Prioritizing Your Resources with link rel='preload'

By Jeff Posnick
Web DevRel @ Google

Have you ever wanted to let the browser know about an important font, script, or other resource that will be needed by the page, without delaying the page's `onload` event? `<link rel="preload">` gives web developers to power to do just that, using a familiar HTML element syntax with a few key attributes to determine the exact behavior. It's a draft standard that's shipping as part of the Chrome 50 release.

Resources loaded via `<link rel="preload">` are stored locally in the browser, and are effectively inert until they're referenced in the DOM, JavaScript, or CSS. For example, here's one potential use case in which a script file is preloaded, but not executed immediately, as it would have been if it were included via a `<script>` tag in the DOM.

```
<link rel="preload" href="used-later.js" as="script">
<!-- ...other HTML... -->
<script>
  // Later on, after some condition has been met, we run the preloaded
  // JavaScript by inserting a <script> tag into the DOM.
  var usedLaterScript = document.createElement('script');
  usedLaterScript.src = 'used-later.js';
  document.body.appendChild(usedLaterScript)
</script>
```

So what's happening here? The `href` attribute used in that example should be familiar to web developers, as it's the standard attribute used to specify the URL of any linked resource.

The as attribute is probably new to you, however, and it's used in the context of a `<link>` element to give the browser more context about the destination of preloading request being made. This additional information ensures that the browser will set appropriate request headers, request priority, as well as apply any relevant Content Security Policy directives that might be in place for the correct resource context.

Learn (a lot) more

Yoav Weiss wrote the definitive guide to using `<link rel="preload">`. If you're intrigued and want to start using it on your own pages, I'd recommend reading through his article to learn more about the benefits and creative use cases.

Goodbye `<link rel="subresource">`

`<link rel="preload">` supersedes `<link rel="subresource">`, which has significant bugs and drawbacks, and which was never implemented in browsers other than Chrome. As such, Chrome 50 removes support for `<link rel="subresource">`.