Caching Introduction

Simon Bragg
https://sibra.co.uk

Cambridge WordPress Meetup Jan 2023

Caching: General

A cache is a local, easily accessible, temporary store, that can be retrieved faster. Used for:

- HTML
- Compiled PHP
- MySQL queries
- Anything really

For example you might:

- Keep a few pens in a holder on your desk: the holder is your pen cache
- Keep your entire stock of pens in box in a cabinet.

Caching: Approach to serve a page

Server: No Caching

1. Request from visitor's browser for a page
2. Server finds PHP for page
3. Runs PHP + SQL calls to get data from memory
4. Creates html for page
5. Sends html to browser.
6. Browser processes html
7. Browser requests CSS, JS, & image files
8. Browser renders page.

Server: With Caching

1. Request from visitor's browser for a page
2. Redirects request to a stored html or compiled PHP file.
3. Sends html file to browser
4. Browser processes html
5. Browser requests various CSS, JS, images
6. Browser renders page

Caching: Redirect

Server needs to redirect browser request to the cached file

Redirect options:

1. “Enhanced”
   Redirect set up in wp-config which is faster.

2. “Basic”
   If can’t write to the wp-config file, then caching plugin runs redirect, which is slower.

Caching: HTML

Plugin caches the resulting HTML output

Most caching plugins do this, as saves time for future requests.
Sometimes called page caching. Usually plugins offer:

- **Cache invalidation mechanism**: so you can serve uncached content when wanted.

- **Minify HTML**: compresses HTML, e.g. could remove comments. Saves a few kBytes/page.

- **Cache Preload:** If clear/empty the cache, then first visitor to page waits for plugin to create the cache for that page. Cache Preload means build the page cache in advance.
- Can clear & rebuild the page cache at set intervals, but avoid running on the hour.

Add images to CSS or HTML

Save browsers requesting & waiting for small images by:

- base64 encode images into the HTML or CSS.
- Useful for e.g. small png used as a bullet point.

Sometimes including small JS scripts inline in the html <head> works.

Caching: PHP

For dynamic sites, where HTML user dependent.

Need to execute PHP. Two types:

PHP Opcode Cache

- PHP compiled into bytecode, executed in a PHP "virtual machine".
- Resides in memory
- Need to access PHP configuration file to enable PHP OpCache.

PHP Object Cache

PHP is object orientated. Objects constantly created & destroyed.

- Object caching enabled my Memcached Object Cache plugin, + various Redis Cache plugins.
- Enable PHP object caching

Caching:MySQL Queries

Possible to cache MySQL queries:

- Only needed, if you're using PHP caching.
  Presumably not needed for HTML caching.

- Apparently difficult to get right.

Browser Caching 1

Asks the user's browser to store the downloaded file.

How browser decides if its local copy is OK:

Settings where browser asks servers' permission to use its copy:

- Last modified header: date file last modified

- Entity tag: Unique version number.
  - Can track unique users
  - Takes precedence over last modified header
  - Doesn't work well on load balances servers across multiple machines.

Browser Caching 2

Settings where browser doesn't request permission, so faster:

- Expires header: HTTP date, GMT time
  - Good for static images

- Cache Control header settings:
  - Has priority over expires headers
  - **Max-age=** Number seconds file remains fresh
  - **Public**: authenticated responses are cacheable. If HTTP authentication required, responses are private.
  - **private**: user stores response & share caches, but not a proxy.
  - **no-cache**: forces request to original server for validation
  - **no-store**: cache cannot keep a copy.
  - **must-revalidate**: obey the freshness commands.
  - **proxy-revalidate**: Like "must-revalidate", but for proxy caches.

Browser Caching 3

Google says: Specify either

- Expires or Cache Control – max age,
  plus
- Last Modified or ETag.

- Need both for 100% on Pingdom. Browser has to ask permission.

- **Cache control** max age depends on how often you change the content, styling and images.
- **Expires** period ranges from a few hours to 1 month.

Combine & Minify CSS

Reduce the number of files sent to browser, i.e. send

- **1 big, compressed file**
  - instead of several small files.

Typical Options:

- Either combine or minify or do both.
- Remove comments
- Line break removal
- Choices for @import processing to create a single file
  - Bubble – e.g. for a patch, forces it to top of CSS file. Bit unreliable.
  - Process: adds additional CSS files in order.

Minifying can make CSS in browser harder to understand.

Critical Path CSS

Send just the CSS to render the "above the fold" page first, then send all the other CSS files.

- Improves First Contentful Paint
- Google stops complaining about "remove unused CSS", as wants priority for "useful" CSS.

- Focus on Home Page/Landing pages,
  - Browser caching means pages should then have CSS & JS

- Could inline the critical path CSS in the <head> and
  - Load the original CSS files asynchronously or in the footer.

- Expensive IMHO.

Combine & Minify JavaScript etc

Never works for me

Options

- Download JS in the <head> or the footer.
- Blocking vs Non-blocking:
  - Blocking: JS downloaded, parsed & executed before page renders the HMTL. Slows first & largest contentful paint times
  - Non-Blocking: Page renders whilst the JavaScript downloads.

Problem:

- If page loads and JS fires “on-load”, then JS doesn’t run.
- Ok if JS runs on events e.g. mouse-over then ok.

Non Blocking JS options

Options include

- Non-blocking using JS: creates a dynamic script node. Like previous example.
- **Async**: script executes when downloaded. Order of execution not guaranteed. But scripts execute before load event.
- **Defer**: Waits for HTML to load before loading Javascript. JS should execute before DOMContentLoaded event, and execute in the defined order.
- **Extsrc**: refers to https://code.google.com/p/extsrcjs/ ?? Works like Defer, is cross-platform, and good if JavaScript uses document.write.
- **Asyncsrc**: is like extsrc, but only works if the JavaScript doesn't use document.write.

Minify & gzip?

They are different compression methods

- **Minify**: Lossy compression, as e.g. shortens class names
- **Gzip**: Compression algorithm, and
  - Requires browser to un-gzip the file.

Answer: in general yes, there is a benefit. It depends on:

- The effort required of the browser to unzip the file,
  versus
- The time saved sending the smaller file to the browser.

Prefetch

Get the browser to pull files in advance of needing them.

- <link rel="**prefetch**" href="/style.css" as="style" /> when you need a resource for the next page
- <link rel="**preload**" href="/style.css" as="style" /> : when you're going to need a resource in a few seconds
- <link rel="**preconnect**" href="https://example.com" /> when you'll need a resource soon, but you don't know its full
- url <link rel="**dns-prefetch**" href="https://example.com" />when you'll need a resource soon, but don't yet know its full url (for older browsers)
- <link rel="**prerender**" href="https://example.com/about.html" /> when most users navigate to a specific page, so speed it up
- <link rel="**modulepreload**" href="/script.js" /> Need an ES module soon.

Beware

Hosts tend to have their own caching, which:

- **Creates difficulties debugging your caching**
- **Some hosts (SiteGround) limit your choice of Caching plugins**

Discussion