JavaScript Debugging in PhpStorm

Ever wondered what's going on in the browser's memory and the DOM when running JavaScript? With PhpStorm, we can debug JavaScript code and step through it line by line, inspecting variables (and even changing them) at runtime. Let's see how.

Next to debugging JavaScript, we can also debug PHP applications with PhpStorm.

1. Installing the JetBrains Chrome extension

To debug JavaScript with PhpStorm, a Chrome extension is used. This extension provides the link between what happens in the IDE and what happens in the browser, and vice-versa. We may have to install this extension manually.

If the browser is closed when starting a debugging session in the IDE, PhpStorm will install the extension automatically. This step can be skipped by closing all browser windows the first time a debugging session is started after installing PhpStorm.

Installing the JetBrains Chrome extension can be done as follows:

1. Open Chrome, go to the settings and click the Extensions item. Alternatively, open Chrome and navigate to chrome://extensions/.
2. Search for the JetBrains IDE Support extension and click the install button (+ FREE near the top).
3. Confirm adding the new extension to Chrome.

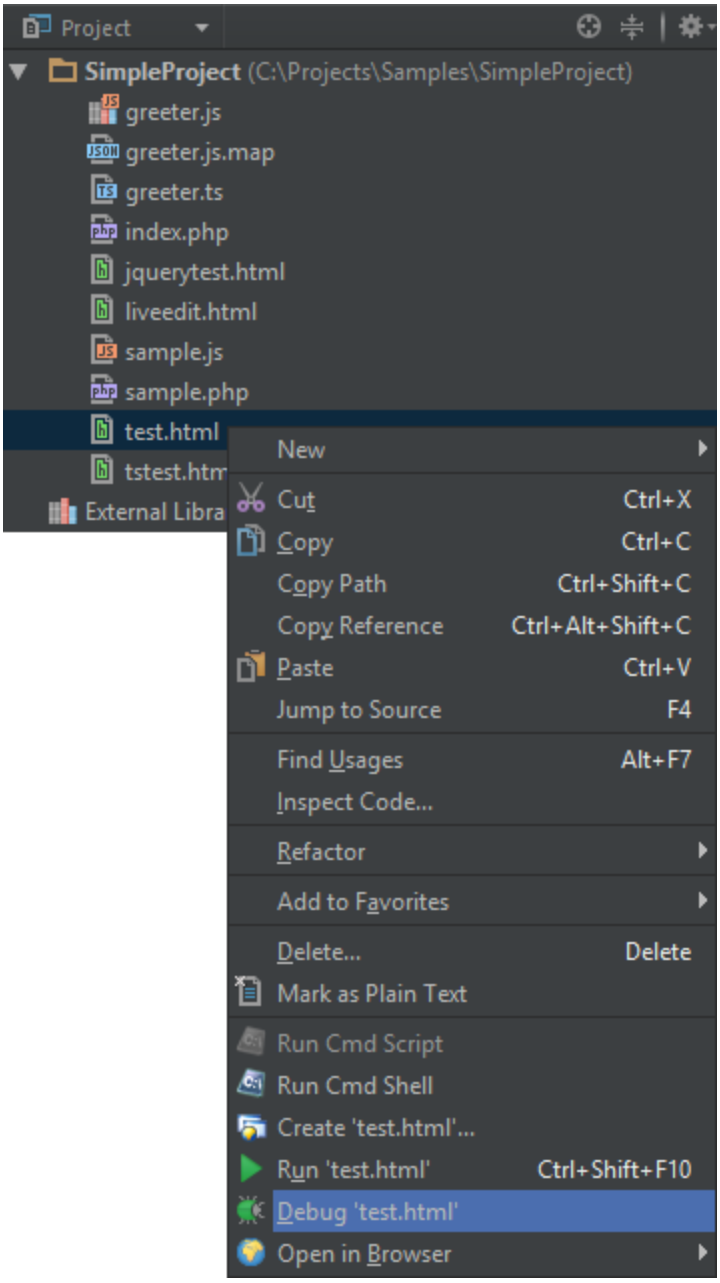
Once installed, we're good to go.

2. Setting Breakpoints

In the editor, we can set breakpoints. Breakpoints are lines of code in which the debugger will pause execution and allows us to see what's going on inside our code. We can set breakpoints by clicking the gutter at a specific line of code or by placing the cursor on a line of code and using the keyboard shortcut, Ctrl+F8 or CMD+F8 on Mac OS X, to toggle the breakpoint. The IDE shows breakpoints as a big red dot in the left gutter.

