CacheQueryOptions Arrive in Chrome 54

By Jeff Posnick
Web DevRel @ Google

If you use the Cache Storage API, either within a service worker or directly from web apps via `window.caches`, there's some good news: starting in Chrome 54, the full set of `CacheQueryOptions` is supported, making it easier to find the cached responses you're looking for.

What options are available?

The following options can be set in any call to `CacheStorage.match()` or `Cache.match()`. When not set, they all default to `false` (or `undefined` for `cacheName`), and you can use multiple options in a single call to `match()`.

`ignoreSearch`

This instructs the matching algorithm to ignore the search portion of a URL, also known as the URL query parameters. This can come in handy when you have a source URL that contains query parameters that are used for, for example, analytics tracking, but are not significant in terms of uniquely identifying a resource in the cache. For example, many folks have fallen prey to the following service worker "gotcha":

```
self.addEventListener('install', event => {
  event.waitUntil(
    caches.open('my-cache')
      .then(cache => cache.add('index.html'))
  );
});

self.addEventListener('fetch', event => {
  // Make sure this is a navigation request before responding.
  if (event.request.mode === 'navigation') {
    event.respondWith(
      caches.match(event.request) || fetch(event.request)
    );
  }
});
```

This sort of code works as expected when a user navigates directly to `index.html`, but what if your web app uses an analytics provider to keep track of inbound links, and the user navigates to `index.html?utm_source=some-referral`? By default, passing `index.html?utm_source=some-referral` to `caches.match()` won't return the entry for `index.html`. But if `ignoreSearch` is set to `true`, you can retrieve the cached response you'd expect regardless of what query parameters are set:

```
caches.match(event.request, {ignoreSearch: true})
```

cacheName

`cacheName` comes in handy when you have multiple caches and you want a response that's stored in one specific cache. Using it can make your queries more efficient (since the browser only has to check inside one cache, instead of all of them) and allows you to retrieve a specific response for a given URL when multiple caches might have that URL as a key. `cacheName` only has an effect when used with `CacheStorage.match()`, not `Cache.match()`, because `Cache.match()` already operates on a single, named cached.

```
// The following are functionally equivalent:
caches.open('my-cache')
  .then(cache => cache.match('index.html'));

// or...
caches.match('index.html', {cacheName: 'my-cache'});
```

ignoreMethod and ignoreVary

`ignoreMethod` and `ignoreVary` are a bit more niche than `ignoreSearch` and `cacheName`, but they serve specific purposes.

`ignoreMethod` allows you to pass in a `Request` object that has any `method` (`POST`, `PUT`, etc.) as the first parameter to `match()`. Normally, only `GET` or `HEAD` requests are allowed.

```
// In a more realistic scenario, postRequest might come from
// the request property of a FetchEvent.
const postRequest = new Request('index.html', {method: 'post'});

// This will never match anything.
caches.match(postRequest);
```

```
// This will match index.html in any cache.
caches.match(postRequest, {ignoreMethod: true});
```

If set to `true`, `ignoreVary` means that cache lookups will be done without regards to any `Vary` headers that are set in the cached responses. If you know that you are not dealing with cached responses that use the Vary header, then you don't have to worry about setting this option.

Browser support

`CacheQueryOptions` is only relevant in browsers that support the Cache Storage API. Besides Chrome and Chromium-based browsers, that's currently limited to Firefox, which already natively supports `CacheQueryOptions`.

Developers who want to use `CacheQueryOptions` in versions of Chrome prior to 54 can make use of a polyfill, courtesy of Arthur Stolyar.

Last updated July 2, 2018.