Debugging with Spy-Js

Spy-js is a tool for JavaScript developers that allows to simply debug/trace/profile JavaScript running on different platforms/browsers/devices. The tool fills gaps that existing browser development tools have and tackles common development tasks from a different angle.

On this page:

Preparing to Debug with Spy-js

1. Download and install Node.js. The framework is required for two reasons:
   - The Spy-js tracing tool is started through *Node.js*.
   - *NPM*, which is a part of the framework, is also the easiest way to download the Spy-js tracing tool.

   For details on using *Node.js* in IntelliJ IDEA, see the section Node.js

   > Alternatively, you can define Node.js as an external tool, as described in the section Configuring third-party tools. This approach is helpful, when you need facilities that are missing in the plugin, for example, the possibility to pass certain parameters as wildcards.

2. If you are going to use the command line mode, make sure the following paths are added to the `PATH` variable:
   1. The path to the parent folder of the *Node.js* executable file.
   2. The path to the `npm` folder.

   This enables you to launch the Spy-js tracing tool and *npm* from any folder.

3. Install and enable the *NodeJS* repository plugin.

   The plugin is not bundled with IntelliJ IDEA, but it is available from the JetBrains plugin repository. Once enabled, the plugin is available at the IDE level, that is, you can use it in all your IntelliJ IDEA projects. See Installing, Updating and Uninstalling Repository Plugins and Enabling and Disabling Plugins for details.

4. Make sure the *Spy-js* plugin is enabled. The plugin is bundled with IntelliJ IDEA and activated by default. If it is not, enable it as described in Enabling and Disabling Plugins.

Spy-js Basics

In order to trace a website script, *Spy-js* has to modify it on the fly. The modification does not affect the logic of the script, it is just inserting instrumentation instructions that report back to *Spy-js* UI about what functions have been invoked when the script executes. To modify your scripts, *Spy-js* has to act as a proxy server that "sits" between your browser and the website you are tracing. When you open a traced website in your browser, *Spy-js* receives the script request, requests the script from your website, receives the script, makes the required modifications, and sends it back to your browser where the script executes, and sends the runtime information to the *Spy-js* UI.

The proxy server can be configured automatically by selecting the **Automatically configure system proxy** check box in the Run/Debug Configuration: Spy-Js dialog or manually. See how to configure proxy settings manually on Windows, Mac, Ubuntu, iOS, Android, Windows Phone. Please note that some desktop browsers have their own screens for proxy settings configuration.

A *Spy-js* session is initiated from IntelliJ IDEA through a run configuration, see Initiating a Spy-js Session.

Spy-js UI

All the tracing-related activities, such as viewing captured events, examining their call stacks, navigating to the source code, etc. are performed in the dedicated Spy-Js Tool Window, in particular in its Trace Run Tab. The tab consists of a toolbar and three panes:

- **Events Pane**

  The pane shows a tree of captured events. The top-level nodes represent *documents* that are Web pages involved in tracing. When you hover the mouse over a *document*, IntelliJ IDEA displays a tooltip with the URL address of the *document*, the browser in which it is opened, and the operating system the browser is running on. The *document* node is also supplied with an icon that indicates the browser in which it is opened.

  Under each *document* node, events detected on the page and scripts started from it are listed. Events of the same type are grouped into visual containers. The header of a container displays the name of the events grouped in it, the average execution time across all the events within the container, and the number of events inside the container. You can expand each node and inspect the individual events in it.

  Script file names have different colour indicators to help distinguishing between them when working with the **Event Stack** pane. By hovering your mouse over a script file name, you can see the full script URL.

  Once an event is clicked, its call stack is displayed in the **Event Stack** pane. The stack is represented by a tree of function calls.

- **Event Stack Pane**

  Once an event in the **Events** pane is clicked, its call stack is displayed in the **Event Stack** pane. The stack is represented by a tree of function calls. Each tree node represents the invoked function. Node text contains the total execution time, the script file name and the function name. When you click a node, the **Quick Evaluation** pane shows additional function call details, such as the number of statements and invoked functions, parameter values and return value, occurred exception details if there was one during the function execution.

