NOTE: this is a condensed (and public) version of an internal doc on the topic.

panicker@google.com
May 6, 2021

tl;dr

While usage of preload for critical resources can result in an improvement in a loading metric, it is a major footgun and comes at a significant cost to the developer. Preloading can devolve into re-creating and maintaining roughly the natural loading sequence of the browser, but entirely manually. This is further exacerbated by a severe issue in Chrome.

NOTE: preload is useful primitive for fetching script without executing, here we focus on critical resource loading with: link "rel=preload"

What are the (theoretical) scenarios for preload usage?

NOTE: critical resources refers to resources impacting any loading metric: FCP, LCP, FID, TTI.

1. late discovered critical resources due to blocking behind another network request
Some resources can only be discovered after fetching and parsing another resource. Eg.

- discover a font after fetching and parsing CSS file
- discover an image after fetching and executing some JS

2. late discovered critical resources due to main thread being busy, delaying HTML parsing
Eg.

- an early sync script delays discovery of critical CSS / JS
- an async 3P script executes early and delays discovery of 1P JS

3. the *external* nature of critical resources delays initial rendering
critical resources like CSS, fonts require fetching an external file, adding a round-trip. The sooner these can be discovered, the better.

So what's the problem?

Due to the pervasive and widespread nature of the above scenarios, the recommendation and usage devolves into: "use preload for critical resources". This easily does more harm than good.

Secondly, early network bandwidth and main thread -- are scarce resources, preloading a resource is typically at the cost of another resource. This means using preload requires developer expertise in deeply understanding the ordering of resources for optimal FCP, LCP, FID, TTI etc. It entails precise knowledge of which resources impact FCP, LCP, FID etc (often unreasonable) and maintaining order even as the application evolves.

    - which CSS is critical? which font files are blocking?

    - which is the LCP image
    - which other above-the-fold images are important for UX

Furthermore, in scenarios where a developer is most tempted to reach for preload, there is often a deeper issue to be addressed, for instance examining alternatives to render-blocking fonts, or parser-interrupting script instead of tacking on a preload.

Finally, this Chrome issue causes preload to jump the queue forcing tedious manual ordering (with preload) on every preceding resource. This manual curation is an additional burden for developers, entailing precise tracking of hero image, external CSS etc This is difficult in a codebase with many contributors, where the page structure may change or additional critical resources may be added over time.

When the curation is sub-optimal or out-of-date, there aren't easy ways to detect the ordering problems. Furthermore, this bug gets in the way of the most compelling usecase of preload for hidden, late discovered resources.

Preloading self-bundled JS is a footgun

Preload can easily do more harm than good here, particularly for server-rendered pages.

Preloading JS causes maximum bandwidth contention with all other higher priority resources like CSS, critical fonts, LCP image. This has the highest maintainability cost, as it also requires preloading and correctly ordering all prior resources (CSS, font, images etc) to manage the contention manually.

Again, this requires precise knowledge of which resources impact FCP, LCP and ordering them correctly, and maintaining the full order for the complete loading sequence. A case of re-creating the natural loading sequence of the browser, but entirely manually.

Preloading fonts is a footgun

Issue with preloading fonts:

- fonts are typically quite heavy due to permutations of variants, weights, glyphs etc and take up significant network bandwidth. This means they can easily block other resources that should be fetched first, such as CSS.. Sadly it is easy to trip over this problem. This is good write-up of the problem from Andy Davies.
- above issue is compounded due to variance in behavior across different browsers
- the need for preload depends to an extent on whether the font needs to be render-blocking plus whether it is late discovered (buried behind a CSS round trip). Without knowing the answers, it is unwise to simply default to adding a preload.
- NOTE: Render blocking fonts may be needed in cases where they are branding critical. However many fonts can be made non-render-blocking (with display value swap or optional)

Okay, so when *should* I preload?

NOTE: After the preload bug is fixed -- I will reassess if it can be promoted for “hidden” late-discovered resources -- currently that is both the most compelling usecase for preload, but also non-trivial and costly to developers, due to the bug.

Preload can be avoided in many cases, with alternative strategies such as inlining of critical CSS and inlining font-css. Preconnect can fill the gaps for some external resources.

Also blurry placeholder for LCP image reduces the need for preloading LCP image, by improving UX significantly (LCP metric still needs to catch up here unfortunately, as it currently doesn’t *intend* to reward the placeholder)

A good use-case for preload is external CSS, for optimizing FCP.

Preloading LCP image can make sense, but it gets into the “power user” territory, as one has to also preload and maintain critical resources for FCP such as CSS and fonts.

Preloading fonts is squarely *power user* territory, it can be useful in cases where:

- when fonts are late discovered after CSS is fetched and parsed, and CSS cannot be inlined
- when fonts are render-blocking, and especially if they are *also* late discovered

NOTE: all the caveats mentioned in the previous section still apply.