Critical Rendering Path

PageSpeed Rules and Recommendations

PageSpeed Insights rules in context: what to pay attention to when optimizing the Critical Rendering Path and why.

Eliminate render-blocking JavaScript and CSS

To deliver the fastest time to first render, you want to minimize and (where possible) eliminate the number of critical resources on the page, minimize the number of downloaded critical bytes, and optimize the critical path length.

Optimize JavaScript Use

JavaScript resources are parser blocking by default unless marked as *async* or added via a special JavaScript snippet. Parser blocking JavaScript forces the browser to wait for the CSSOM and pauses construction of the DOM, which in turn can significantly delay the time to first render.

Prefer async JavaScript resources

Async resources unblock the document parser and allow the browser to avoid blocking on CSSOM prior to executing the script. Often, if the script can be made async, it also means it is not essential for the first render - consider loading async scripts after the initial render.

Avoid synchronous server calls

Use the `navigator.sendBeacon()` method to limit data sent by XMLHttpRequests in `unload` handlers. Because many browsers require such requests to be synchronous, they can slow page transitions, sometimes noticeably. The following code shows how to use `navigator.sendBeacon()` to send data to the server in the `pagehide` handler instead of in the `unload` handler.

```
<script>
  function() {
    window.addEventListener('pagehide', logData, false);
    function logData() {
      navigator.sendBeacon(
        'https://putsreq.herokuapp.com/Dt7t2QzUkG18aDTMMcop',
        'Sent by a beacon!');
    }
  }();
</script>
```

Defer parsing JavaScript

Any non-essential scripts that are not critical to constructing the visible content for the initial render should be deferred to minimize the amount of work the browser has to perform to render the page.

Avoid long running JavaScript

Long running JavaScript blocks the browser from constructing the DOM, CSSOM, and rendering the page. As a result, any initialization logic and functionality that is non-essential for the first render should be deferred until later. If a long initialization sequence needs to be run, consider splitting it into several stages to allow the browser to process other events in between.

Optimize CSS Use

CSS is required to construct the render tree and JavaScript will often block on CSS during initial construction of the page. You should ensure that any non-essential CSS is marked as non-critical (e.g. print and other media queries), and that the amount of critical CSS and the time to deliver it is as small as possible.

Put CSS in the document head

All CSS resources should be specified as early as possible within the HTML document such that the browser can discover the `<link>` tags and dispatch the request for the CSS as soon as possible.

Avoid CSS imports

CSS import (@import) directive enables one stylesheet to import rules from another stylesheet file. However, these directives should be avoided because they introduce additional roundtrips into the critical path: the imported CSS resources are discovered only after the CSS stylesheet with the @import rule itself has been received and parsed.

Inline render-blocking CSS

For best performance, you may want to consider inlining the critical CSS directly into the HTML document. This eliminates additional roundtrips in the critical path and if done correctly can be used to deliver a “one roundtrip” critical path length where only the HTML is a blocking resource.

Next Sections

Updated on 2014-10-16

Authors

Ilya Grigorik