**How to reference your CSS files?**

In the previous lecture we spoke about media queries.

Specifically, if you reference an external file with a media query within the <link> tag, that CSS file will no longer be render blocking (provided of course the media query condition is not met), thus improving the Critical Rendering Path (CRP).

Although this is true, there are always other things to consider.

Let me explain why ...

Recap

We know that whenever we request a page from a server, there are 2 types of resources: render blocking and non-render blocking resources.

Render blocking resources are those assets that are essential for the rendering of a web page. This means the browser stops processing other resources while a render blocking resource is being parsed. This = bad.

By default, HTML and CSS are render blocking resources. HTML is essential for the DOM (Document Object Model) being constructed while CSS is essential for the CSSOM (CSS Object Model) being constructed.

However, not all CSS is essential for rendering the page. What users want immediately is the above-the-fold portion of the page, that is, the part of the page visible without scrolling.

What is Conditional CSS?

Conditional CSS is only used under certain conditions. For example, CSS for different screen sizes (like mobile) or print layouts or projecting onto a large monitor are typical examples of conditional CSS. These stylesheets should only be essential if the user loads the page under the given condition - for instance, on a mobile device or print page or large monitor.

I've got good news and bad news.

Lets start off with the bad news. The browser treats conditional CSS files as render blocking resources by default.

Now for the good news … there’s an easy way to make them non-render blocking using media queries.

For those of you who want some history, the @media attribute was introduced in CSS2, and it made it possible to define different style rules for different media *types*. Media queries in CSS3 extended the CSS2 media types idea. Now, instead of looking for a type of device, the media query looks at the *capability*of the device. To cut a long story short, media queries are now widely in use and supported.

How to use the media queries to prevent CSS render blocking

There are 3 ways to include media queries.

**#1, INSIDE HTML**

As we saw in the previous lecture, there is the <link> element that goes right in the document <head>. In this example. we’re telling the browser that we want to use different stylesheets at different viewport sizes:

1. <html>
2. <head>
3. <!-- Served to all users. media="all" is the default. This means that it applies in all cases, which means it is always render blocking. -->
4. <link rel="stylesheet" href="all.css" media="all" />
6. <!-- now lets define some media rules below. The browser will apply these stylesheets only if the page meets the condition and hence these stylesheet declarations don't need to block the rendering of the page when it is first loaded -->
8. <!-- Served to screens that are at least 18em wide -->
9. <link rel="stylesheet" href="small.css" media="(min-width: 18em)" />
10. <!-- Served to screens that are at least 50em wide -->
11. <link rel="stylesheet" href="medium.css" media="(min-width: 50em)" />
12. <!-- Served to screens that are at least 100em wide -->
13. <link rel="stylesheet" href="large.css" media="(min-width: 100em)" />
14. <!-- Served to screens that are at least 120em wide -->
15. <link rel="stylesheet" href="extra-large.css" media="(min-width: 120em)" />
16. <!-- Served to print media, like printers. -->
17. <link rel="stylesheet" href="print.css" media="print" />
18. </head>
19. <!-- rest of your page HTML and JS here -->
20. </html>

This can be a nice way to organize your code during development, and fine-tune the performance of your site by splitting styles up in a way that they’re not all downloaded at the same time. **In other words - this reduces the number of render blocking files.**

**Important:**using the media attribute doesn't stop conditional CSS from still being downloaded. The browser will still download these files, but they won’t be in the critical rendering path. As a result, the page load won't be blocked whilst they are downloading. So, for example, if a small screen device like a phone visits the site, it will only download the stylesheets in the media queries that match its viewport size. But if a larger desktop screen comes along, it will download the entire bunch because it matches all of those queries (excluding the print media query of course) (I could have structured the above media queries differently so that a large screen would not have to download all files, but I wanted to show you the above to make a point).

And finally, another place to include media rules inside HTML is directly inline. In other words, you could also include the media attribute to the <style> element used for inline CSS:

1. <style media="screen and (max-width: 600px)">
2. .desktop-banner {
3. display: none;
4. }
5. </style>

**#2, INSIDE CSS**

CSS is the most common place to place a media query. They go right in the stylesheet in an @media rule that wraps elements with conditions for when and where to apply a set of styles when a browser matches those conditions.

1. /\* On screens that are 600px or less, set the background color to olive \*/
2. @media screen and (max-width: 600px) {
3. body {
4. background-color: red;
5. }
6. }

**#3. INSIDE JAVASCRIPT**

We can use media queries in JavaScript, too! They work a lot like they do in CSS. But this is beyond the scope of our discussion.