3. Starting a Debugging Session

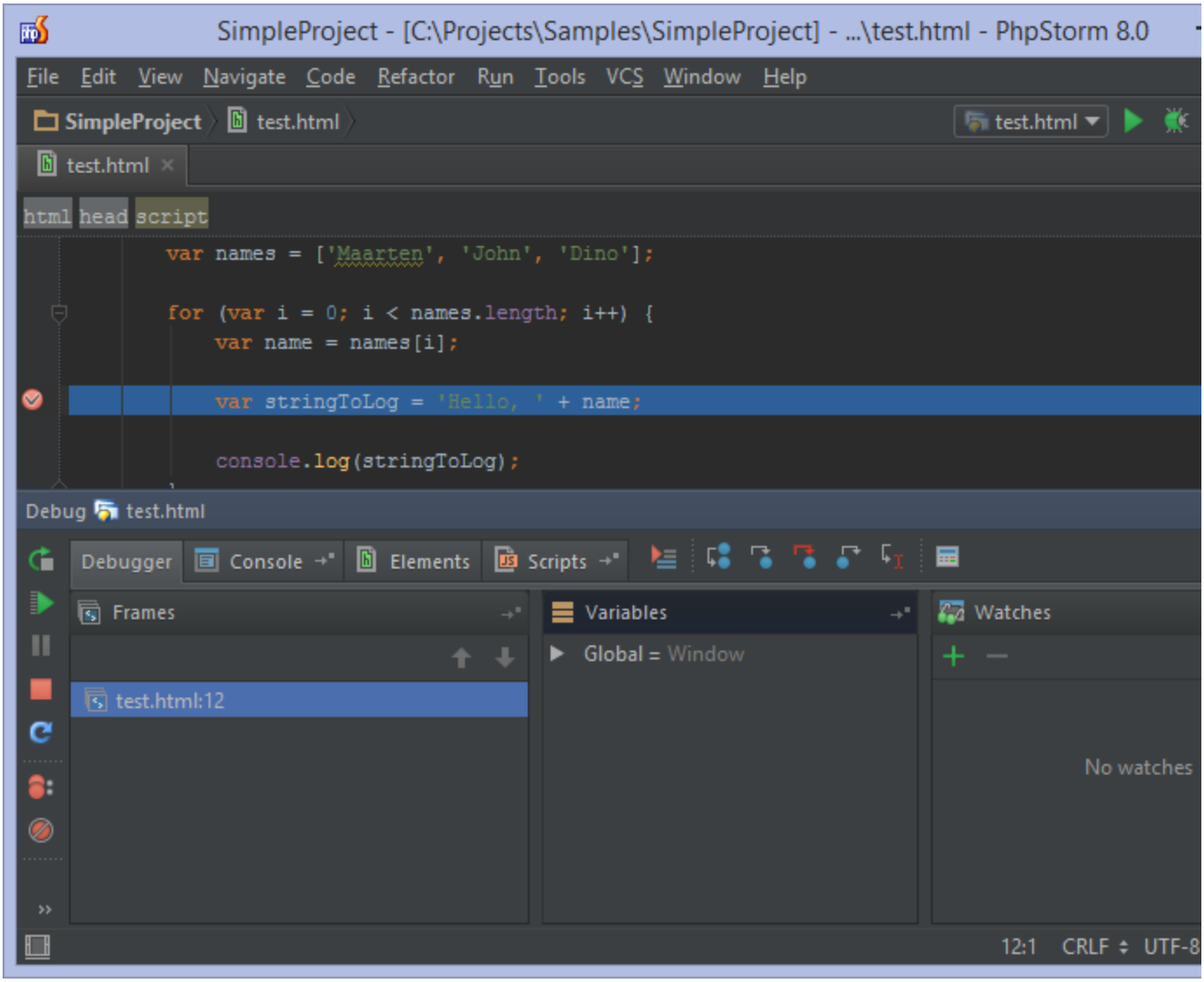
When we've added a breakpoint, we can start a debugging session. This can be done by creating a JavaScript Debug Configuration, something we can do manually from the toolbar, or by using the context menu on the file we want to debug and clicking Debug '<filename'>.



