JavaScript Developer Tools

Introduction

The Developer Tools are useful for running JavaScript code, editing HTML and CSS styles without having to refresh the page, and viewing performance data.

https://developer.chrome.com/docs/devtools/overview

View and change the DOM

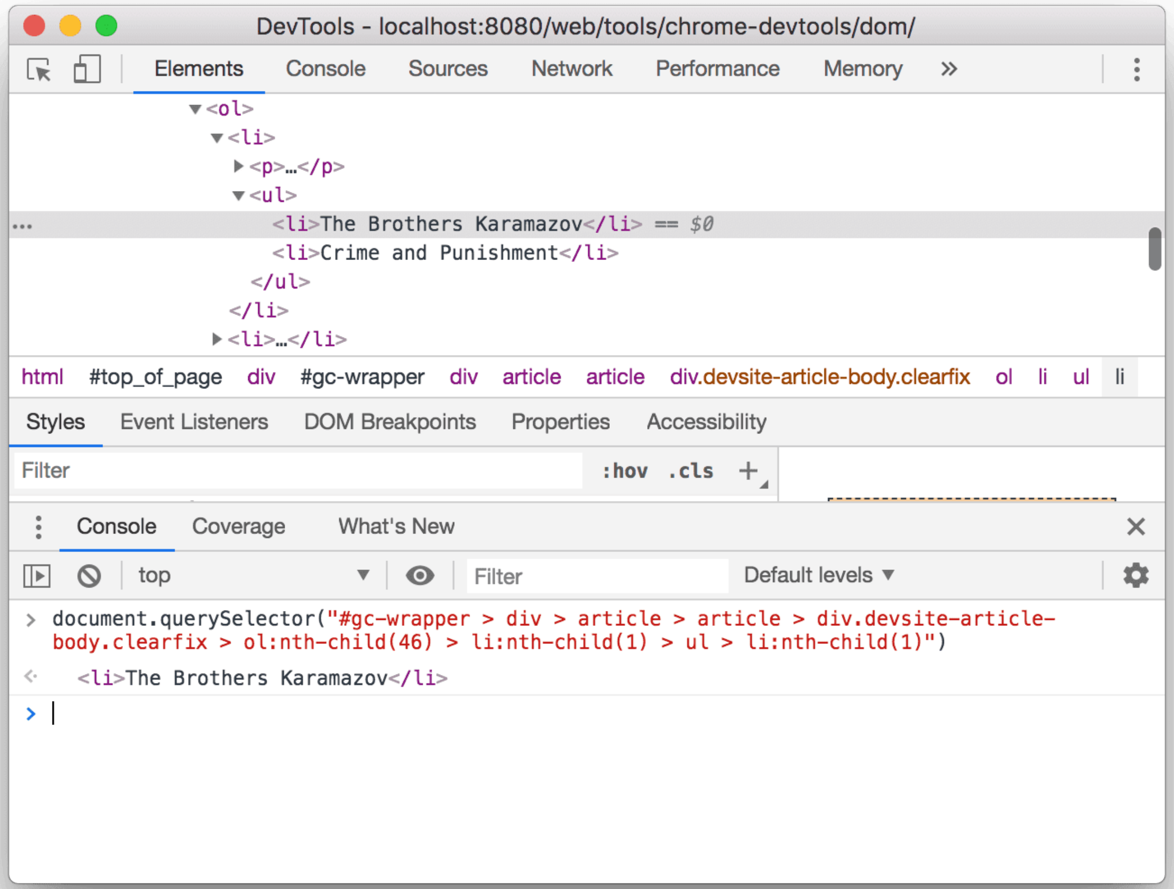
The DOM (Document Object Model) is the tree where the nodes are the HTML elements. The HTML represents initial page content and the DOM contains current (likely modified) page content.

The DevTools window show the DOM. You can edit the HTML and CSS here. `$0` is a shortcut that represents the current node. Use `dir($0)` to view the node in object format. Right-click to add/change attributes or edit as HTML. You can drag and drop the DOM nodes to move them around.

