By @blittle

Table of Contents

Chrome Developer Tools

OpenWest 2014

Bret Little - @little_bret

http://github.com/blittle

Basics

Basics

- Stable - Updated roughly every two-three weeks for minor releases, and every 6 weeks for major releases.
- Beta - Updated every week roughly, with major updates coming every six weeks, more than a month before the Stable channel will get them.
- Dev - Updated once or twice weekly.
- Canary - Nightly update

Chrome Release Channels

The Canary channel is not available for Linux. If you want a nightly build see the chromium project

Create multiple instances of chrome that have separate plugins enabled through User Profiles.

Internal Chrome Pages

A complete list of internal Chrome URLs is available at chrome://about/

Flags of interest

- `chrome://flags` - Experimental features that may cause instability in the browser.
    - `Enable Developer Tools experiments` - Useful for new additions to the developer tools.
    - `Enable Experimental JavaScript` - Enable ECMAScript 6 additions.
    - `Enable experimental Web Platform features` - Enable experimental features in Blink
- `chrome://inspect` - Device inspection

Navigation

Load the developer tools by pressing `F12`.

The developer tools are separated into eight tabs in addition to a settings dialog:

- Elements - Inspect and manipulate the DOM tree and associated styles and event listeners.
- Network - Monitor and examine network requests.
- Sources - Examine, edit, and debug source code.
- Timeline - Display and record activity as it runs, including events, script activity, page rendering, and memory usage.
- Profiles - Record CPU and memory profiles.
- Resources - Examine cookies, local storage, indexedDB, and web sql.
- Audits - Execute page performance and CSS usage audits.
- Console

Navigate between the separate tabs by pressing `ctr` + `[` and `ctr` + `]`

The Console is available at from any tab by pressing the `escape` key or `ctr` + `~`

More keyboard shortcuts

Settings

The settings dialog is available by pressing the cog icon in the top right corner of the dev tools or with the keyboard shortcut `?` □

Full settings documentation

Options to note

- Disable cache (while DevTools is open)
- Enable JavaScript and CSS source maps
- Experiments - Will only appear if enabled chrome://flags/#enable-devtools-experiments
    - Enable support for async stack traces
    - Enable frameworks debugging support
    - Show step-in candidates while debugging
    - Allow custom UI themes
    - Enable FlameChart mode in Timeline
- Workspace

Elements

Elements

DOM Tree

- As you hover over elements in the DOM tree, they will automatically highlight on the page.
- The path to selected element is displayed at the bottom of the window.
- Inspect a specific element with the ▫ or by right clicking an element on the page and click `Inspect Element`
- Options to edit the content of the DOM are available from the context menu (right click).
- Drag and drop DOM nodes.
- It can be very useful to copy the CSS or XPath to a selected element.
- Modification can be undone by `ctr` + `z`
- `ctl` + `alt` + `click` on a DOM node to auto expand all children of an element.

Styles

- The toggle state button allows you to toggle CSS pseudo anchor state.
- Clicking the source file name or `ctr` + `click` the property will load the associated source file.
- Properties that are ~~crossed out~~ are either overriden or unknown to Chrome.

Exercise

Inspect the following element and toggle its state.

State changes

Increment property values with the following shortcuts:

Increment Value	key-stroke
.1	`alt` + `up` & `alt` + `down`
1	`up` & `down`
10	`shift` + `up` & `shift` + `down`

LESS and SASS Source Maps

Chrome supports mapping the transpiled CSS rules back to the original SASS or LESS source. Make sure CSS source maps are enabled within the settings dialog.

Exercise

This Button is Styled With Less

DOM Breakpoints

DOM Breakpoints are useful to break on DOM mutation events, such as when a node is removed, modified, or its attributes are changed. You can view all current DOM breakpoints on the `DOM Breakpoint` tab.

Exercise

Find which script is modifying the following element by using DOM Breakpoints:

0

Event Listeners

Inspect event listeners associated with individual DOM nodes. Use the filter icon to restrict the event listeners to only the selected node.