This will create the debug configuration for us, and will also launch the browser.

4. Debugging!

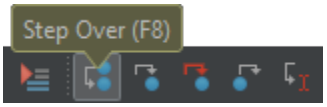
Once the browser has been launched, we can start debugging. PhpStorm will pause execution at our breakpoint and highlight the line of source code which is about to be executed.



In the IDE, a new tool window is opened. From here, we can navigate through the running code and inspect variables.

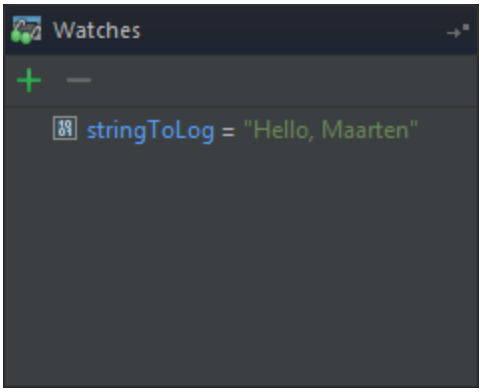
4.1. Stepping through Code

Using the toolbar buttons or the keyboard shortcuts, we can step through code. Step Into (F7) follows every statement and lets us inspect them one by one. Step Over (F8) just executes the active statement as a whole without stepping deeper down the execution stack. If we just want to run until the next breakpoint, we can do so pressing F9.



4.2. Watches

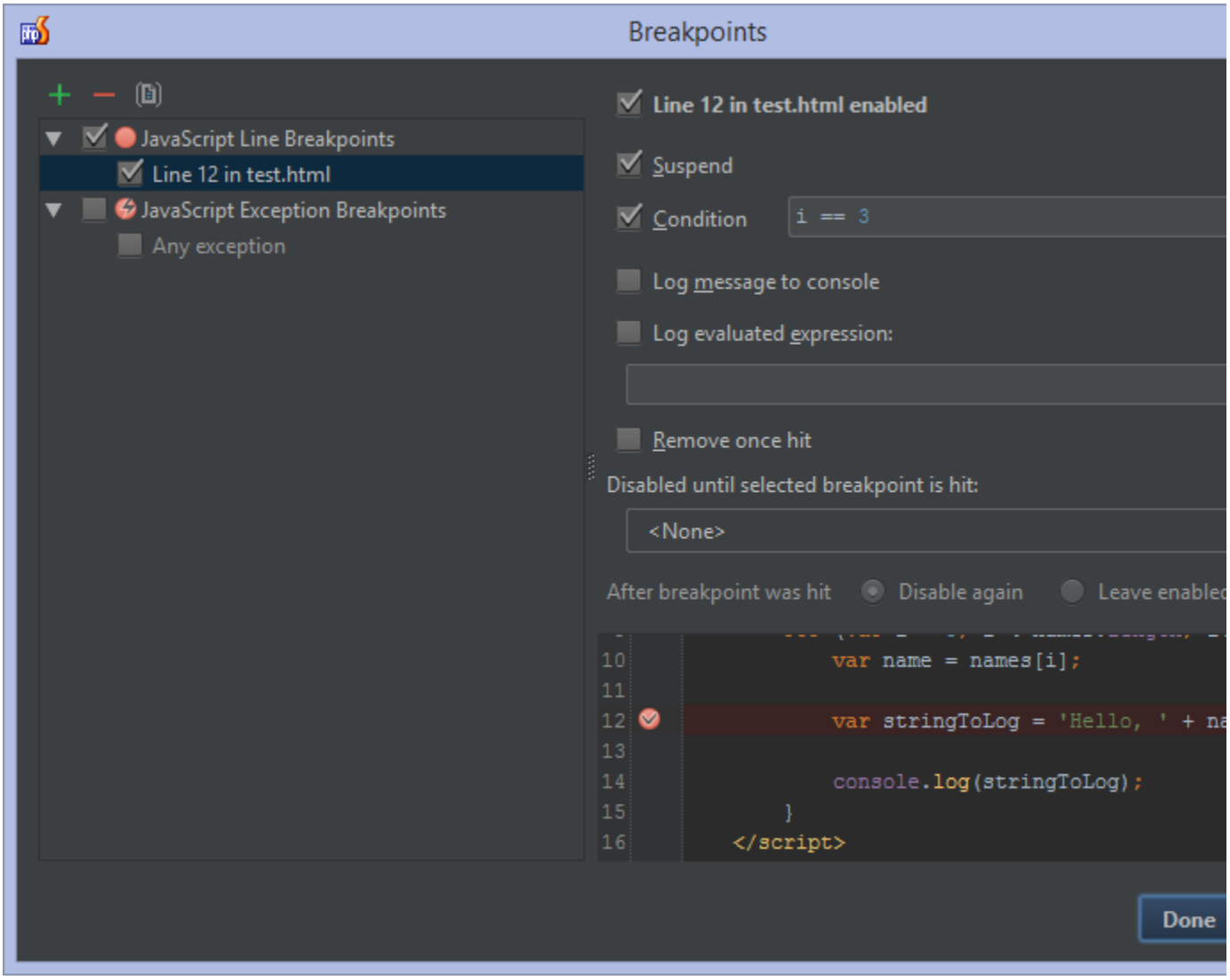
If we want to pin a variable for inspection, we can add it to the watches list so we can see the value of that variable whenever it is in scope while stepping through our code. This makes it easier to just inspect the variables we are interested in.



