

JavaScript Tools: Chrome Developer Tools

Sang Shin

JPassion.com

“Learn with Passion!”



Acknowledgment

- Some slides of this presentation are created from the contents of Google Developers Website, which are available with Creative Commons Attribution 3.0 License

Topics

- Authoring and development workflow
- Debugging JavaScript
- Using the console

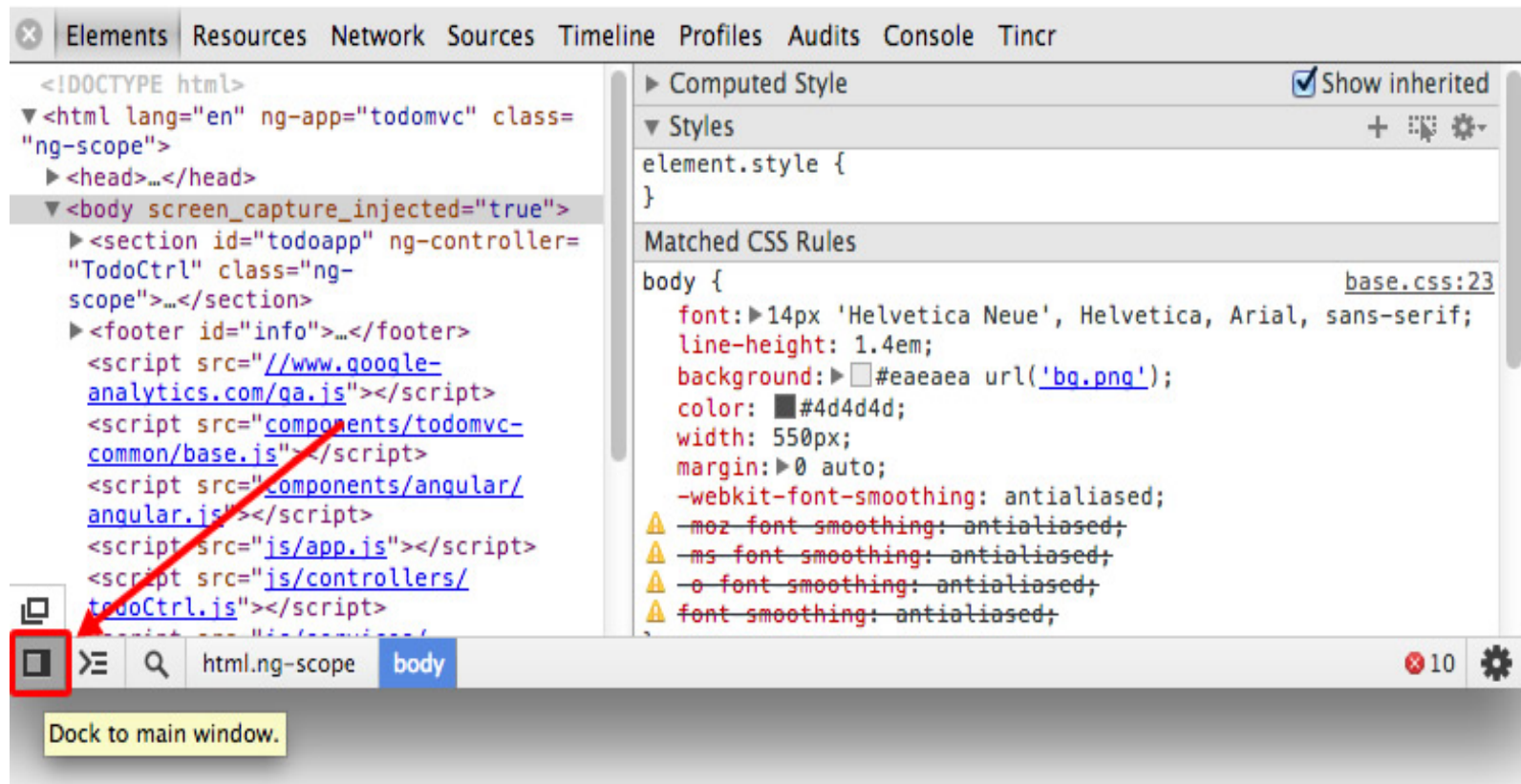
Authoring & Development Workflow

Authoring Tasks

- Docking
- Search, navigate and filter
- Live editing scripts & styles
- Custom JavaScript snippets
- Persistence extensions