Exercise

Find the onclick handler definition for the button below:

Alert Me

Computed Properties

Sometimes CSS properties are dynamically computed based upon the context of the style. For example an property value with an `em` unit is proportional to the parent element.

Exercise

Inspect the element below and analyze the defined versus computed properties:

Here is some calculated content

Network

Network

The network tab is very useful for inspecting network requests. Note: the developer tools need to be open for requests to be logged.

- The red button at the top left enables and disables the logging of requests (defaults on).
- The filter icon allows you to filter request by type and by key words.
- Mouse over individual requests on the timeline column to view timing information.
- The initiator column has a link to the file and line number from where the request was initiated.

Exercise

Use the network tab in the developer tools to find in the source where an AJAX request is made to `some-response.json`

Server Response:

WebSockets

Chrome developer tools contain a websockets view for analyzing websocket message frames.

- Green messages represent client to server (upload) messages while white represent server to client (download).
- The frames do not automatically update. Click on the request within the name column to refresh the frames.
- Right click to copy the data out of the frames.

Exercise

Inspect the frames sent within the following websocket connection:

Message: Send Message

DISCONNECTED

XHR Breakpoints

XHR Breapoints allow you to break whenever a server request URL contains a particular value. Add an XHR breakpoint within the Sources tab. Click the `+` icon and enter a value to break on.

Exercise

Use an XHR breakpoint to determine where in the code the following request orginates from:

Send Request

Copy Request Data

Right click individual requests to copy or save XHR data. Particularly useful, the `Copy as cURL` option will generate a `curl` query to reproduce the request. If the request type is `application/json` you can also "replay" the request.

Sources

Sources

One of the developer tools most powerful features is the sources tab. This section is essentially an integrated development environment which allows you to build, test, and debug web apps.

Navigate within the sources tab with the following shortcuts

key-stroke	Action
`ctrl` + `O`	Search scripts and stylesheets by filename*
`ctrl` + `F`	Search text within the current file
`ctrl` + `shift` + `F`	Search text accross all files
`ctrl` + `shift` + `O`	Navigate to a JavaScript function/method or CSS rule within the current file
`ctrl` + `shift` + `F`	Search text accross all files*
`ctrl` + `G`	Navigate to line number within current file
`alt` + `-` & `alt` + `+`	Navigate through source code editing locations - `+` is forward and `-` back
`ctrl` + `D`	Multi-select find next occurance
`ctrl` + `U`	Undo last multi-select occurance

* Available in all tabs (network, sources, elements, etc)

Pretty Print

Auto format minified JavaScript and CSS source with pretty print.

Editing Content

Chrome includes a smart editor live modifying application sources. The editor provides dynamic hints and code auto complete. Changes to CSS files are immediately applied whereas JavaScript modifications are applied with `ctrl` + `S`. If the browser successfully applies your code modifications, `Recompilation and update succeeded.` will be printed to the console.

Local Modifications

As you modify source files, an asterisk `*` appended to the filename represents a modified file. After applying the changes, an exclamation icon will in the tab header and the background changes to red to flag that the file has not been saved to disk. Save the file to disk by right clicking the background and selecting `Save As...`.

Chrome tracks the history and modifications to each file. Right click the the file and click `Local Modifications...` to inspect and revert file history events.

Workspaces

With workspaces you can grant Chrome permission to update your local file system with changes made within the sources editor. Do so by going into the developer tools settings and adding a local directory containing your web app. Confirm Chrome's access to the local file system. Right click a source file and click `Map to File System Resource...` and enter the corresponding local file resource. After restarting the developer tools, all file modifications that are saved (`ctrl` + `S`) will be saved to disk. Chrome will still track a history of revertable modifications.

Notice when the file gets saved and the browser is refreshed the changes persist. The `*` disappears from the file name and the background is not red. Also

Debugging

1. Resume script execution - `F8`
2. Step over to the nex function call - `F10`
3. Step into the next function call - `F11`
4. Step out of the current function scope - `shift` + `F11`
5. Deactivate all breakpoints.
6. Pause on exceptions - very useful for inspecting the context of application errors.
7. Watch Expressions - expressions that will be continuously evaluated as you step through the application.
8. Call stack - View and navigate to frames in the call stack. Can evaluate expressions and inspect the context of other frames.
9. Scope Variables - inspect variables within the current context.
10. Breakpoints - a list of all current breakpoints in the application. Enable/disable them through the checkbox.
11. Click a line to add or remove a breakpoint.
12. The currently executing line.

