Updates to the Service Worker Cache API

By Jake Archibald

Human boy working on web standards at Google

I've been asked to write this post on a fairly minor update to the service worker cache API. I didn't think it warranted its own article, but after a long debate that eventually came down to a game of rock-paper-scissors, I lost, so here it is.

Are you ready to hear about the updates to Chrome's implementation of the service worker cache API?

I can't hear you! I said, are you ready to hear about the updates to Chrome's implementation of the service worker cache API?

(you may only read on if you've jumped onto your chair and shouted "YEAHHH!". Simultaneously pretending to swing a lasso is optional, but encouraged).

cache.addAll arrived in Chrome 46

Yes! That's right! Cache! Dot add all! Chrome 46!

Ok ok, sarcasm aside, this is actually a pretty big deal, as **`cache.addAll`** is the last remaining part of the cache "essentials" polyfill, meaning it's no longer needed.

Here's how **`cache.addAll`** works:

```
// when the browser sees this SW for the first time
self.addEventListener('install', function(event) {
  // wait until the following promise resolves
  event.waitUntil(
    // open/create a cache named 'mysite-static-v1'
    caches.open('mysite-static-v1').then(function(cache) {
      // fetch and add this stuff to it
      return cache.addAll([
        '/',
        '/css/styles.css',
        '/js/script.js',
        '/imgs/cat-falls-over.gif'
      ]);
```

```
    })
  );
});
```

addAll takes an array of urls & requests, fetches them, and adds them to the cache. This is transactional - if any of the fetching or writing fails, the whole operation fails, and the cache is returned to its previous state. This is particularly useful for install operations like above, where a single failure should be an overall failure.

Note: As of Chrome 50, the **cache.addAll()** (and **cache.add()**) behavior has changed slightly. Now, all of the responses that are added to the cache need to have a **2xx** (i.e. "OK") response code. Any responses with non-**2xx** response codes, including opaque responses from non-CORS requests, will cause the **cache.addAll()** to reject.

The example above is within a service worker, but the caches API is fully accessible from pages too.

Firefox already supports this API in their developer edition, so it'll land with the rest of their service worker implementation.

But wait, that's not all, there are more cache improvements in the pipeline…

cache.matchAll coming to Chrome 47

This allows you to get multiple matches:

```
caches.open('mysite-static-v1').then(function(cache) {
  return cache.matchAll('/');
}).then(function(responses) {
  // …
});
```

The above will get everything in **mysite-static-v1** that matches **/**. The cache lets you have multiple entries per URL if they're independently cacheable, eg if they have different **Vary** headers.

Firefox already supports this in their developer edition, so it'll land with the rest of their service worker implementation.

Cache query options coming to Chrome… soon

Here's a pretty standard fetch handler:

```
self.addEventListener('fetch', function(event) {
  event.respondWith(
    caches.match(event.request).then(function(response) {
      return response || fetch(event.request);
    })
  );
});
```

If we have `/` cached, and we get a request for `/`, it'll be served from the cache. However, if we get a request for `/?utm_source=blahblahwhatever` that *won't* come from the cache. You can work around this by ignoring the url search string while matching:

```
self.addEventListener('fetch', function(event) {
  event.respondWith(
    caches.match(event.request, {
      ignoreSearch: true
    }).then(function(response) {
      return response || fetch(event.request);
    })
  );
});
```

Now `/?utm_source=blahblahwhatever` will be matched to the entry for `/`! The full options are:

- **`ignoreSearch`** - ignore the search portion of the url in both the request argument and cached requests
- **`ignoreMethod`** - ignore the method of the request argument, so a POST request can match a GET entry in the cache
- **`ignoreVary`** - ignore the vary header in cached responses

Firefox already supports this in their… ok you know the drill by now. Go tell Ben Kelly how great he is for getting all of this into Firefox.

If you want to follow the Chrome implementation of the cache query options, check out crbug.com/426309.

See you next time for another exciting chapter of "what we implemented in the cache API"!

---

registered trademark of Oracle and/or its affiliates.

Last updated July 2, 2018.