Keeping things fresh with stale-while-revalidate

- What's it mean?
- Live Example
- What are the common use cases?
- How does it interact with service workers?
  - Use a service worker approach if...
  - Use a Cache-Control approach if...
- Learn more

`stale-while-revalidate` helps developers balance between immediacy—*loading cached content right away*—and freshness—*ensuring updates to the cached content are used in the future*. If you maintain a third-party web service or library that updates on a regular schedule, or your first-party assets tend to have short lifetimes, then `stale-while-revalidate` may be a useful addition to your existing caching policies.

Support for setting `stale-while-revalidate` alongside `max-age` in your `Cache-Control` response header is available in Chrome 75 and Firefox 68.

Browsers that don't support `stale-while-revalidate` will silently ignore that configuration value, and use `max-age`, as I'll explain shortly...

What's it mean?

Let's break down `stale-while-revalidate` into two parts: the idea that a cached response might be stale, and the process of revalidation.

First, how does the browser know whether a cached response is "stale"? A `Cache-Control` response header that contains `stale-while-revalidate` should also contain `max-age`, and the number of seconds specified via `max-age` is what determines staleness. Any cached response newer than `max-age` is considered fresh, and older cached responses are stale.

If the locally cached response is still fresh, then it can be used as-is to fulfill a browser's request. From the perspective of `stale-while-revalidate`, there's nothing to do in this scenario.

But if the cached response is stale, then another age-based check is performed: is the age of the cached response within the additional time window provided by the `stale-while-revalidate` setting?

If the age of a stale response falls into this window, then it will be used to fulfill the browser's request. At the same time, a "revalidation" request will be made against the network in a way that doesn't delay the use of the cached response. The returned response might contain the same information as the previously cached response, or it might be different. Either way, the network response is stored locally, replacing whatever was previously cache, and resetting the "freshness" timer used during any future `max-age` comparisons.

However, if the stale cached response is old enough that it falls outside the `stale-while-revalidate` window of time, then it will not fulfill the browser's request. The browser will instead retrieve a response from the network, and use that for both fulfilling the initial request and also populating the local cache with a fresh response.

Live Example

Below is a simple example of an HTTP API for returning the current time—more specifically, the current number of minutes past the hour.

https://glitch.com/edit/#!/s-w-r-demo

In this scenario, the web server uses this `Cache-Control` header in its HTTP response:

```
Cache-Control: max-age=1, stale-while-revalidate=59
```

This setting means that, if a request for the time is repeated within the next 1 second, the previously cached value will still be fresh, and used as-is, without any revalidation.

If a request is repeated between 1 and 60 seconds later, then the cached value will be stale, but will be used to fulfill the API request. At the same time, "in the background," a revalidation request will be made to populate the cache with a fresh value for future use.

If a request is repeated after more than 60 seconds, then the stale response isn't used at all, and both fulfilling the browser's request and the cache revalidation will depend on getting a response back from the network.

Here's a breakdown of those three distinct states, along with the window of time in which each of them apply for our example:

A diagram illustrating the information from the previous section.

What are the common use cases?

While the above example for a "minutes after the hour" API service is contrived, it illustrates the expected use case—services that provide information which needs to be refreshed, but where some degree of staleness is acceptable.

Less contrived examples might be an API for the current weather conditions, or the top news headlines that were written in the past hour.

Generally, any response that updates at a known interval, is likely to be requested multiple times, and is static within that interval is a good candidate for short-term caching via `max-age`. Using `stale-while-revalidate` in addition to `max-age` increases the likelihood that future requests can be fulfilled from the cache with fresher content, without blocking on a network response.

How does it interact with service workers?

If you've heard of `stale-while-revalidate` chances are that it was in the context of recipes used within a service worker.

Using stale-while-revalidate via a `Cache-Control` header shares some similarities with its use in a service worker, and many of the same considerations around freshness trade-offs and maximum lifetimes apply. However, there are a few considerations that you should take into account when deciding whether to implement a service worker-based approach, or just rely on the `Cache-Control` header configuration.

Use a service worker approach if...

- You're already using a service worker in your web app.
- You need fine-grained control over the contents of your caches, and want to implement something like a least-recently used expiration policy. Workbox's Cache Expiration module can help with this.
- You want to be notified when a stale response changes in the background during the revalidation step. Workbox's Broadcast Cache Update module can help with this.
- You need this `stale-while-revalidate` behavior in all modern browsers.

Use a Cache-Control approach if...

- You would rather not deal with the overhead of deploying and maintaining a service worker for your web app.
- You are fine with letting the browser's automatic cache management prevent your local caches from growing too large.
- You are fine with an approach that is not currently supported in all modern browsers (as of July 2019; support may grow in the future).

If you're using a service worker and also have `stale-while-revalidate` enabled for some responses via a `Cache-Control` header, then the service worker will, in general, have "first crack" at responding to a request. If the service worker decides not to respond, or if in the process of generating a response it makes a network request using `fetch()`, then the behavior configured via the `Cache-Control` header will end up going into effect.

Learn more

- `stale-while-revalidate` response in the Fetch API spec.
- RFC 5861, covering the initial `stale-while-revalidate` specification.
- The HTTP cache: your first line of defense, from the "Network reliability" guide on this site.

Hero image by Samuel Zeller.

□□□ #3
□ □□□ □□□ 2026-10-31 22:18:53 UTC
□ cocobook-sync □□□ 2026-11-04 17:38:07 UTC