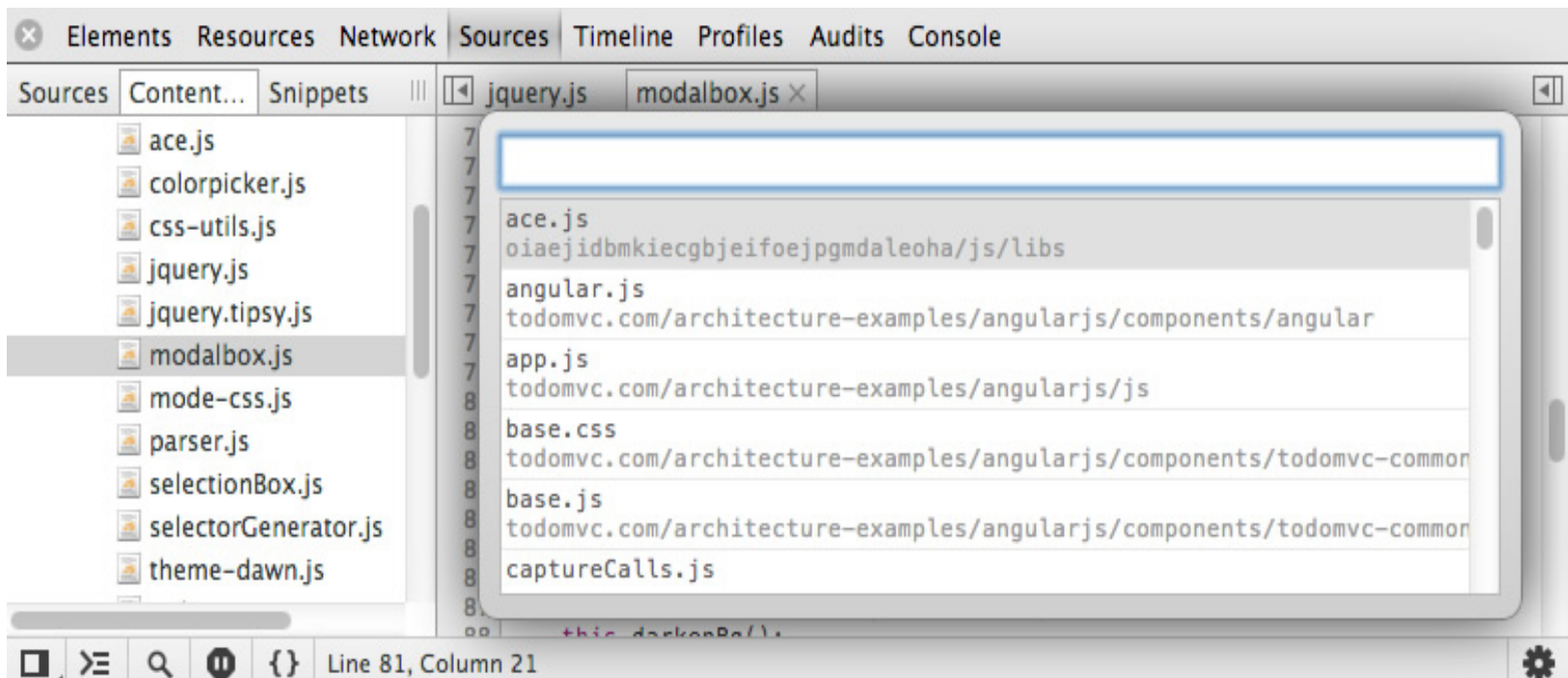
Docking

- Horizontal, vertical, separate window



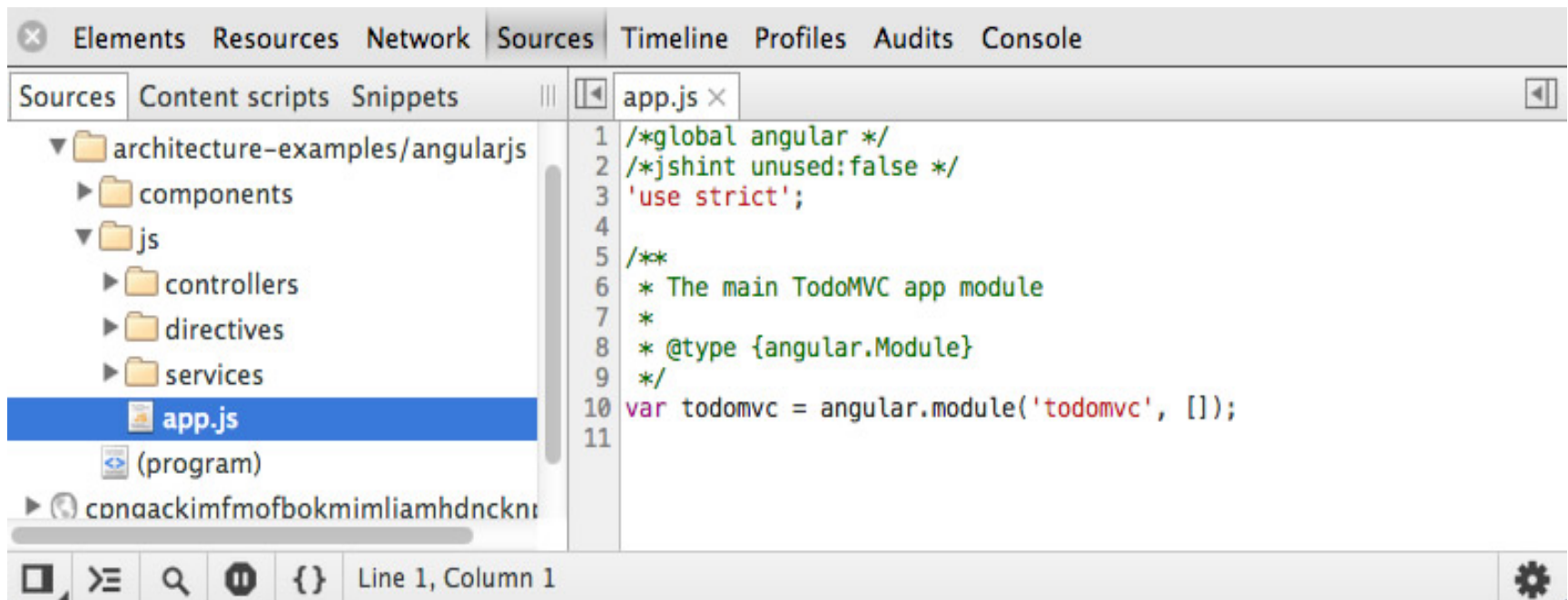
Search, Navigate and Filter

- The DevTools allow you to search across all script, stylesheet and snippet files



Live Editing Scripts & Styles

- The DevTools support editing both styles and scripts live, without the need for a full page refresh. This helps when testing design changes, prototyping JavaScript functions or snippets



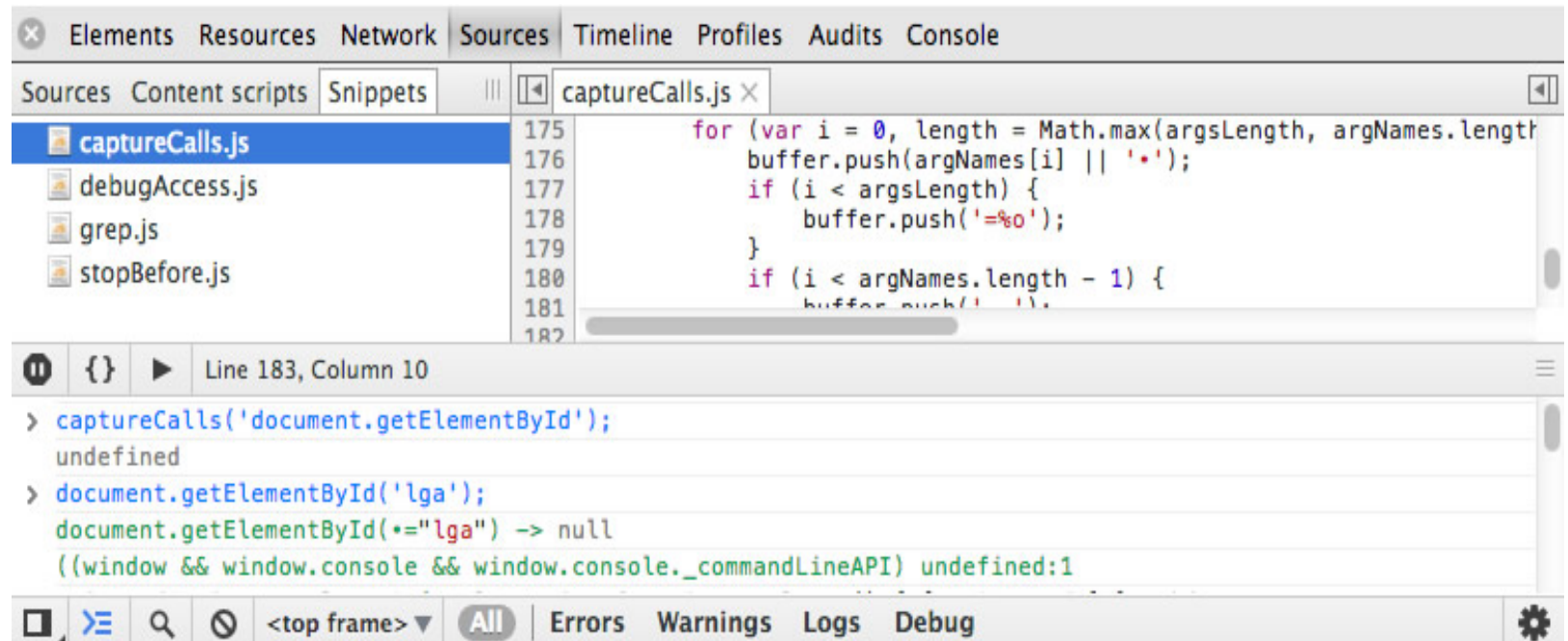
The screenshot shows the Chrome DevTools interface. The 'Sources' panel is open, displaying a file tree on the left and the code editor on the right. The file tree shows a folder structure: 'architecture-examples/angularjs' containing 'components', 'js', 'controllers', 'directives', and 'services'. The 'app.js' file is selected and highlighted in blue. The code editor shows the following JavaScript code:

```
1 /*global angular */
2 /*jshint unused:false */
3 'use strict';
4
5 /**
6  * The main TodoMVC app module
7  *
8  * @type {angular.Module}
9  */
10 var todomvc = angular.module('todomvc', []);
11
```