4.3. Breakpoint Options and Conditional Breakpoints

Imagine debugging a loop. If we place the breakpoint inside the loop body, it will be hit with every iteration. If we are only interested in seeing what's going on in our code for a specific iteration, we can make the breakpoint a conditional breakpoint.

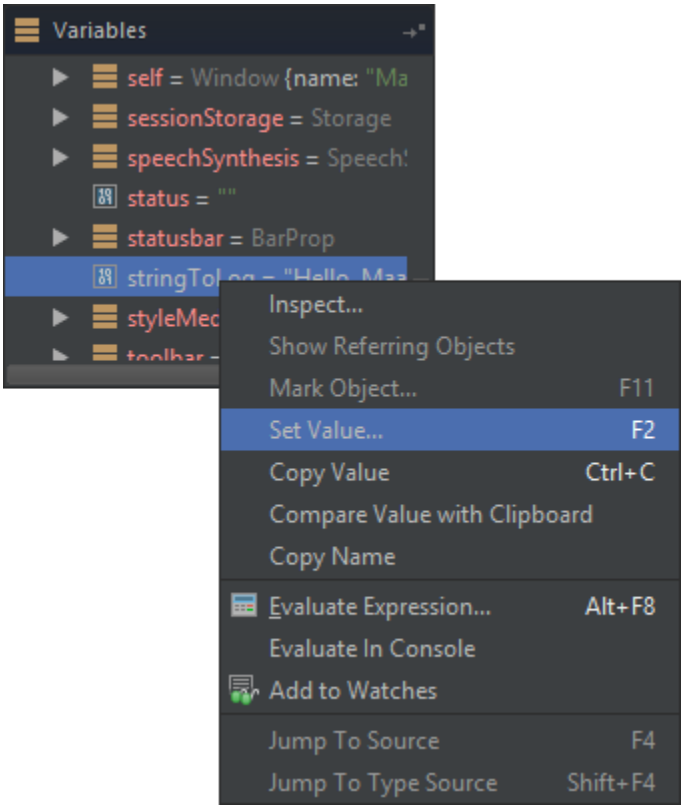
Using the Run | View Breakpoints menu or press Ctrl+Shift+F8 (CMD+Shift+F8 on Mac OS X), we can specify the condition when the breakpoint is valid and the debugger should pause execution. For example, when our iterator variable i is equal to 3.



We can do much more with breakpoints! As we can see in the previous screenshot, we can log a message to console when the breakpoint is hit, optionally not pausing execution but just showing us a trace of the code execution. We can also remove a breakpoint after it has been hit, or enable it only if another breakpoint was hit first.

4.4. Altering Variables while Executing Code

While debugging, it may be useful to be able to alter variables while executing code. For example, we can change a variable to test a specific condition. Variables can be edited at runtime by using the Set value... context menu in the debug tool window, or using the F2 keyboard shortcut.