You can right-click any part of the page and inspect to bring up DevTools focused on the element.

If you need to refer back to a node many times, store it as a global variable.

You can copy a JS path that you can use later to reference the node in an automated test when you right click and go to "Copy". Then a `document.querySelector()` expression that resolves to the node has been copied to your clipboard (so you can paste it).



```
▼<ol>
  ▼<li>
    ▶<p>…</p>
    ▼<ul>
        <li>The Brothers Karamazov</li> == $0
        <li>Crime and Punishment</li>
      </ul>
    </li>
  ▶<li>…</li>
```

```
> document.querySelector("#gc-wrapper > div > article > article > div.devsite-article-
  body.clearfix > ol:nth-child(46) > li:nth-child(1) > ul > li:nth-child(1)")
<·   <li>The Brothers Karamazov</li>
>
```

View and change CSS

You can edit an element's style in the element.style section of the `Styles` pane. You can add a class with `.cls`. You can add a pseudostate by clicking :hov and choosing the desired one.

Simulate mobile devices with device mode

The second button on the top-left will start simulating the UI for a mobile viewport. The default dimensions are set to Responsive. You can change the dimensions directly by editing the values in Responsive mode/selecting a device in the dropdown menu or click a section on the series of tabs at the top to select a device with certain dimensions.

You can rotate the screen with the button with the rotating arrows. You can add/remove custom devices.

Throttling is for simulating a slow CPU and/or low bandwidth networks. Emulating mobile devices on a laptop or desktop has limitations because of architecture/hardware differences between mobile devices and laptops/desktops. So this emulation is an approximation.

We can work with media queries here by clicking on the 3 vertical dots. The blue bar has `max-width` breakpoints and the orange bar has `min-width` breakpoints.

Device pixel ratio (DPR) is the ratio between physical pixels on the hardware screen and logical (CSS) pixels. The DPR tells the browser how many screen pixels to use to draw a CSS pixel. Chrome, for instance, uses this when drawing on High Dots Per Inch displays. You can edit the DPR in the emulation.

You can adjust the zoom value by clicking on the 100% dropdown menu.

You can even test out geolocation, orientation, etc. features of your website.

Debugging JavaScript

First reproduce the bug by finding a series of actions that gets you the same erroneous result.

Thr `Sources` panel is where we debug JavaScript. On the top left, is the file navigator pane where you can see the files that the page uses. Click on a file to see it in the top right pane code editor pane. Underneath is the JavaScript debugging pane.

You can pause the code right when say an EventListener executes by using breakpoints. Set these under `Event Listener Breakpoints`. Then try performing that event such as a mouse click. The line of code you are frozen at will be shown. You can also set breakpoints at a specific line of code, when a DOM node changes, when an exception gets thrown, etc.

Once paused at say the `onClick()` function, you can set a future breakpoint and resume execution up to that line of code right before it executes. You can see the call stack to reach this function, locals, and globals. You can write test code in the Watch bar such as: `typeof sum` to see what the type of a variable named `sum` is to see if that is the correct type or not. You can play around further in the `Console` tab. The variables that are in scope wherever you paused will be available for working with here!

Once you found the bug and know how to fix it you can save with `Ctrl-S` and run without breakpoints by disabling breakpoints to see if it works now!

There are several advantages to using breakpoints as opposed to using console.log() print functions to debug:

