**Avoid an excessive DOM size**

May 2, 2019 • Updated Oct 4, 2019

Appears in: [Performance audits](https://web.dev/lighthouse-performance)

A large DOM tree can slow down your page performance in multiple ways:

* **Network efficiency and load performance**

A large DOM tree often includes many nodes that aren't visible when the user first loads the page, which unnecessarily increases data costs for your users and slows down load time.

* **Runtime performance**

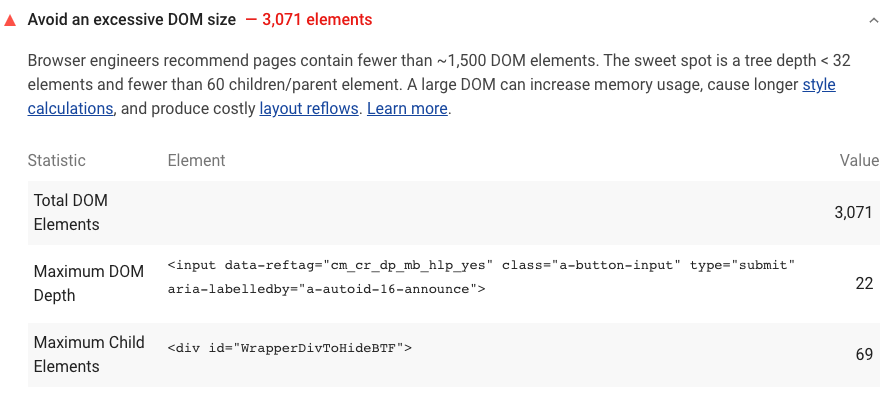
As users and scripts interact with your page, the browser must constantly [recompute the position and styling of nodes](https://developers.google.com/web/fundamentals/performance/rendering/reduce-the-scope-and-complexity-of-style-calculations?utm_source=lighthouse&utm_medium=cli). A large DOM tree in combination with complicated style rules can severely slow down rendering.

* **Memory performance**

If your JavaScript uses general query selectors such as document.querySelectorAll('li'), you may be unknowingly storing references to a very large number of nodes, which can overwhelm the memory capabilities of your users' devices.

**How the Lighthouse DOM size audit fails** [**#**](https://web.dev/dom-size/?utm_source=lighthouse&utm_medium=devtools#how-the-lighthouse-dom-size-audit-fails)

[Lighthouse](https://developers.google.com/web/tools/lighthouse/) reports the total DOM elements for a page, the page's maximum DOM depth, and its maximum child elements:



Lighthouse flags pages with DOM trees that:

* Warns when the body element has more than ~800 nodes.
* Errors when the body element has more than ~1,400 nodes.

See the [Lighthouse performance scoring](https://web.dev/performance-scoring) post to learn how your page's overall performance score is calculated.

**How to optimize the DOM size** [**#**](https://web.dev/dom-size/?utm_source=lighthouse&utm_medium=devtools#how-to-optimize-the-dom-size)

In general, look for ways to create DOM nodes only when needed, and destroy nodes when they're no longer needed.

If you're currently shipping a large DOM tree, try loading your page and manually noting which nodes are displayed. Perhaps you can remove the undisplayed nodes from the initially loaded document and only create them after a relevant user interaction, such as a scroll or a button click.

If you create DOM nodes at runtime, [Subtree Modification DOM Change Breakpoints](https://developers.google.com/web/tools/chrome-devtools/javascript/breakpoints#dom) can help you pinpoint when nodes get created.