Restart Frame

When at a breakpoint inside debugger mode, you can conveniently restart the current stack frame by right clicking the call stack and selecting `Restart Frame`. This can be useful to inspect the current frame without reloading the page with a new breakpoint.

Long Resume

Resume with all pauses blocked for 500ms by long pressing on the resume button and selecting the second resume button in the menu. Useful to skip over many breakpoints.

In this demonstration we hit the first breakpoint twice. When we resume the second time we do not hit the second breakpoint.

Exercise

Open up the developer tools and refresh the page which should automatically break. Examine the difference between resuming and long-resuming the breakpoint.

Asynchronous Stack Traces

Asynchronous stack traces allow you to inspect function calls beyond the current event loop. This is particularly useful because you can examine the scope of previously executed frames that are no longer on the event loop. This feature is currently an experiment and needs to be enabled.

In the example above, notice the difference when asyncronous stack traces are enabled and disabled. Also note that the feature needs to be enabled **before** the breakpoint is reached.

More about asynchronous stack traces

Skipping Frameworks

Often times in debugging a web app, you do not want to step inside framework code, such as jQuery or lodash. Under the `Experiments` section of the settings, enable `Enable frameworks debugging support`. Once enabled, a new option in the settings should appear `Skip stepping through sources with particular names`. The field expects a regular expression file name.

Exercise

Open up the developer tools and refresh the page which should automatically break. Enable skipping frameworks support for `lodash`. In stepping into the lodash.reduce call, examine the difference when it is enabled and disabled.

Conditional Breakpoints

Conditional breakpoints allow you to break inside a code block when a defined expression evaluates to true. Conditional breakpoints highlight as orange instead of blue. Add a conditional breakpoint by right clicking a line number, selecting `Add Conditional Breakpoint`, and entering an expression.

Note: Because the conditional breakpoint simply evaluates an expression, you can add useful logging statements within the expression.

Uncaught Exceptions

Sometimes you may have a console error but have no idea where or how the exception was thrown. You can have the dev tools immediate break at uncaught or caught exceptions by clicking the □ icon. Use in conjunction with pretty-print in minfied apps.

Exercise

Reload the page. A error should be in the console. Use the uncaught exceptions to break at the error location and determine how to fix the problem.

Store as Global

The dev tools allow you to pick a scoped variable and automatically store a reference to it globally. This is convenient in debugging an application where you want to continuously inspect the contents of an object as a global variable.

Right click a `Scope Variable` and select `Store as Global Variable`. The console will immediately output the temporary name of the now globally available variable.

Note: A beta version of Chrome may be necessary for this feature.

Web Workers

HTML5 provides an API for spawning background scripts that can run concurrently with the primary JavaScript execution thread. Chrome developer tools provide a convenient way for debugging these background workers (threads). Each separate worker has its own dedicated inspector listed as a link. Clicking on one of the worker inspectors will boot a new instance of the chrome dev tools specific for that background thread. Optionally you can also have workers break automatically when they first boot.

Exercise

A worker has been started in the background that accepts two numbers, adds them, and returns the result. Pass data to the worker thread and use the chrome developer tools to debug the output.

+ = Calculate

Source Maps

The Chrome dev tools support source maps, which allow you to debug transpiled JavaScript code as their original source language. This may include TypeScript, CoffeeScript, ClojureScript, or ECMAScript 6. Sourcemaps are especially useful because you can place breakpoints, step through, and debug the originally authored source. Make sure that you enable JavaScript source maps by checking the option within the settings:

Exercise

Enable sourcemaps in the developer tool settings. Close and reload the developer tools. Open up `ctrl` + `O` `example.ts` and `example.coffee`. Explore these files by reloading the page with break points placed in the files.