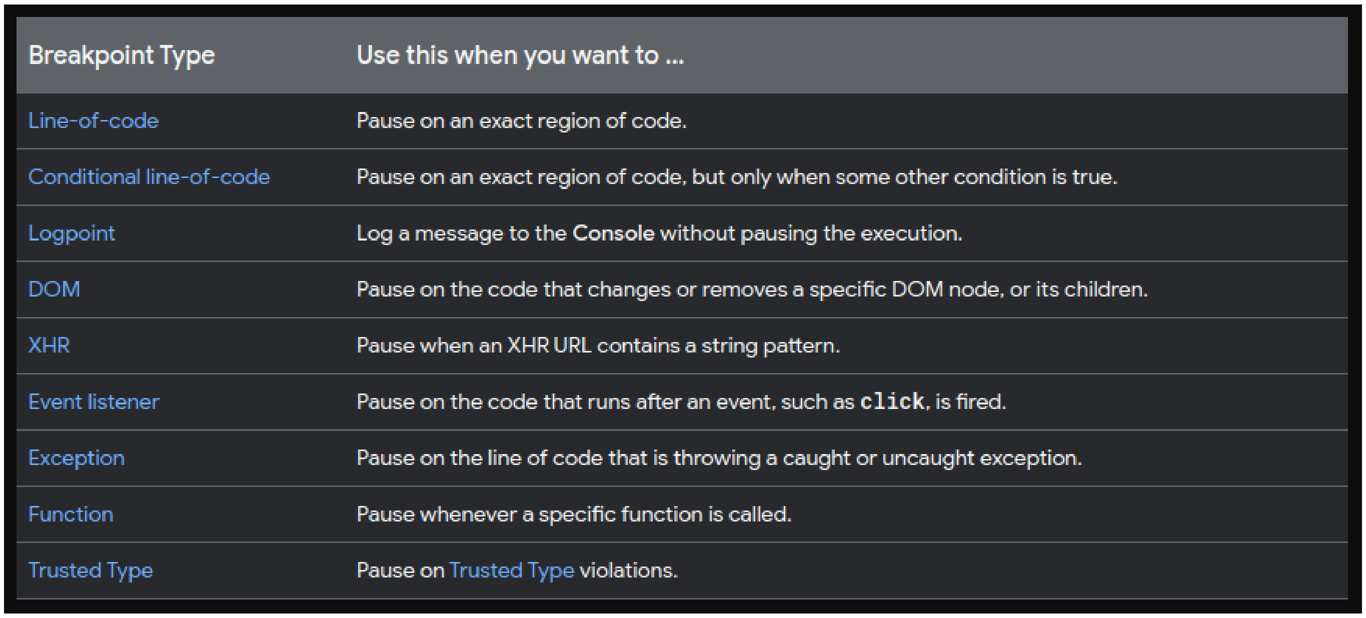
- With console.log(), you need to manually open the source code, find the relevant code, insert the console.log() statements, and then reload the page in order to see the messages in the Console. With breakpoints, you can pause on the relevant code without even knowing how the code is structured
- In your console.log() statements you need to explicitly specify each value that you want to inspect. With breakpoints, DevTools shows you the values of all variables at that moment in time. Sometimes there are variables affecting your code that you're not even aware of

Pause your code with breakpoints

Conditional breakpoints only pause when an expression you choose returns true. This is VERY handy for debugging a for loop. You can thus stop at a certain iteration.

Logpoints help you log some values without triggering a breakpoint. `Ctrl-Click` on a line, select logpoint. Enter expressions that you want to see.

You can break on exceptions, both caught and uncaught. An uncaught exception is an error that is not explicitly handled. A caught exception is one that is anticipated and handled. The debugger will stop on these if you check pause on uncaught and caught exceptions.



Breakpoint Type	Use this when you want to ...
Line-of-code	Pause on an exact region of code.
Conditional line-of-code	Pause on an exact region of code, but only when some other condition is true.
Logpoint	Log a message to the **Console** without pausing the execution.
DOM	Pause on the code that changes or removes a specific DOM node, or its children.
XHR	Pause when an XHR URL contains a string pattern.
Event listener	Pause on the code that runs after an event, such as `click`, is fired.
Exception	Pause on the line of code that is throwing a caught or uncaught exception.
Function	Pause whenever a specific function is called.
Trusted Type	Pause on Trusted Type violations.

You can add the line `debugger` to your code to set a breakpoint in your code itself (not the DevTools UI).

You can break on a DOM change by right-clicking and choosing `Break on`.

Types of DOM change breakpoints

- Subtree modificiations: 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.