So, should you have media queries within <link> tags, or include all of your media rules in one large CSS file?

Drumroll ......... it depends (don’t you just hate that!).

Let me explain.

Firstly, let me tackle including media rules within the <head> directly inline. I have heard developers swear by including CSS media queries within the <head> section because they argue it will cause your page to load much faster than linking externally to them. On the other side of the equation, I’ve heard a (greater) number of developers say that this will just cause way too much admin in the long run, because you’ll have to change the CSS on multiple pages every time you want to make a change.

**My personal view: I am not a fan of including CSS in the <head> section, given maintainability. I prefer to have all of my CSS in external files, and I think it's good practice, even if it comes at a small expense of having an extra HTTP request.**

Now that we've dealt with inline, we can cut to the chase: should you place all of your media queries in one CSS file, or place media queries within <link> elements?

Let me start off by saying that having different CSS files for different media queries (i.e. including media queries within the <link> tag) will reduce the number of render blocking resources. So if this is your top and only priority, then you should always do this.

Simple right?

Not so fast captain.

Having all of your media queries in different files may help with organizing your code during development, and reduce render blocking resources, but there's a flip side.

On the flip side, having multiple files also means that there are multiple HTTP requests to make. And HTTP requests are more expensive in terms of loading time as it has to contact the server and fetch the file. Another thing to bear in mind is that once a CSS file is loaded, it is cached by the browser. This means that even-though it might be initially slower to load the massive.css file, it doesn't need to be loaded again when you navigate around the site.

**In my opinion, which approach you use (using media queries in the HTML <link> element, or using @media rules in CSS) largely depends on how large your site is.**

For smaller websites where visitors are most likely to never navigate around the site other than a few pages, or if your site only has a few pages, then I would argue it's better to take full advantage of media queries within the <link> element, or dare I say it, even include inline CSS and media rules into the <head> section of the page. Adding specific code for devices inline can (sometimes) be a good way to use media queries if you only need to make a few changes. However, I tend to always prefer having multiple CSS stylesheets targeting different media rules. This way I can completely separate the styles shown to desktop browsers and those used for small screen devices for example, take advantage of improving the CRP, whilst keeping my CSS clean and separate. Maintainability will not be much of an issue for me in this case, because the site is small.

However, for larger sites things may differ and it may not be as simple as having different CSS files for each media rule. It could result in a maintenance nightmare! For example, what if you want to design a banner. Sometimes it is more convenient to have a banner.css file, and include all @media queries inside that file for easy reference. In my experience, for large sites, there is normally a combination of different methods used. If you look at a site like Amazon, for example, having one massive CSS file will not be practical either. That's why if you inspect their page you'll notice several CSS file links, as well as some inline CSS too. So what I'm trying to say is that for very large sites, you may want to divide your CSS files into function - one global one, one for the page layout, one for the menus, one for the tab controls, one for banners etc (the splitting out of CSS files is therefore not done on a media rule basis, but more on a modular component basis).

Another thing to mention is that most developers I know use css pre-processers (like Grunt, or Gulp) so you can build your CSS in modules, and let the pre-processor compile them into one large minified file at the end. Another option is to take your existing CSS file with one or more media queries, and extract all @media rules and save them as separate files using a plugin (like PostCSS for example). But anyways, I digress ...

Can you begin to see why I said that it ALL DEPENDS!

Conclusion

* If you add the media attribute to the <link> tag that calls a CSS file, you can turn it from a “render blocking” resource into a “non-render blocking” one, which will shorten the critical rendering path. **If this is your top priority (i.e. you don't care too much about multiple HTTP requests and/or maintainability) then this is arguably the best approach.**
* The downside is that every time you want to change a global CSS rule, you have to make the change on multiple files. However, you can still make a template for each section of your site, and update the code to multiple pages and files in minutes.
* You should always still minify your CSS files. Minification is the process of removing unnecessary or redundant data from CSS files such as whitespaces, comments, and characters. There is no need to do it by hand, of course, nowadays there are multiple compressor tools available on the internet for file minification.
* Avoid using @import. The @import function allows us to include external CSS files in HTML documents or inside other CSS files. This affects your web page speed as it loads every imported file separately instead of paralleled and also create unnecessary HTTP request to the server.
* Finally, I suggest you spend more time improving efficiency of your server side code or database, than worrying too much about having one vs multiple CSS files. It will be more productive as most of the inefficiency lies in server side code.

Whew. I know. This was quite a mouth full and a lot to take in.

Hope it makes sense.

See you in the next lecture

(p.s. if you would like to comment on what you do and give your opinion, please get involved in Q&A. I'm always interested in hearing what other developers like doing)