Snippets

Chrome provides a "snippets" bucket within the developer tools which allows you to manage small (or large) scripts that can be injected onto a page. Scripts can do things like inject jQuery into a page, log all globally defined variables, or print out all colors from computed styles used in elements on the page.

Useful JavaScript development snippets

Timeline

Timeline

Exersize

Each button below will append 4000 rows to a table. The first is almost 10 times slower than the second. Use the timeline to help determine where what causes the performance hit.

Generate Elements 1

Add Elements

Generate Elements 2

Add Elements

Profiles

Profiles

Exersize

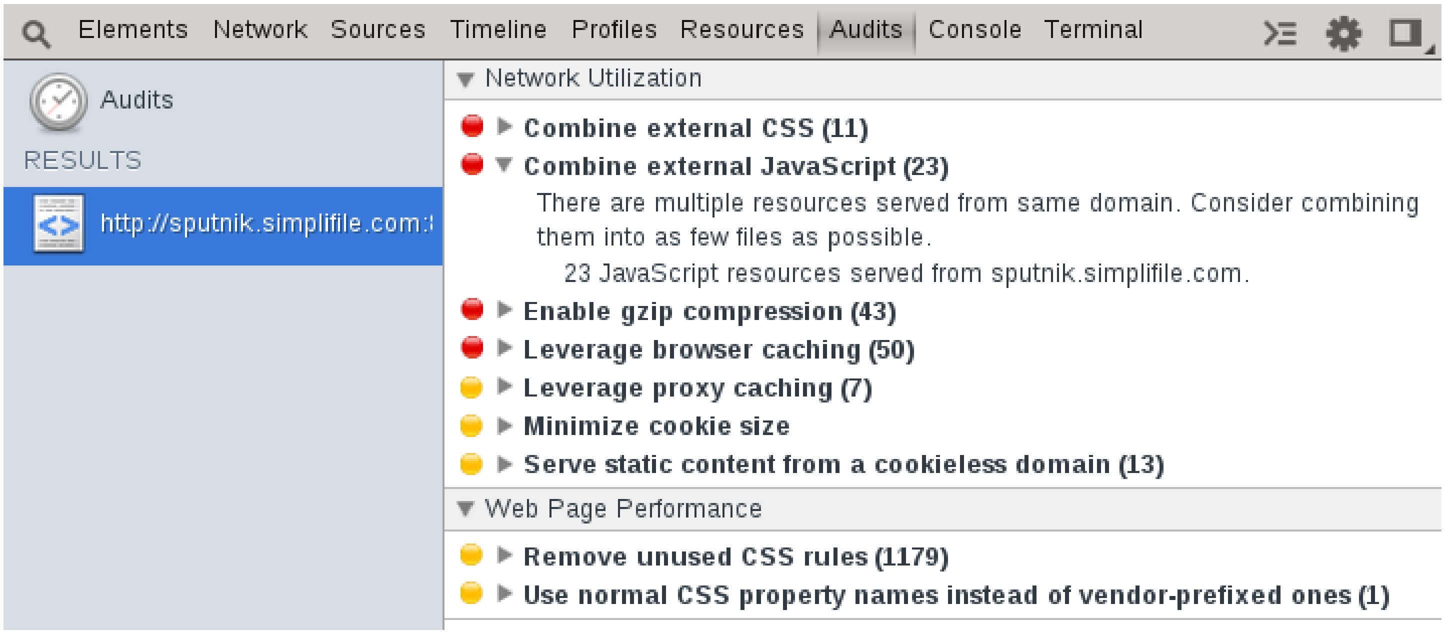
Each time you generate rows is pressed, the table will be cleared and 500 rows will be placed within it. Pressing it multiple times generates memory leaks. Use the profile tool to determine the location of the leak.

Generate Rows

Audits

Audits

This tab allows you to run network utilization and performance audits on your web app. The audit results are automatically sorted based upon severity, red being the most severe and yellow the least.