XHR/fetch breakpoints

You can break when the requested URL of an XHR (XMLHttpRequest; a JavaScript API to create HTTP requests) contains a specific string. It pauses on the line of code where the XHR calls `send()`.

One example of when this is helpful is when you see that your page is requesting an incorrect URL, and you want to quickly find the AJAX or Fetch source code that is causing the incorrect request. For instance, you can break on any request that contains `org` in the URL by typing `org` into the text input when creating the XHR/fetch breakpoint.

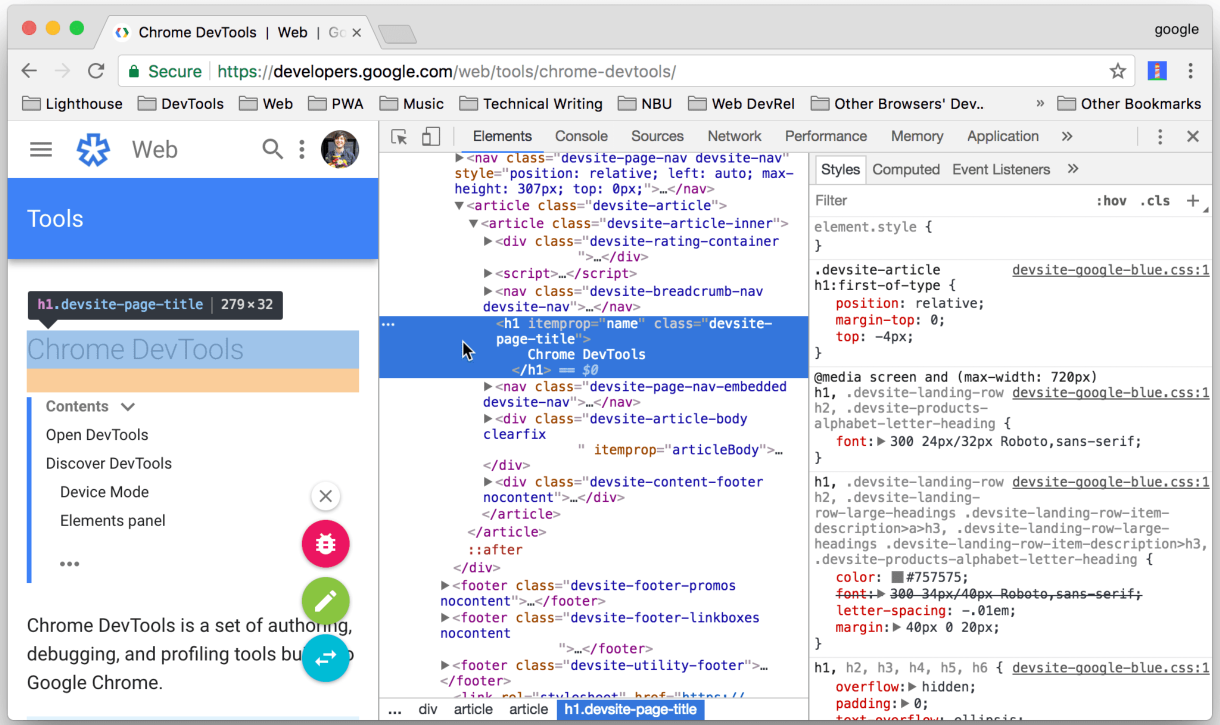
Function breakpoints

Call `debug(functionName)`, where `functionName` is the function you want to debug, when you want to pause whenever a specific function is called. You can insert `debug()` into your code (like a `console.log()` statement) or call it from the DevTools Console. `debug()` is equivalent to setting a line-of-code breakpoint on the first line of the function.

Make sure the function is in scope (important when you have nested functions) by setting a line-of-code breakpoint somewhere the function is in scope, triggering the breakpoint, and then calling `debug()` in the DevTools Console while the code is still paused on your line-of-code breakpoint.

Select an element

The **Elements** panel of DevTools lets you view or change the CSS of one element at a time.