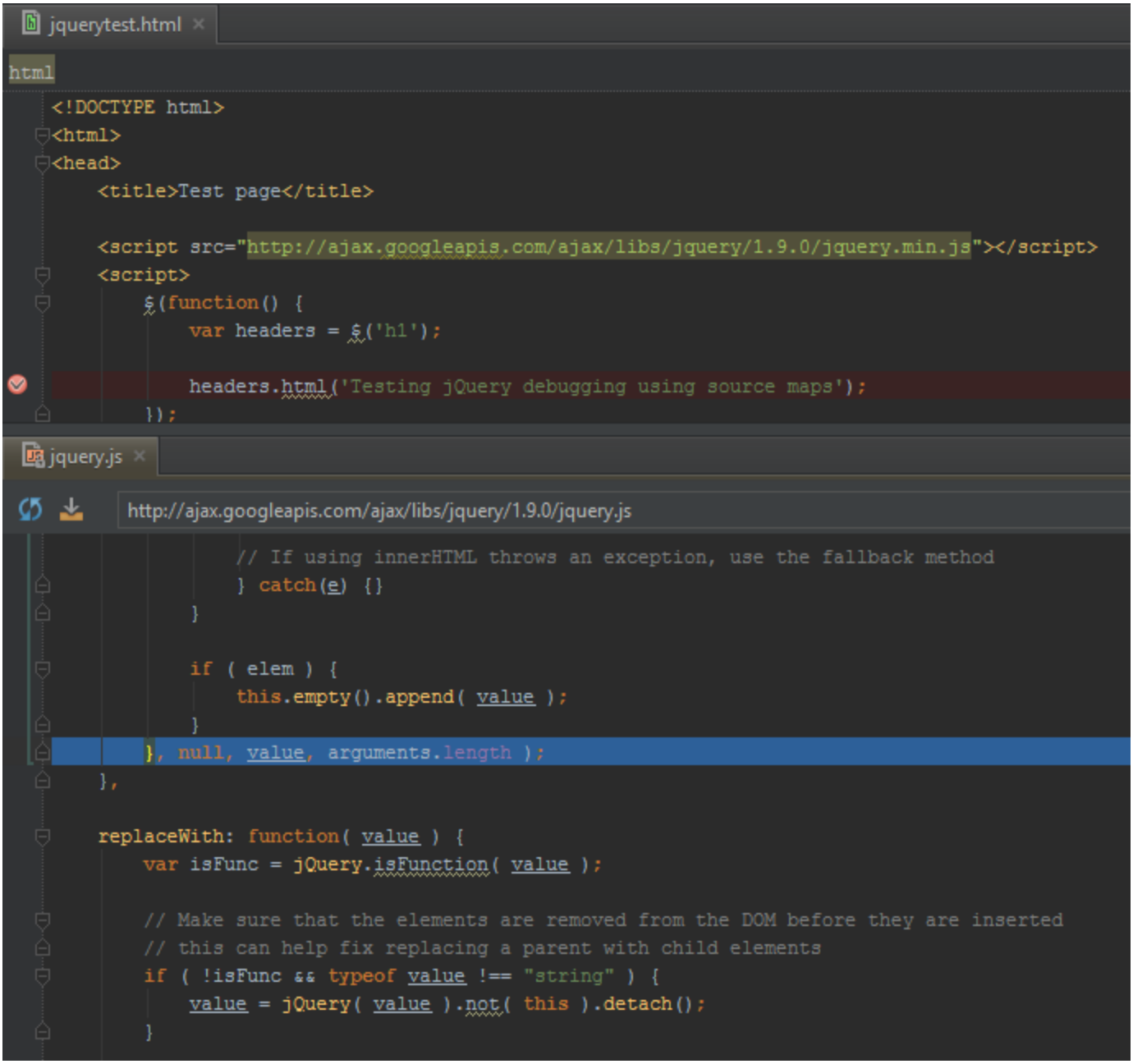
We can then enter any value we want the variable to contain and continue execution of our code with it.

Using Evaluate Expression..., we can evaluate any expression during runtime. This can be useful to quickly test a condition, call another function and so on.

