This document contains background information about the browser application cache and how it interacts with Meteor. For details on the `appcache` package API see http://docs.meteor.com/#appcache

How the Browser Uses the App Cache

A key to understanding how the browser uses the application cache is this:

The browser always loads the app cache in the background.

Or, to put it another another way, the browser never waits for the app cache to be updated.

For example, consider what happens when a user navigates to the app's web page for the first time, when they don't have the application cached yet. The browser will load the app as if it were a standard online application not using an app cache; and then the browser will also populate the app cache in the background. The *second* time the user opens the web page the browser load the app from the cache. Why? Because the browser never waits for the app cache to loaded. The first time the user opens the page the cache hasn't been loaded yet, and so the browser loads the page incrementally from the server as it does for web pages that aren't cache enabled.

As another example, suppose the user has previously opened the web page and so has an old copy of the application in the app cache. The user now returns to the web page, and in the meantime a new version of the application has been published. What happens? Since the browser never waits for the app cache, it will at first display the old version of the web page. Then, as Meteor makes its livedata connection and sees that a code update is available, the page will reload with the new code.

This behavior may seem strange. Why not check first to see if new code is available and avoid potentially briefly displaying an old version of the app? But consider if the user is offline, or has a bad or intermittent Internet connection. We don't know *how long* it will take to discover that a new version of the app is available. It could be five seconds or a minute or an hour... depending on when the browser is able to connect. So rather than waiting, not knowing how long the wait may be, instead the browser enables using the application offline by loading the application from the cache, and then updates the cache when a new version is available and can be downloaded.

The App Cache and Meteor Code Reloads

The appcache package is designed to support Meteor's hot code reload feature. (If you see any problems with code reloads when using the app cache, please report it as a bug).

With the appcache package installed a code reload will follow these steps:

- Meteor's livedata stream connection notices that a code update is available and initiates a reload.
- The appcache package is notified that a code migration has started (as the appcache package plugs into Meteor's reload `onMigrate` hook). The appcache package calls `window.applicationCache.update()` to ask the browser to update the app cache. The appcache package then reports back to reload that it isn't ready for migration yet... until the browser reports that the app cache has finished updating. The reload is thus delayed until the new code is in the cache.
- Meteor's reload calls `window.location.reload()`, which reloads the app in the web page with the new code in the app cache.

Designed for Static Resources Only

When a browser displays an image in an application, we can describe that image as either being static (it stays the same for any particular version of the application) or dynamic (it can change as the application is being used).

For example, a "todo" app might display a green checkmark image for completed todo items. This would be a static image, because it is part of the application and changing the image would require a code push.

Conversely, we might imagine that the app could allow the user to upload images to add to a todo item's description. These images would be dynamic images, because new images can be added and images can be images as the application is running.

The appcache package is only designed to cache static resources. As an "application" cache, it caches the resources needed by the application, including the HTML, CSS, Javascript and files published in the public/ directory.