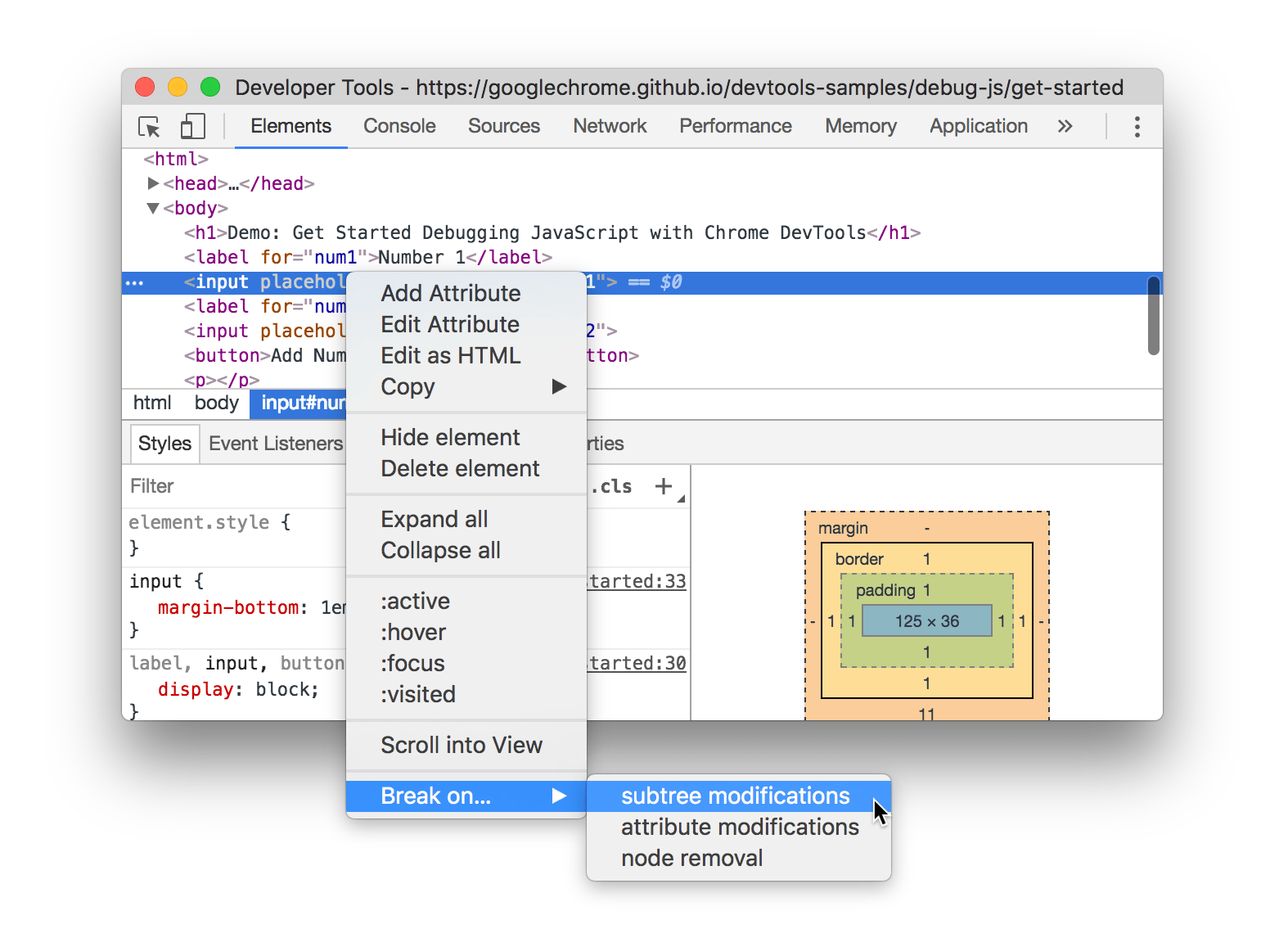
If you can't avoid a large DOM tree, another approach for improving rendering performance is simplifying your CSS selectors. See Google's [Reduce the Scope and Complexity of Style Calculations](https://developers.google.com/web/fundamentals/performance/rendering/reduce-the-scope-and-complexity-of-style-calculations) for more information

**DOM change breakpoints**

Use a DOM change breakpoint when you want to pause on the code that changes a DOM node or its children.

To set a DOM change breakpoint:

1. Click the **Elements** tab.
2. Go the element that you want to set the breakpoint on.
3. Right-click the element.
4. Hover over **Break on** then select **Subtree modifications**, **Attribute modifications**, or **Node removal**.



**Figure 5**: The context menu for creating a DOM change breakpoint

### Types of DOM change breakpoints

* **Subtree modifications**. Triggered when a child of the currently-selected node is removed or added, or the contents of a child are changed. Not triggered on child node attribute changes, or on any changes to the currently-selected node.
* **Attributes modifications**: Triggered when an attribute is added or removed on the currently-selected node, or when an attribute value changes.
* **Node Removal**: Triggered when the currently-selected node is removed.

## Overview of when to use each breakpoint type

The most well-known type of breakpoint is line-of-code. But line-of-code breakpoints can be inefficient to set, especially if you don't know exactly where to look, or if you are working with a large codebase. You can save yourself time when debugging by knowing how and when to use the other types of breakpoints.

|  |  |
| --- | --- |
| **Breakpoint Type** | **Use This When You Want To Pause...** |
| [Line-of-code](https://developer.chrome.com/docs/devtools/javascript/breakpoints/#loc) | On an exact region of code. |
| [Conditional line-of-code](https://developer.chrome.com/docs/devtools/javascript/breakpoints/#conditional-loc) | On an exact region of code, but only when some other condition is true. |
| [DOM](https://developer.chrome.com/docs/devtools/javascript/breakpoints/#dom) | On the code that changes or removes a specific DOM node, or its children. |
| [XHR](https://developer.chrome.com/docs/devtools/javascript/breakpoints/#xhr) | When an XHR URL contains a string pattern. |
| [Event listener](https://developer.chrome.com/docs/devtools/javascript/breakpoints/#event-listeners) | On the code that runs after an event, such as click, is fired. |
| [Exception](https://developer.chrome.com/docs/devtools/javascript/breakpoints/#exceptions) | On the line of code that is throwing a caught or uncaught exception. |
| [Function](https://developer.chrome.com/docs/devtools/javascript/breakpoints/#function) | Whenever a specific function is called. |