Chrome DevTools is a set of authoring, debugging, and profiling tools bu... Google Chrome.

```
▶<nav class="devsite-page-nav devsite-nav"
style="position: relative; left: auto; max-
height: 307px; top: 0px;">…</nav>
▼<article class="devsite-article">
  ▼<article class="devsite-article-inner">
    ▶<div class="devsite-rating-container
              ">…</div>
    ▶<script>…</script>
    ▶<nav class="devsite-breadcrumb-nav
    devsite-nav">…</nav>
      <h1 itemprop="name" class="devsite-
      page-title">
          Chrome DevTools
        </h1> == $0
    ▶<nav class="devsite-page-nav-embedded
    devsite-nav">…</nav>
    ▶<div class="devsite-article-body
    clearfix
              " itemprop="articleBody">…
    </div>
    ▶<div class="devsite-content-footer
    nocontent">…</div>
    </article>
  </article>
  ::after
</div>
▶<footer class="devsite-footer-promos
nocontent">…</footer>
▶<footer class="devsite-footer-linkboxes
nocontent
              ">…</footer>
▶<footer class="devsite-utility-footer">…
</footer>
```

```
element.style {
}
.devsite-article          devsite-google-blue.css:1
h1:first-of-type {
   position: relative;
   margin-top: 0;
   top: -4px;
}
@media screen and (max-width: 720px)
h1, .devsite-landing-row devsite-google-blue.css:1
h2, .devsite-products-
alphabet-letter-heading {
   font:▶ 300 24px/32px Roboto,sans-serif;
}
h1, .devsite-landing-row devsite-google-blue.css:1
h2, .devsite-landing-
row-large-headings .devsite-landing-row-item-
description>a>h3, .devsite-landing-row-large-
headings .devsite-landing-row-item-description>h3,
.devsite-products-alphabet-letter-heading {
   color: ■#757575;
   ~~font:▶ 300 34px/40px Roboto,sans-serif;~~
   letter-spacing: -.01em;
   margin:▶ 40px 0 20px;
}
h1, h2, h3, h4, h5, h6 { devsite-google-blue.css:1
   overflow:▶ hidden;
   padding:▶ 0;
```

On the screenshot, the `h1` element that's highlighted blue in the **DOM Tree** is the selected element. To the right, the element's styles are shown in the **Styles** tab. To the left, the element is highlighted in the viewport, but only because the mouse is hovering over it in the **DOM Tree**.

There are many ways to select an element:

- In your viewport, right-click the element and select **Inspect**.
- In DevTools, click The top-left **Select an element arrow-pane combo** and then click the element in the viewport.
- In DevTools, click the element in the **DOM Tree**.
- In DevTools, run a query like `document.querySelecetor('p')` in the **Console**, right-click the result, and then select **Reveal In Elements panel**.

View CSS

Use the **Elements > Styles** and **Computed** tabs to view CSS rules and diagnose CSS issues.

Elements > Styles shows tooltips with useful information when you hover over specific elements.

Hover over a selector to see a tooltip with its specificity (3 numbers for ID, CLASS, TYPE) weight.

View only the CSS that's actually applied to an element

The **Styles** tab shows you all of the rules that apply to an element, including declarations that have been overridden. When you're not interested in overridden declarations, use the **Computed** tab to view only the CSS that's actually being applied to an element. Check the **Show All** checkbox to see all properties.

Search and filter an element's CSS

Use the **Filter** box on the **Styles** and **Computed** tabs to search for specific CSS properties or values.

To navigate the **Computed** tab, group the filtered properties in collapsible categories by checking **Group**.

View cascade layers

Cascade layers enable more explicit control of your CSS files to prevent style-specificity conflicts.

Check the **Styles** tab and click on the Layer to see the specificity order.

Copy CSS

From a single drop-down menu in the **Styles** tab, you can copy separate CSS rules, declarations, properties, or values.

Add a CSS declaration to an element

Add a CSS declaration in the element.style section.