Service worker communication

Importing `ServiceWorkerModule` into your `AppModule` doesn't just register the service worker, it also provides a few services you can use to interact with the service worker and control the caching of your application.

Prerequisites A basic understanding of the following: * Getting Started with Service Workers.

`SwUpdate` service

The `SwUpdate` service gives you access to events that indicate when the service worker discovers and installs an available update for your application.

The `SwUpdate` service supports three separate operations: * Getting notified when an updated version is *detected* on the server, *installed and ready* to be used locally or when an *installation fails.* * Asking the service worker to check the server for new updates. * Asking the service worker to activate the latest version of the application for the current tab.

Version updates

The `versionUpdates` is an `Observable` property of `SwUpdate` and emits four event types: * `VersionDetectedEvent` is emitted when the service worker has detected a new version of the app on the server and is about to start downloading it. * `NoNewVersionDetectedEvent` is emitted when the service worker has checked the version of the app on the server and did not find a new version. * `VersionReadyEvent` is emitted when a new version of the app is available to be activated by clients. It may be used to notify the user of an available update or prompt them to refresh the page. * `VersionInstallationFailedEvent` is emitted when the installation of a new version failed. It may be used for logging/monitoring purposes.

Checking for updates

It's possible to ask the service worker to check if any updates have been deployed to the server. The service worker checks for updates during initialization and on each navigation request—that is, when the user navigates from a different address to your application. However, you might choose to manually check for updates if you have a site that changes frequently or want updates to happen on a schedule.

Do this with the `checkForUpdate()` method:

This method returns a `Promise<boolean>` which indicates if an update is available for activation. The check might fail, which will cause a rejection of the `Promise`.

In order to avoid negatively affecting the initial rendering of the page, `ServiceWorkerModule` waits for up to 30 seconds by default for the application to stabilize, before registering the ServiceWorker script. Constantly polling for updates, for example, with setInterval() or RxJS' interval(), prevents the application from stabilizing and the ServiceWorker script is not registered with the browser until the 30 seconds upper limit is reached.

Note that this is true for any kind of polling done by your application. Check the {@link ApplicationRef#isStable isStable} documentation for more information.

Avoid that delay by waiting for the application to stabilize first, before starting to poll for updates, as shown in the preceding example. Alternatively, you might want to define a different {@link SwRegistrationOptions#registrationStrategy registration strategy} for the ServiceWorker.

Forcing update activation

If the current tab needs to be updated to the latest application version immediately, it can ask to do so with the `activateUpdate()` method:

Calling `activateUpdate()` without reloading the page could break lazy-loading in a currently running app, especially if the lazy-loaded chunks use filenames with hashes, which change every version. Therefore, it is recommended to reload the page once the promise returned by `activateUpdate()` is resolved.

Handling an unrecoverable state

In some cases, the version of the application used by the service worker to serve a client might be in a broken state that cannot be recovered from without a full page reload.

For example, imagine the following scenario: - A user opens the application for the first time and the service worker caches the latest version of the application. Assume the application's cached assets include `index.html`, `main.<main-hash-1>.js` and `lazy-chunk.<lazy-hash-1>.js`. - The user closes the application and does not open it for a while. - After some time, a new version of the application is deployed to the server. This newer version includes the files `index.html`, `main.<main-hash-2>.js` and `lazy-chunk.<lazy-hash-2>.js` (note that the hashes are different now, because the content of the files changed). The old version is no longer available on the server. - In the meantime, the user's browser decides to evict `lazy-chunk.<lazy-hash-1>.js` from its cache. Browsers might decide to evict specific (or all) resources from a cache in order to reclaim disk space. - The user opens the application again. The service worker serves the latest version known to it at this point, namely the old version (`index.html` and `main.<main-hash-1>.js`). - At some later point, the application requests the lazy bundle, `lazy-chunk.<lazy-hash-1>.js`. - The service worker is unable to find the asset in the cache (remember that the browser evicted it).

Nor is it able to retrieve it from the server (because the server now only has `lazy-chunk.<lazy-hash-2>.js` from the newer version).

In the preceding scenario, the service worker is not able to serve an asset that would normally be cached. That particular application version is broken and there is no way to fix the state of the client without reloading the page. In such cases, the service worker notifies the client by sending an `UnrecoverableStateEvent` event. Subscribe to `SwUpdate#unrecoverable` to be notified and handle these errors.

More on Angular service workers

You might also be interested in the following: * Service Worker Notifications.