  Some tree nodes (functions that called other functions) can be expanded. If the expanded tree node contains too many children (invoked functions), some of them will not be displayed immediately. To see those children, click the link with "X item(s)" text in the desired tree level. Alternatively, hover your mouse over the information icon next to the "X item(s)" link and you will see a brief list of invoked functions without having to expand the node.

  The pane is synchronized with the editor, so you can navigate from an item in the stack tree to the corresponding *trace file* or *source file*.

  - A *trace file* is a write-protected prettified version of the script selected in the **Events** pane or the script whose function is double clicked in the **Event Stack** pane. A *trace file* is named `<file name>.js.trace`. When you double click an item in the stack tree or select it and choose **Jump to Trace** on the context menu of the selection, the corresponding *trace file* opens in the editor with the cursor positioned at the clicked function. Another approach is to press the **Autoscroll to Trace** toggle button. In this case, the trace file opens when you click an event or script in the **Events** pane.

    You can not only jump to a function but also to the place in the code where it was called from. To do that, select the required item and choose **Jump to Caller** on the context menu.

    The contents of the file are highlighted.

  - You can also navigate to the *source file* displayed as as, without prettifying , by selecting an item in the **Event Stack** pane and choosing **Jump to Source** on the context menu of the selection. If the traced site is mapped with a IntelliJ IDEA project, IntelliJ IDEA detects the corresponding local file according to the mapping and opens this file in the editor. If you are tracing a site that is not mapped to a IntelliJ IDEA project, IntelliJ IDEA opens the read-only *page source*, just as if you chose **View Page Source** in the browser.

- **Quick Evaluation Pane**

  When you click a node in the **Event Stack** pane, the **Quick Evaluation** pane shows additional function call details, such as the number of statements and invoked functions, parameter values and return value, occurred exception details if there was one during the function execution.

Initiating a Spy-js Session

1. Create a run configuration. To do that, choose **Run | Edit Configurations** on the main menu, click the **Add New Configuration** toolbar button +, and choose **Spy-js** on the context menu. In the Run/Debug Configuration: Spy-js dialog box that opens, specify the following:
   1. The location of the Node interpreter.
   2. The *trace server port*. This port number must be the same as your system proxy port. If the **Automatically configure system proxy** check box is selected, the specified port number is automatically set for the system proxy server. Otherwise you will have to specify the value of the field in the system proxy settings manually.

      The *trace server port* is filled in automatically. To avoid port conflicts, it is recommended that you accept the suggested value and keep the **Automatically configure system proxy** check box selected.
   3. The way to configure the proxy server.

      To have the system proxy server activated automatically with the port specified in the **Trace server port** field, select the **Automatically configure system proxy** check box.

      Clear the **Automatically configure system proxy** check box to specify proxy settings manually. See how to configure proxy settings manually on Windows, Mac, Ubuntu, iOS, Android, Windows Phone. Please note that some desktop browsers have their own screens for proxy settings configuration.
   4. From the **Use** dropdown list, choose the way to specify the way to configure a tracing session.
      - To have *Spy-js* apply its internal predefined configuration, choose **Default configuration**.
      - To have your custom manually created configuration applied, choose the **Configuration file** option and then specify the location of your custom configuration file in the **Configuration** field below. A configuration file is a JavaScript file with the extension `.js` that contains valid JavaScript code that meets the Spy-js configuration requirements.

        Type the path to the configuration file manually or click the **Browse** button ... and choose the location in the dialog box that opens. Once specified, a configuration file is added to the drop-down list so you can get if next time from the list instead of specifying the path.
2. To launch a tracing session, click the **Run** toolbar button ▶. The Spy-Js Tool Window opens with an empty **Trace Run** tab and a **Trace Proxy Server** tab informing you about the status of the proxy server.
3. Switch to the browser and refresh the page to start debugging from. *Spy-js* starts capturing events on this page and the **Spy-js** tool window shows them in the **Events** pane.

