The danger of opaque responsess

While working with Workbox I discovered a hidden danger of opaque responses. This post will address these shortcomings and provide an imperfect solution for how to do it.

Normally, opaque responses are not cacheable along with additional restrictions. See Jeff Posnick's Stack Overflow Answer for details but it boils down to the following items:

- Because opaque responses are meant to be a black-box you won't get meaningful information from most of the properties of the `Response` class, or call the various methods that make up the Body interface, like `json()` or `text()`
- Browsers pad the opaque resource size. In the case of Chrome the *minimum* size that any single cached opaque response contributes to the overall storage usage is approximately 7 megabytes
- The status property of an opaque response is always set to 0, regardless of success or failure
- The `add()/addAll()` methods of the Cache API will reject if any response is outside the 2XX range

Are opaque requests useful?

Yes, there are places where opaque requests can be used without problem. Based on MDN's Cross-origin network access article:

- `<script src="…"></script>`. Error details for syntax errors are only available for same-origin scripts
- `<link rel="stylesheet" href="…">` as long as it's served with the correct mime type
- Images displayed by `<img>`
- `<video>` and `<audio>`
- Plugins embedded with `<object>`, `<embed>`, and `<applet>`
- Fonts applied with `@font-face`. Some browsers allow cross-origin fonts, others require same-origin
- Anything embedded by `<frame>` and `<iframe>`

Caching opaque responses

The example below creates a new `Request` for an opaque response.

We then use the Fetch API to retrieve the requested object and we put it in the cache.

The idea is that by putting the resource in the cache we're opting in to accepting whatever the resource is, even an error.

Because of the padding we need to be extra careful when we decided how many items we want to cache before our origin runs out of quota

```
const request = new Request('https://third-party-no-cors.com/', {
    mode: 'no-cors'
  });
'no-cors'// Assume `cache` is an open instance of the Cache class.
fetch(request).then(response => cache.put(request, response));
```

Workbox.js provides additional functionality to make it easier to work with opaque responses by using plugins.

The handler below uses two plugins: `expiration` and `cacheableResponse`.

The expiration plugin (`workbox.expiration.Plugin()`) dictates how long Workbox will keep the resources in the cache.

The cacheableResponse plugin (`workbox.cacheableResponse.Plugin()`) changes the behavior of Workbox regarding opaque responses.

`statuses` indicates what HTTP status codes we want to accept for this handler. I've chosen to automatically accept 0 in addition to 200 as valid status codes for this request.

`purgeOnQuotaError: true` tells Workbox that it's ok to delete this cache when we hit the quota limit for this domain. We do this because we're accepting opaque responses and they are padded (at least 7 MB each in Chrome).

```
const extFontHandler = workbox.strategies.staleWhileRevalidate({
  cacheName: 'external-fonts',
  plugins: [
    new workbox.expiration.Plugin({
      maxAgeSeconds: 30 * 24 * 60 * 60,
      // maxEntries: 20,
    }),
    new workbox.cacheableResponse.Plugin({
      statuses: [0, 200],
      // Automatically cleanup if quota is exceeded.
      purgeOnQuotaError: true,
    }),
  ],
});
```

Each handler is associated with one or more routes. My project is working only with Adobe Fonts (Typekit) and Google Fonts. I split the domains into two, one for each provider I'm working with.

The first route uses a regular expression to match `use.typekit.net`.

The second one is more commplicated. The regular expression matches either `fonts.google.com` or `fonts.gstatic.com`. There may be other domains and that will mean that we need additional routes for those dommains.

```
// Third party fonts from typekit
workbox.routing.registerRoute(/https:\/\/use\.typekit\.net/, (a/https:\/\/
});

// Third party fonts from google fonts
workbox.routing.registerRoute(/https:\/\/fonts\.(googleapis|gstatic)\.com/
  return extFontHandler.handle(args);
});
/https:\/\/fonts\.(googleapis|gstatic)\.com/
```

We've accepted opaque responses and have configured the Service Worker to expire the requests after a given period of time and to delete the cache when we hit the quota limit.

So this provides an imperfect solution to caching opaque requests. I don’t know if this is better than hosting all resources locally or not. It is what I have now and it works as intended.