Bing and DuckDuckGo (as well as social share crawlers) may not be able to interpret ES6 syntax.

Real-Life Example

The Svelte framework was recently significantly updated and revised to version 3. With this major overhaul came more precisely triggered assignment-based page reactivity. There's a fun viral video about it going around. Having to write or transpile the 'names' array code to older push() syntax for Google in Svelte requires an extra step because push() adds values to an array but it isn't a variable assignment operation, which is necessary to trigger page reactivity in Svelte 3.

```
<script> let names = [ 'Amy', 'Bruce', 'Chris' ]; names.push('David'); names = names; // To trigger Svelte reactivity
</script>
```

It's easy to see why now being able to use ES6:

```
<script> names = [...names, 'David'];
</script>
```

…is more developer friendly for Svelte users than before.

Evergreen Chromium rendering

Now that Googlebot's evergreen Chromium rendering engine can be counted on, React, Angular, Vue, Svelte 3, and vanilla JavaScript users can worry a little less about polyfills specific to Chrome 41 and writing or transpiling down ES6 syntax in projects anymore. Concerns still exist, however. You need to test and make sure the rendering engine is behaving the way you anticipate. Google is more guarded about exposing its resources than a user's browser would be.

Google recommends that users check out the documentation to find references to Google's Web Rendering Service (WRS) instances: basically Chromium 74, currently, in products like the mobile-friendly test and the URL Inspection Tool. For

example, a Geo location script might ask for browser location services. Google's rendering engine doesn't expose that API. These kinds of exceptions in your JavaScript may halt your indexing.

Tracking Googlebot

If you're still tracking visits from older versions of Chrome in your server logs, eventually they will update the user-agent string to reflect the version of Chrome they are running. Also, keep in mind that Google is a fairly large and dispersed company with divisions that have varying access to its network resources. A particular department might have settings to modify in order to begin using the new Chrome engine, but it stands to reason that everything will be using it very soon, especially for critical Web crawling services.

Technical SEO Advice

What does this mean for technical SEOs? There will be fewer critical indexing issues to point out for sites running modern JavaScript. Traditional advice, however, will remain largely intact. For example, the new rendering engine does not shortcut the indexing render queue for reactive code. That means sites running React, Angular, or Vue etc. are still going to be better off pre-rendering relatively static sites, and best off server-side rendering (SSR) truly dynamic sites.

The nice thing about being a Technical SEO is we get to advise developers about practices that should align with Googlebot and that mostly they ought to be doing in the first place. The nice thing about being a SEO Developer is there's a never-ending river of exciting modern code to play with, especially with Google now caught up with Chromium 74. The only drawback is evergreen Chromium Googlebot doesn't help you with Bing, DuckDuckGo, or social media sharing crawlers.

That’s A Pretty Big Drawback

The more things change the more they stay the same. You should still advise clients about pre-rendering and SSR. This ensures that no matter what user-agent you’re dealing with, it will receive rendered content for search or sharing. The predicament we find ourselves in is that if the planned application has a huge volume of reactive parts to it, for example constantly updating sports scores or stock market prices, we must do reactivity and SSR alone won’t work.

That’s when it’s necessary to do SSR and ship custom JavaScript for deferred hydration, similar to code-splitting. Basically, the complete HTML is shipped as fully rendered at the server, and then JavaScript takes care of updating the reactivity parts. If JavaScript doesn’t render in Bing or DuckDuckGo, then it’s all right because you already shipped fully rendered HTML. This can seem excessive but keep in mind that the search engine will only ever be able to represent rankings for your page in the state it was at a particular point in time, anyway.

Why Such Reactivity?

SSR can accomplish the SEO rendering feat across user-agents for you, and user browsers can run JavaScript for reactive features. But why bother? If you are using a reactive framework just because you can, maybe you didn’t need to in the first place. If you want to avoid all the trouble and expense of having myriad complex details to manage when the nature of your site doesn’t require much reactivity, then it’s a really good idea to build static sites using a strategy with pre-rendering if necessary, or write vanilla JavaScript for the feature or two which may actually require reactivity.

Server Side Rendering

If you think server-side rendering is a piece of cake, read a post that describes some of the horrors you might encounter before you charge in, especially if you're trying to retrofit a pre-existing application. In short, you should be writing universal JavaScript and it gets complex quickly including security implications. Luckily, there is also a terrific new set of nicely written posts that comprise a fairly thorough React tutorial if you're working from scratch. We highly recommended reading it to supplement the official React guide.

A New Hope

Things move quickly and keeping up can be tough, even for Google. The news that it has updated to Chrome 74 for rendering more of the modern Web is long overdue. It's important that we know it intends to improve Googlebot to within weeks of the consumer version of Chrome releases. We can now test more code using local software to make sure our sites work with Googlebot. A very intriguing new paradigm for reactivity is Svelte. Svelte has a SSR output mode that you can test directly in its tutorial REPL. Svelte brings us reactivity that is closer to vanilla JavaScript than others, a real achievement.

About The Author

Detlef Johnson is Editor at Large for Third Door Media. He writes a column for Search Engine Land entitled "Technical SEO for Developers." Detlef is one of the original group of

pioneering webmasters who established the professional SEO field more than 20 years ago. Since then he has worked for major search engine technology providers, managed programming and marketing teams for Chicago Tribune, and consulted for numerous entities including Fortune 500 companies. Detlef has a strong understanding of Technical SEO and a passion for Web programming. As a noted technology moderator at our SMX conference series, Detlef will continue to promote SEO excellence combined with marketing-programmer features and webmaster tips.
http://feeds.searchengineland.com/~r/searchengineland/~3/mCS2LPDn45Q/evergreen-googlebot-chromium-rendering-engine-316652