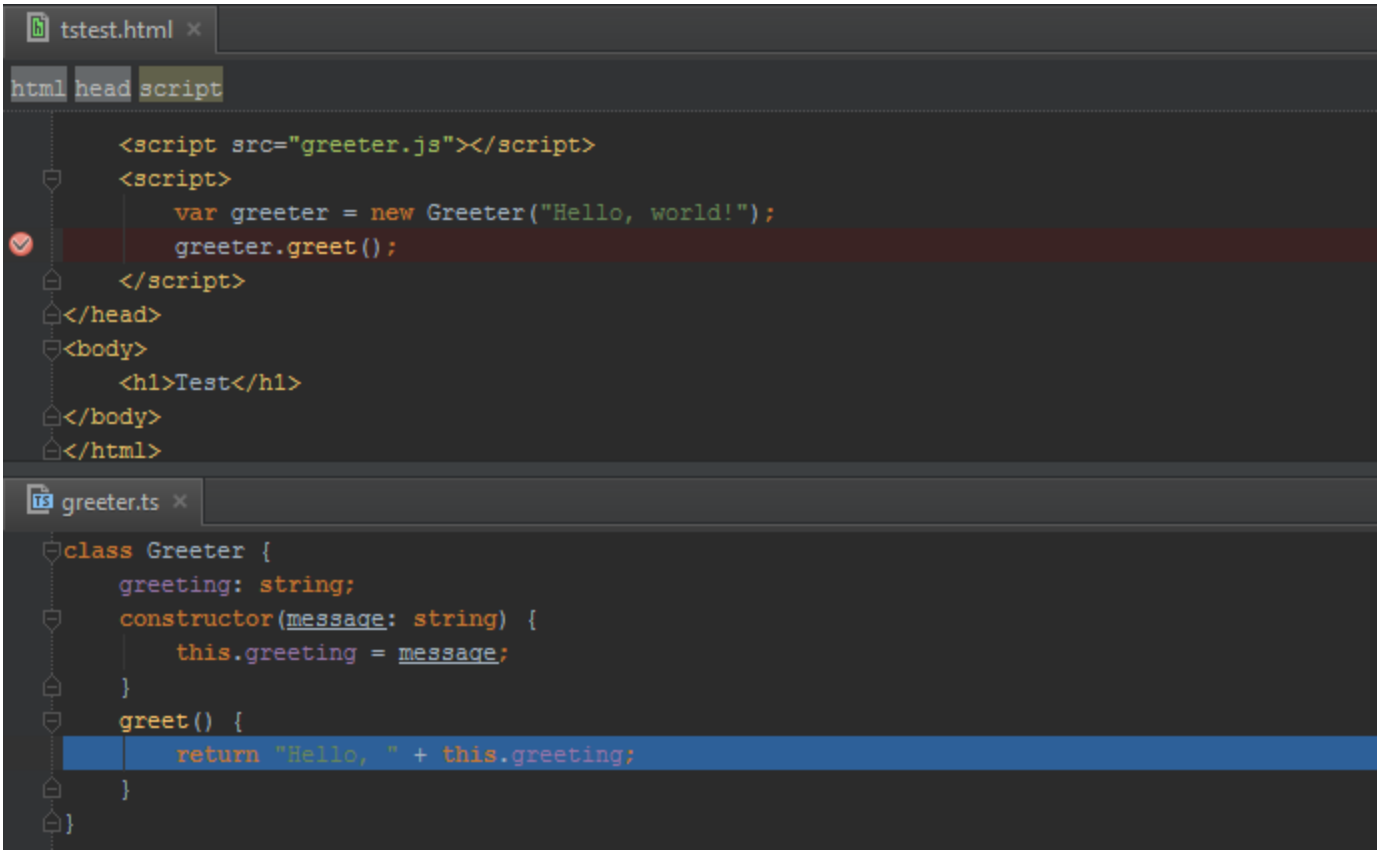
(optional) Source maps

The JavaScript debugger also allows us to work with source maps. This comes in very handy when using minified JavaScript, or languages like CoffeeScript or TypeScript that compile to JavaScript.

For example when using a minified version of jQuery, we can start debugging and step into jQuery code. Even though it is minified, PhpStorm will recognize the source map is present and will show us the full jQuery code instead of the minified version.



When working with TypeScript or CoffeeScript, languages that translate into JavaScript, the idea of source maps is also used. We can set breakpoints in CoffeeScript or TypeScript code and start a debugging session. The debugger will know how to map the executed JavaScript into the original file and hits the breakpoints in CoffeeScript or TypeScript, even if the browser is actually executing JavaScript code.



(optional) Spy-js

If we enable the Spy-js plugin in PhpStorm, we can debug it without breakpoints and profile it without any specialized tools. Spy-js uses historical execution data to provide us with a debugging experience after we ran our code. Read more about Spy-js on our blog.

Tweet