Clear-Site-Data integration with Prerender and Prefetch [Public]

We propose adding two new values to the Clear-Site-Data header to help developers target clearing the prerender and prefetch cache: “prefetchCache” and “prerenderCache”.

Github issue: Specify clear-site-data integration, and special keywords for prefetch and prerender · Issue #357 · WICG/nav-speculation

Background

The Clear-Site-Data header is a powerful tool for web developers to clear various types of data stored by a website, such as cookies and storage. Currently, the “cache” value of this header includes the functionality to evict prefetches and cancel prerenders. However, there is a need for more granular control to allow developers to specifically target these actions without affecting other cached data.

One specific issue brought up by Shopify related to prerendering is that when a user adds a product to the cart and then navigates to a prerendered page, the cart content or cart count displayed on the prerendered page does not reflect the recent addition to the cart. This discrepancy occurs because the prerendered page shows the state before the product was added to the cart. While there could be workarounds like using local storage flags and forcing page reloads, these are more complicated and would go against maintaining the purpose and efficiency of prerendering. Using Clear-Site-Data header value “prerenderCache” to specifically target clearing the prerender cache would help Shopify ensure pages are in sync.

Implementation

On top of the existing values to the Clear-Site-Data header, we will also include:

- “prefetchCache”: Used to evict prefetches that are scoped to referrer origin.
- “prerenderCache”: Used to cancel pretenders that are scoped to referrer origin.

These added values will not affect other caches, just their respective target.

Example code

Client Side:

```
addToCartButton.onclick = () => {
  fetch("/add-to-cart", { method: "POST", body: JSON.stringify(cartData),
headers: { "Content-Type": "application/json" } })
      .then(response => {
      if (response.ok) {
      console.log("Item added to cart successfully.");
      } else {
      console.error("Failed to add item to cart.");
      }
      });
};
```

Server Side:

```
const express = require('express');
const app = express();
app.use(express.json());

app.post('/add-to-cart', (req, res) => {
  // Logic to add item to cart
  const cartData = req.body;
  // Assuming addItemToCart is a function that handles adding the item to
the cart
  addItemToCart(cartData);

  // Send Clear-Site-Data header to clear both prefetch and prerender
caches
  res.set('Clear-Site-Data', '"prefetchCache", "prerenderCache"');
  res.status(200).send('Item added to cart and caches cleared.');
});

function addItemToCart(cartData) {
  // Implementation for adding item to cart
  console.log('Item added to cart:', cartData);
}
```

Clear-Site-Data response header implementation

	Prefetch cache cleared	Prerender cache cleared
Clear-Site-Data: “cache”	x	x
Clear-Site-Data: “prefetchCache”	x	
Clear-Site-Data: “prerenderCache”		x

Choice of origin scope

There have been discussions around how to scope different types of data in the context of the Clear-Site-Data header, with potential options including scoping by host, origin, site, domain, or other boundaries. In the case of prefetchCache and prerenderCache, we’ve decided to scope these to the origin.

This decision aligns with the approach taken for DOM-accessible storage (such as localStorage and IndexedDB) and execution contexts (like service workers and scripts), both of which are scoped by origin. Following this pattern maintains consistency across web platform features and simplifies mental models for developers and implementers.

Additionally, this origin-based scoping fits with the heuristic that these caches are tied to a specific origin’s trust boundary. For example, prefetching and prerendering are security-sensitive operations because they can involve executing or preparing resources ahead of user navigation, and it’s important that these operations respect origin isolation. Scoping by origin ensures a clear separation between data and behavior belonging to different origins, which helps avoid unintended data retention or cross-origin leakage.

Same-Origin and Cross-Origin Prefetch and Prerender Handling

When the server sends back Clear-Site-Data header with prefetchCache and/or prerenderCache value(s), prefetches and prerenders that are same-origin will be cleared, as they are scoped by referrer origin. Currently, prerendering is restricted to same-origin documents by default.

Since there is support for cross-origin prefetch and credentialed-prerender, prefetches and prerenders that are cross-origin but have the same referrer origin as the response's origin will also be cleared. For example, if Origin A sends Clear-Site-Data: prefetchCache, then we should clear all prefetches with referrer origin A, even if the prefetched page's origin is different. This also means that we do not always clear prefetches where the prefetched response's origin is A, unless the referrer origin is also A.

Which value(s) to pass in the header?

The current implementation is to decouple prefetchCache and prerenderCache values, meaning passing in only prefetchCache will only clear the prefetch cache, and passing in only prerenderCache will only clear the prerender cache. In most cases, clearing prefetch will likely mean that the prerenders are out of date and should be cleared. However, a prerendered page can be kept up to date (using something like BroadcastChannel) even if the resource from the prefetch cache is cleared. In those cases, it is useful to have these functionalities separate.

Tldr: If you are not using something like BroadcastChannel to keep prerendered pages up to date and you would like to clear both prerefetches and prerenders, passing both prefetchCache and prerenderCache to Clear-Site-Data is recommended (see example code above).

Otherwise, passing prefetchCache only will only clear prefetches and keep prerenders still available to activate.