Code Quality

16th July 2020

1). Debugging in chrome.

Debugging is the process of finding and fixing errors within a script. All modern browsers and most other environments support debugging tools – a special UI in developer tools that makes debugging much easier. It also allows to trace the code step by step to see what exactly is going on.

We'll be using Chrome here, because it has enough features, most other browsers have a similar process.

2). "Sources" panel.

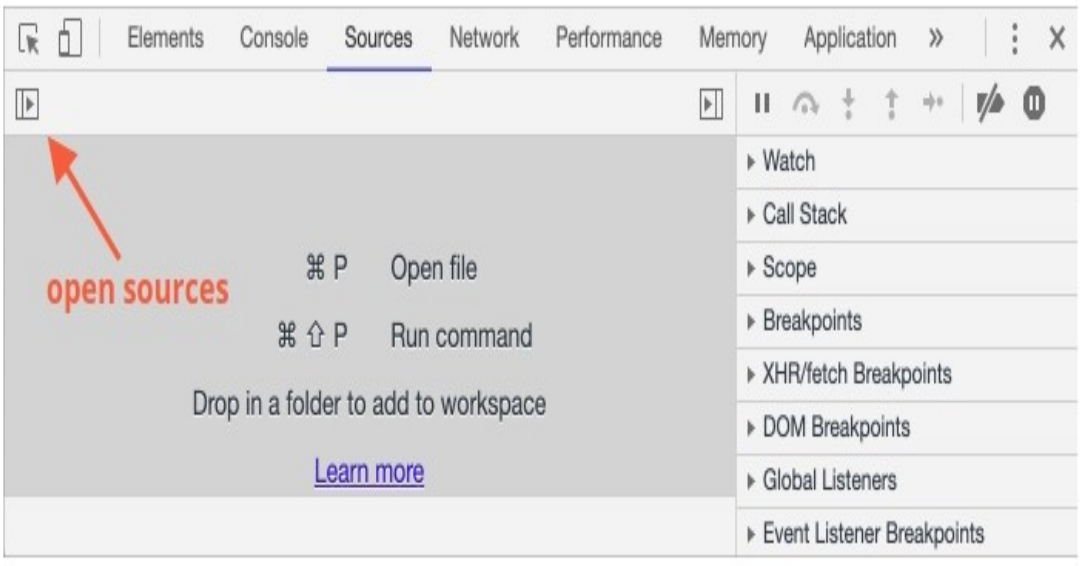
Your Chrome version may look a little bit different, but it still should be obvious what's there.

To open the sources panel follow the following steps:

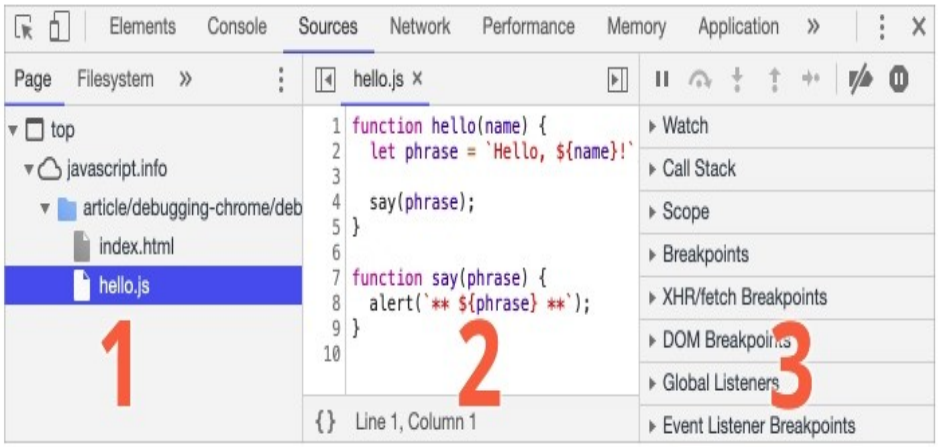
a). open our app webpage in chrome or any other browser of your choice.

b). Turn on developer tools with F12 (Mac: Cmd+Opt+I).

c). Select the Sources panel.



The toggler button (open sources) opens the tab with files. Click it and select hello.js in the tree view. Here's what should show up:



The Sources panel has 3 parts:

1. The File Navigator pane lists HTML, JavaScript, CSS and other files, including images that are attached to the page. Chrome extensions may appear here too.

2. The Code Editor pane shows the source code.

3. The JavaScript Debugging pane is for debugging, we'll explore it soon.

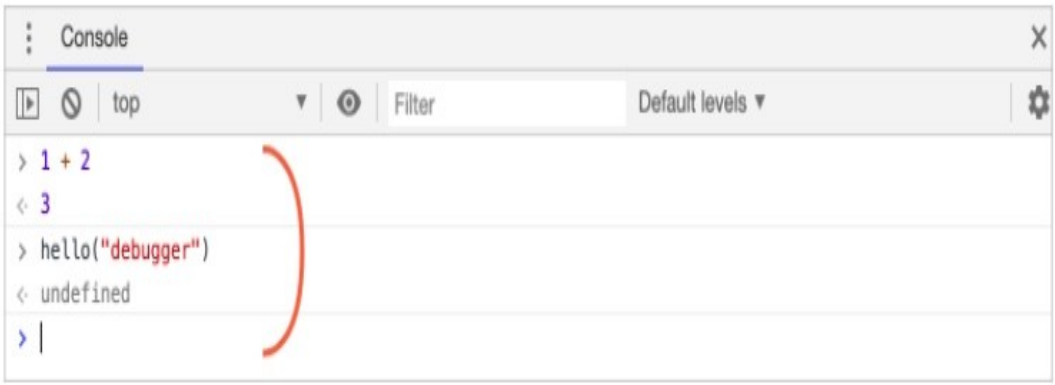
When done exploring it you can click the same toggler again to hide the resources list and give the code some space.