Network Utilization

- Combine external CSS (11)
- Combine external JavaScript (23)
  There are multiple resources served from same domain. Consider combining them into as few files as possible.
  23 JavaScript resources served from sputnik.simplifile.com.
- Enable gzip compression (43)
- Leverage browser caching (50)
- Leverage proxy caching (7)
- Minimize cookie size
- Serve static content from a cookieless domain (13)

Web Page Performance

- Remove unused CSS rules (1179)
- Use normal CSS property names instead of vendor-prefixed ones (1)

Console

Console

In addition to providing a convenient workspace for testing code, the console also provides a chrome specific API for inspecting and debugging applications. It is important to remember that the console is available from all other tabs by pressing the `Esc` key.

1. Clear the console output
2. Filter the console to display specific output
3. Change the console input to be executed with the context of a specific frame (frame, iframe, or extension).
4. Enter multi-line code with `shift` + `enter`
5. The shortcut `ctrl` + `L` clears the console

Exercise

Load the console and notice at the top frame the global variable `iFrameApp` is not defined. Switch the frame to iframe.html and it is defined.

Query DOM

The console provides an API for querying the DOM with CSS selectors or with XPath. Querying with CSS selectors is the same API as jQuery, and will default to jQuery if jQuery is loaded. The XPath API is similar except with `$x('my/xpath')`

When ever a DOM element is output within the console, you can right click the element and select `Reveal in Elements Panel` to reveal its location. Remember, within the elements panel, we can right click any element and copy its XPath or CSS location.

As you select elements within the elements pane, chrome retains a history of objects selected. From the console, you can reference previously selected elements with `$0, $1, $2, $3, $4` with `$0` being the most recent.

Monitor Events

From the console you can usefully monitor events with the `monitorEvents` API. The API takes an object to be monitored and then an event to listen for. For example:

```
monitorEvents(document, ["scroll"]);
monitorEvents($('#action-button'), ["mousedown", "mouseup"]);
```

To stop monitoring events, simply call `unmonitorEvents()` passing the object to stop monitoring on.

```
unmonitorEvents(document);
```

Commands

The console provides the following useful commands:

- `assert` - A simple assertion API to verify functionality
- `time` and `timeEnd` - Measure how long something takes
- `timeStamp` - Programatically mark the timeline.
- `debugger` - Force the debugger to break
- `profile` and `profileEnd` - Programatically start and stop a CPU profile

```
console.assert($('#my-component').length, "My component exists on the page");

console.time("ajax request");
$.get('/some/url', function() {
   // Will print to the screen the
   // elapsed time between start and end
   console.timeEnd("ajax request");
});

function removeCalendar() {
   console.timeStamp("removing calendar");
   calendarElement.remove();
   calendarDisplayed = false;
}
```

Mobile

Mobile

The primary mobile functionality provided by the developer tools is through device, screen, user-agent, and sensor emulation. The emulation functionality is somewhat hidden within the console slideout available from every pane when you press `Esc`.

You can either manually modify the screen resolution, pixel ratio, and user agent, or you can pick from pre-existing devices including: Galaxy s4, Nexus 5, iPhone 5, iPad 4, and other devices.

You can also see how your app behaves based upon the accelerometer and geolocation.

USB Debugging

Remotely debugging allows you the full chrome developer tools enabled within the environment of your remote device. You can remotely debug mobile devices by navigating to chrome://inspect.

Plug your mobile device into your development machine. Check the checkbox `Discover USB devices`. You may need to authorize the connection on your mobile device. Once authorized, you should see a list of all available inspection targets on your device. Click `inspect` to launch an instance of the developer tools. Within this instance of the dev tools, all functionality should be available.

If your device and version of chrome support it, a new icon should appear in the very top right of the developer tools. Press this button to enable a screencast from your mobile device to your development machine.

Extensions

Extensions

- Live Reload -
- GruntJS
- HTML5 Terminal
- BrowserSync
- IDEs
    - Sublime Inspector
    - Jetbrains Inspector

Custom Skins

Themes

Newer versions of chrome officially support custom skins for the developer tools by simply installing chrome extensions. Older versions can still have custom themes, but requires manually copy and pasting a stylesheet within your chrome application data directory.

http://devthemez.com/themes/chrome-developer-tools lists a variety of skins available.

My particular favorite Zero Dark Matrix