CRITICAL RENDERING PATH

CRITICAL RENDERING PATH

The Critical Rendering Path is the sequence of steps the browser goes through to convert the HTML, CSS, and JavaScript into pixels on the screen.

CRITICAL RENDERING PATH

Critical Rendering Path

---

1. DOM

The HTML response is parsed from bytes to characters then into tokens which create nodes that become the Document Object Model Tree.

CSS is turned into a tree of objects called the CSS Object Model, which maps css rules to the styles for all elements. This process is very fast.

The DOM and CSSOM are combined into the render tree. Each DOM node is mapped to a matching CSSOM rule. Only visible nodes are included.

2. CSSOM

3. Render Tree

4. Layout

Compute the geometry of each node against the size of the viewport. Size and position on the screen are determined.

In the painting or rasterization phase, the browser converts each box calculated in the layout phase to actual pixels on the screen.

Moves elements on to a separate layer. This improves paint and repaint performance but costs more memory. Compositing may occur.

5. Paint

6. Compositing

Well Formed HTML

HTML syntax errors do not throw errors, the browser will fix the syntax and move on.

The object presentation of the HTML document and the interface of HTML elements to the outside world by JavaScript.

Think of it as an API and not just the elements on the page.

As the HTML is being parsed, the Preload Scanner will look for script, link and image tags to download to optimize speed and performance.

Document Object Model

Preload Scanning

Network Connections

When requesting resources, the browser is configured to allow a max number of simultaneous connections that lives on a separate thread for increased performance.

Most browsers allow for 6 concurrent conecctions.

Script tags will pause parsing, waiting for the script to be executed or if external, fetched and executed. Parsing will then continue.

Use async or defer on the script tag to prevent this.

CSS Render Blocking

Javascript Render Blocking

Script tags will pause parsing, waiting for the script to be executed or if external, fetched and executed. Parsing will then continue.

Use async or defer on the script tag to prevent this.

CSS is render blocking. The browser blocks page rendering until it receives and processes all of the CSS.

CSS is render blocking because rules can be overwritten, so the content can't be rendered until the CSSOM is complete.

Javascript Compilation

Building the Accessibility Tree

Reflow

While the CSS is being parsed and the CSSOM created, other assets, including JavaScript files, are downloaded (thanks to the preload scanner).

JavaScript is interpreted, compiled, parsed and executed.

The browser also builds an accessibility tree that assistive devices use to parse and interpret content.

The accessibility object model (AOM) is like a semantic version of the DOM. Until the AOM is built, the content is not accessible to screen readers.

Reflow is any subsequent size and location recalculations of any part of the page or the entire document. This will require re-layout and repaint.

A reflow will cause a reflow of the parent and children.

Device Refresh Rate

Frame Performance

Request Animation Frame

Most devices today refresh their screens 60 times a second. The browser needs to match the device's refresh rate and put up 1 new picture, or frame, for each of those screen refreshes.

Each of those frames has a budget of just over 16ms (1 second / 60 = 16.66ms). In reality, however, the browser has housekeeping work to do, so all of your work needs to be completed inside 10ms.

When visual changes are happening on screen you want to do your work at the right time for the browser, which is right at the start of the frame. Don't use setTimeout or setInterval, instead use requestAnimationFrame.

Reduce CSS Complexity

Avoid Layout Thrashing

Animations and Painting

Reduce complexity by using less selectors for a given styling rule.

Use BEM for a class-centric approach.

Reading and writing in close succession to the DOM will cause reflows before the next frame, causing “forced synchronous layout” and decreasing performance.

Group DOM read and writes together to avoid forced reflows. Checkout FastDOM.

Changing any property apart from transforms or opacity always triggers paint.

Use transforms and opacity for your animations.

Use Compositor to Avoid Painting

USER-CENTRIC
PERFORMANCE
METRICS

User Interactivity Perception Times

Changing any property apart from transforms or opacity always triggers paint.

Use transforms and opacity for your animations.

USER-CENTRIC PERFORMANCE METRICS

0–100 ms — Instant feel, constant flow
100–300 ms —Slight percetible delay
300–1000 ms — Loss of task focus, perceptible delay
1 s+ — Mental context switch;
10s+ — User leaves

First Paint

First Contentful Paint

First Meaningful Paint

The point when the browser renders anything that is visually different from what was on the screen prior to navigation.

The metric that answers the question: “is it happening?”

The point when the browser renders the first bit of content from the DOM.

The metric that answers the question: “is it happening?”

While subjective, this usually means when a major block of content is visible on the screen.

The metric that answers the question: “is it useful?”

Time To Interactivity

First Input Delay

RAIL Performance Model

The point at which your application is both visually rendered and capable of reliably responding to user input.

The metric that answers the question: “is it usable?”

Measures the time from when a user first interacts with your site (i.e. when they click, tap, scroll) to the time when the browser is actually able to respond to that interaction.

Represents user pain attempting to intereact with site.

Response
Animation
Idle
Load

Response

Animation

Idle

Response time reflects how rapidly your website reacts to inputs.

A response time under 100ms feels immediate; anything slower is noticeable to the user.

Animation speed applies to any visual animations you have on the page as well as user scrolling and dragging.

The ideal goal is 60fps, or one frame every 16ms.

Idle work refers to what is happening in the background of your website after it initially loads.

Idle work should be divided into 50ms blocks so that it doesn’t drag down response time.

Load

PERFORMANCE CHECKLIST

Performance Tools

Load, in the context of the RAIL model, means the first meaningful paint, which should appear less than one second after the user requests your page.

PERFORMANCE CHECKLIST

Google Lighthouse in Chrome
WebPageTest.org

Minify Assets

Concatentation

Web Font Strategy

Removes whitespace, comments and reduces file length.

Minify HTML, CSS and Javascript.

Combine files into one for CSS and Javascript.

Use compression (WOFF2 format).
Subset fonts to reduce file size.
Preload for faster download.
Use font-display: swap to prevent flashing issues.

Images

Javascript Loading

Javascript Profiling

Automate image compression.
Serve the right format.
Use a Lazy Loading strategy.
Images are responsive.
Image Sprites reduce requests.

Adding async or defer are highly recommended if your scripts are placed in the top of your page but less valuable if just before your </body> tag. Use defer most of the time to prevent html parser blocking.

Use browser developer tools to measure and view performance metrics and see bottlenecks.

Page Weight

Page Load Time

Minimize HTTP Requests

Total page weight should ideally be 500kb. Median is typically 1500kb.

Page should load in three seconds or less. The faster your site/app the lower the bounce rate.

Combine CSS, JS to reduce requests. Remove un-essential files.

Use Compression

Preresolve & Preload Resources

Javascript Strategies

Use Brotli or Gzip to compress static assets like HTML,CSS and Javascript. Integrate into automation workflow.

Use resource hints for the browser to load critical resources with: dns-prefetch, preconnect, prefetch and preload.

Implement tree-shaking, scope hoisting and code splitting. Read up on current methods to reduce size for your current tech stack.

Content Delivery Network

Service Worker

Use a CDN to delivery your assets. Popular JS and CSS libraries will have offer this option.

Cache assets with a service worker. Many other optimizations and performance gains can be be implemented with a service worker.