Configuring the Range of Events to Display

By default, the *Spy-js* tool captures all events on all opened Web pages, excluding *https secure* web sites, unless you have specified a URL address explicitly in the run configuration. The **Events** pane of the **Spy-js** tool window shows all captured events. If for some reasons you do not want to have all events captured, you can suppress capturing some of them by applying user-defined event filters. All the available filters are listed upon clicking the **Capture Events** button on the toolbar, the currently applied filter is marked with a tick. By default the **Capture All** predefined filter is applied. You can define new custom filters or add event patterns to existing filters on the fly.

Defining a new event filter

1. Click the **Capture Events** button on the toolbar, and then choose **Edit Capture Exclusions** from the list.
2. In the Spy-Js Capture Exclusions Dialog that opens, click the **Add** toolbar button in the left-hand pane.
3. In the right-hand pane, specify the filter name in the **Exclusion name** field and configure a list of exclusion rules.

   To add a rule, click , the **Add Condition to Exclusion** dialog box opens. Type a pattern in the **Value/pattern** text box, in the**Condition type** drop-down list specify whether the pattern should be applied to event types or script names. When you click **OK**, %product^ brings you to the Spy-Js Capture Exclusions Dialog.

   To edit a rule, select it in the list, click , and update the rule in the dialog box that opens. To remove a rule, select it in the list and click .

Creating exclusion rules on the fly

While navigating through the tree of already captured events in the **Events** pane, you may come across some events or scripts that you definitely do not want to trace. You can create a filter as described above but in this case you will have to leave the pane. With IntelliJ IDEA, you can create an exclusion rule based on any event or script, as soon as you have detected such event or script, right from the **Events** pane. The rule will be either added to the currently applied filter or a new filter will be created if the current setting is **Capture All**.

- To add an event to an exclusion filter on the fly, select the event to exclude and choose **Mute <event name> event** or **Mute <script name> file**.

If a user-defined filter is currently applied, the new rule is added to it silently. If **Capture All** is currently active, the Spy-Js Capture Exclusions Dialog opens, where you can create a new filter based on the selected event or script or choose an existing filter and add the new rule to it.

Synchronization and Navigation between the Panes and the Editor

The **Events** and **Event Stack** panes are synchronized: when you click an event or script in the **Events** pane, its call stack is displayed in the **Event Stack** pane. To have also the corresponding trace file opened in the editor, press the **Autoscroll to Trace** toggle button on the toolbar.

The **Event Stack** pane is synchronized with the editor: when you click an item in the stack tree twice, the corresponding trace file opens in the editor with the cursor positioned at the clicked function.

To synchronize the **Events** pane directly with the editor, press the **Autoscroll to Trace** toggle button on the toolbar. In this case, as soon as you click a node in the **Events** pane, its call stack is displayed in the **Event Stack** pane and the corresponding trace file is opened in the editor.

Navigating from an event or a script to the trace file

A *trace file* is a write-protected prettified version of the script selected in the **Events** pane or the script whose function is double clicked in the **Event Stack** pane. A *trace file* is named `<file name>.js.trace`. To navigate from an event or a script to the trace file, do one of the following:

- Activate automatic navigation:
  1. In the **Events** pane, press the **Autoscroll to Trace** toggle button on the toolbar.
  2. Click the required event or script.
- In the **Event Stack** pane, click the required item in the call stack twice or choose **Jump to Trace**. on the context menu. The trace file opens with the cursor positioned at the clicked function.

Navigating from an event or script to the source file

You can also navigate to the *source file* displayed as as, without prettifying . To navigate from an event or script to the source file:

- In the **Event Stack** pane, select the required item in the call stack and choose **Jump to Source** on the context menu of the selection.

If the traced site is mapped with a IntelliJ IDEA project, IntelliJ IDEA detects the corresponding local file according to the mapping and opens this file in the editor. If you are tracing a site that is not mapped to a IntelliJ IDEA project, IntelliJ IDEA opens the read-only *page source*, just as if you chose **View Page Source** in the browser.

Navigating from a function to its call

To navigate from a function to the place in the code where it was called from:

- In the **Event Stack** pane, select the required item in the call stack and choose **Jump to Caller** on the context menu of the selection.

See Also

Reference:

- Spy-Js Tool Window

External Links:

- https://github.com/spy-js/spy-js#spy-js-documentation

Web Resources:

- Developer Community