Request: cache property

- Value
- Examples
- Specifications
- Browser compatibility
- See also

The **`cache`** read-only property of the Request interface contains the cache mode of the request. It controls how the request will interact with the browser's HTTP cache.

Value

A `RequestCache` value. The available values are:

- `default` — The browser looks for a matching request in its HTTP cache.
  - If there is a match and it is fresh, it will be returned from the cache.
  - If there is a match but it is stale, the browser will make a conditional request to the remote server. If the server indicates that the resource has not changed, it will be returned from the cache. Otherwise the resource will be downloaded from the server and the cache will be updated.
  - If there is no match, the browser will make a normal request, and will update the cache with the downloaded resource.
- `no-store` — The browser fetches the resource from the remote server without first looking in the cache, *and will not* update the cache with the downloaded resource.
- `reload` — The browser fetches the resource from the remote server without first looking in the cache, *but then will* update the cache with the downloaded resource.
- `no-cache` — The browser looks for a matching request in its HTTP cache.
  - If there is a match, *fresh or stale,* the browser will make a conditional request to the remote server. If the server indicates that the resource has not changed, it will be returned from the cache. Otherwise the resource will be downloaded from the server and the cache will be updated.

    - If there is no match, the browser will make a normal request, and will update the cache with the downloaded resource.
- `force-cache` — The browser looks for a matching request in its HTTP cache.
    - If there is a match, *fresh or stale*, it will be returned from the cache.
    - If there is no match, the browser will make a normal request, and will update the cache with the downloaded resource.
- `only-if-cached` — The browser looks for a matching request in its HTTP cache.
    - If there is a match, *fresh or stale*, it will be returned from the cache.
    - If there is no match, the browser will respond with a 504 Gateway timeout status.

  The `"only-if-cached"` mode can only be used if the request's `mode` is `"same-origin"`. Cached redirects will be followed if the request's `redirect` property is `"follow"` and the redirects do not violate the `"same-origin"` mode.

Examples

```
// Download a resource with cache busting, to bypass the cache
// completely.
fetch("some.json", { cache: "no-store" }).then((response) => {
  /* consume the response */
});

// Download a resource with cache busting, but update the HTTP
// cache with the downloaded resource.
fetch("some.json", { cache: "reload" }).then((response) => {
  /* consume the response */
});

// Download a resource with cache busting when dealing with a
// properly configured server that will send the correct ETag
// and Date headers and properly handle If-Modified-Since and
// If-None-Match request headers, therefore we can rely on the
// validation to guarantee a fresh response.
fetch("some.json", { cache: "no-cache" }).then((response) => {
  /* consume the response */
});

// Download a resource with economics in mind! Prefer a cached
// albeit stale response to conserve as much bandwidth as possible.
fetch("some.json", { cache: "force-cache" }).then((response) => {
```

```
  /* consume the response */
});

// Naive stale-while-revalidate client-level implementation.
// Prefer a cached albeit stale response; but update if it's too old.
// AbortController and signal to allow better memory cleaning.
// In reality; this would be a function that takes a path and a
// reference to the controller since it would need to change the value
let controller = new AbortController();
fetch("some.json", {
  cache: "only-if-cached",
  mode: "same-origin",
  signal: controller.signal,
})
  .catch((e) =>
    e instanceof TypeError && e.message === "Failed to fetch"
      ? { status: 504 } // Workaround for chrome; which fails with a TypeError
      : Promise.reject(e),
  )
  .then((res) => {
    if (res.status === 504) {
      controller.abort();
      controller = new AbortController();
      return fetch("some.json", {
        cache: "force-cache",
        mode: "same-origin",
        signal: controller.signal,
      });
    }
    const date = res.headers.get("date"),
      dt = date ? new Date(date).getTime() : 0;
    if (dt < Date.now() - 86_400_000) {
      // if older than 24 hours
      controller.abort();
      controller = new AbortController();
      return fetch("some.json", {
        cache: "reload",
        mode: "same-origin",
        signal: controller.signal,
```

```
      });
    }

    // Other possible conditions
    if (dt < Date.now() - 300_000)
      // If it's older than 5 minutes
      fetch("some.json", { cache: "no-cache", mode: "same-origin" }); // no cancellation or
return value.
    return res;
  })
  .then((response) => {
    /* consume the (possibly stale) response */
  })
  .catch((error) => {
    /* Can be an AbortError/DOMException or a TypeError */
  });
```

Specifications

Browser compatibility

See also

- ServiceWorker API
- HTTP access control (CORS)
- HTTP

□□□ #3
□ □□□ □□□ 2026-10-31 03:27:53 UTC
□ cocobook-sync □□□ 2026-11-04 17:34:52 UTC