3). **Console**.

If we press **Esc**, then a console opens below. We can type commands there and press **Enter** to execute.

After a statement is executed, its result is shown below.

For example, here 1+2 results in 3, and hello("debugger") returns nothing, so the result is undefined:



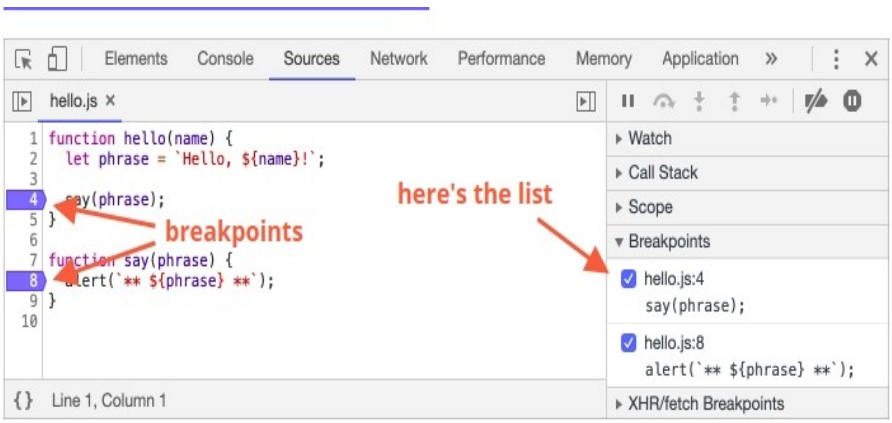
4). **BreakPoints**

A breakpoint is a point of code where the debugger will automatically pause the JavaScript execution.

Let's examine what's going on within the code of the app webpage In hello.js, click at line number `4`. Yes, right on the `4` digit, not on the code.

Congratulations! You've set a breakpoint. Please also click on the number for line `8`.

It should look like this (blue is where you should click):



While the code is paused, we can examine current variables, execute commands in the console etc. In other words, we can debug it.

We can always find a list of breakpoints in the right panel. That's useful when we have many breakpoints in various files. It allows us to:

1). Quickly jump to the breakpoint in the code (by clicking on it in the right panel).

2). Temporarily disable the breakpoint by unchecking it.

3). Remove the breakpoint by right–clicking and selecting Remove.

5). **Debugger command**.

We can also pause the code by using the debuggercommand in it, like this:

```
function hello(name) {
  let phrase = `Hello, ${name}!`;

  debugger;  // <-- the debugger stops here

  say(phrase);
}
```

That's very convenient when we are in a code editor and don't want to switch to the browser and look up the script in developer tools to set the breakpoint.

– **Mocha**.

Mocha is an open source test framework that is used to run your automated tests in Node. It comes with a wide range of features that allow you to create descriptive automated tests, robust reports and even execute your automated tests every time a file is changed locally.

We can use Mocha to run any type of test we like within describe functions by adding code and assertions to determine a pass or fail status. Whether it's a low level Unit checks against back end services or JavaScript functions or full stack tests focused on integration of components, Mocha provides the structure for running your tests.

– **Eslint**.

ESLint is a linter for JavaScript. Linters will analyze code as it is written and identify a variety of basic syntax problems. The use of ESLint will allow you to catch errors, particularly easy to resolve but annoying ones such as missing brackets or typos, before executing the code. ESLint is available as a Node package. It has also been set up as a plugin for many code editors such as Sublime Text 3 and VS Code, which will then mark the offending errors right in your editor window.

– **Raygun**

Tired of spending time digging through logs to find your JavaScript errors? Raygun Crash Reporting is the answer, and has everything you need to find and assess the impact of JavaScript bugs and performance problems. It's quick and easy to set up. Clink this https://app.raygun.com/signp for a free trial of Raygun sign up.

Get your APIs keys from your Raygun dashboard and paster it before your body tags. Now Raygun will begin collecting data and notifying you of issues

Conclusion

Building good and comprehensive tests for your JavaScript applications helps enable stable and robust applications. Good tests also serve as valuable documentation for what your applications are actually supposed to be doing.

JavaScript debugging can be a challenge, and having the right tools available to you can make or break the debugging process.

Every application and use case will be different, but learning about new tools will help you find out what your best options might be.

JavaScript debugging is easier when you know where the error is! Raygun helps developers detect, diagnose, and resolve issues with greater speed and accuracy

"Programming allows you to think about thinking, while debugging you learn learning" Nicholas Negroponte

Best wishes Lux tech academy

Email: luxtinc@gmail.com