Offline-first, fast, with the sw-precache module

By Jeff Posnick
Web DevRel @ Google

You can't read about service workers without getting excited—they're the behind-the-scenes infrastructure that make it possible for web *pages* to act more like web *applications*. Upcoming web platform features like background sync and push notifications will rely on service workers, and following the release of Chrome 40, service worker-based caching is available to use today. If you've wanted to add service worker-powered offline support to your sites, but weren't sure how to get started, the **`sw-precache`** module is for you! **`sw-precache`** hooks into your existing **`node`**-based build process (e.g. **`Gulp`** or **`Grunt`**) and generates a list of versioned resources, along with the service worker code needed to precache them. Your site can start working offline and load faster even while online, by virtue of caching.

The service worker generated by **`sw-precache`** will cache and serve the resources that you configure as part of your build process. For smaller, mostly static sites, you can have it precache every image, HTML, JavaScript, and CSS file that makes up your site. Everything will both work offline, and load fast on subsequent visits without any extra effort. For sites with lots of dynamic content, or many large images that aren't always needed, precaching a "skeleton" subset of your site often makes the most sense. You can combine **`sw-precache`** with one of the service worker "recipes" or techniques outlined in the offline cookbook to provide a robust offline experience with sensible fallbacks—e.g. when a large, uncached image is requested offline, serve up a smaller, cached placeholder image instead.

Because **`sw-precache`** integrates into your site's build process, you can use wildcards to precache *all* of the resources that match a pattern—there's no list of files or URLs that you have to manually keep up to date. What's more, the module automatically versions all your cached resources based on a hash of each file's contents. When a change to any file is detected as part of your build process, the generated service worker knows to expire the old version and fetch the new version of the resource. All of the cache entries that remain the same are left untouched.

Here's a basic example of using **`sw-precache`** as part of a **`gulp`** build:

```
gulp.task('generate-service-worker', function(callback) {
  var path = require('path');
  var swPrecache = require('sw-precache');
  var rootDir = 'app';

  swPrecache.write(path.join(rootDir, 'service-worker.js'), {
    staticFileGlobs: [rootDir + '/**/*.{js,html,css,png,jpg,gif}'],
    stripPrefix: rootDir
  }, callback);
});
```

You'll see information about which resources will be precached, as well as the total precache size as part of the task output:

```
Starting 'generate-service-worker'...
Caching static resource 'app/css/main.css' (65 B)
Caching static resource 'app/images/one.png' (593 B)
Caching static resource 'app/images/two.png' (641 B)
Caching static resource 'app/index.html' (2.09 kB)
Caching static resource 'app/js/a.js' (7 B)
Caching static resource 'app/js/b.js' (7 B)
Caching static resource 'app/js/service-worker-registration.js' (3.37 kB)
Total precache size is about 6.77 kB for 7 resources.
Finished 'generate-service-worker' after 14 ms
```

There's a lot more information at the GitHub project page, including a demo project with **`gulpfile.js`** and **`Gruntfile.js`** samples, and a script you can use to register the generated service worker. If you'd like to see it in action, just check out the recently launched Google I/O 2015 web app—thanks (in part) to **`sw-precache`**, you can browse it at your leisure, online or off.

Last updated July 2, 2018.