The status bar at the bottom indicates 'Line 1, Column 1'.

Custom JavaScript Snippets

- Sometimes you want to be able to save smaller scripts, bookmarklets and utilities so that you've always got them available to you while debugging in the browser



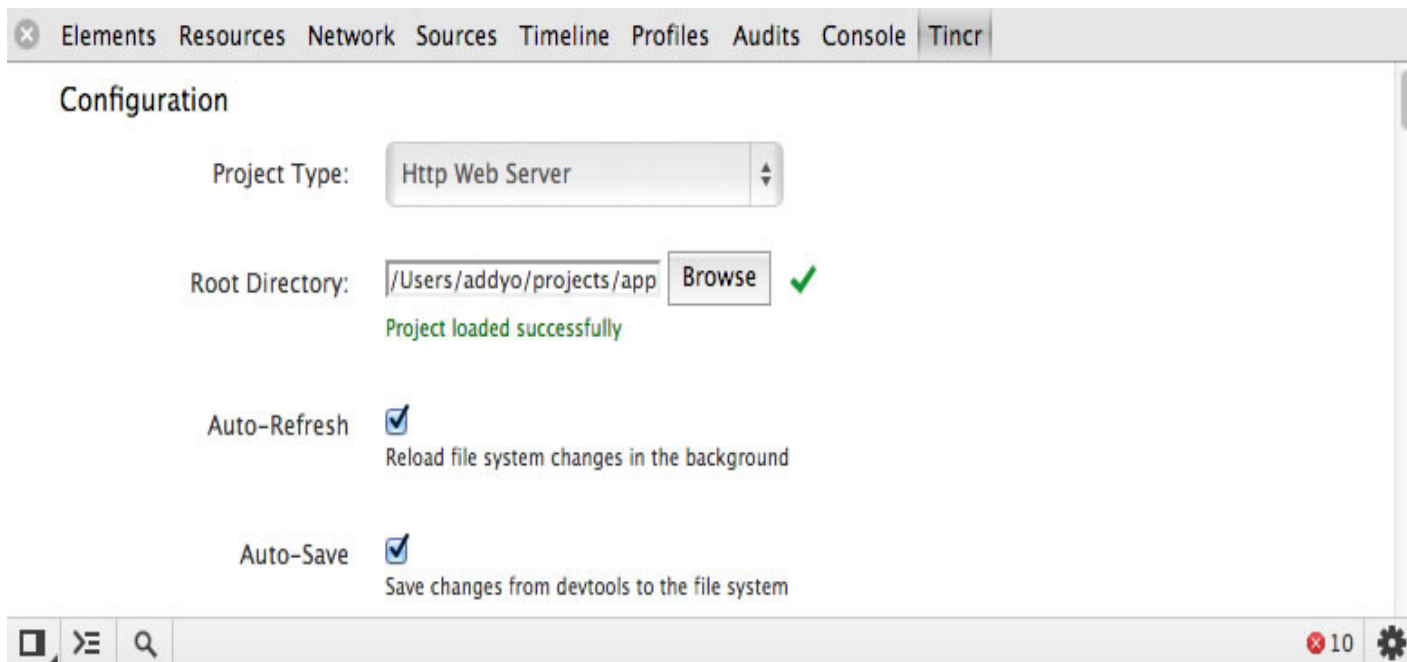
The screenshot displays the Chrome DevTools interface. The 'Sources' panel is open to the 'Snippets' section, showing a list of custom JavaScript files: `captureCalls.js`, `debugAccess.js`, `grep.js`, and `stopBefore.js`. The `captureCalls.js` file is selected, and its code is visible in the editor. The code is a function that iterates over arguments and their names, pushing them into a buffer. The console shows the execution of this function with various arguments, including `document.getElementById`, `'lga'`, and a regular expression `document.getElementById(.*="lga")`, resulting in `null`. The console also shows the `console._commandLineAPI` object.

```
for (var i = 0, length = Math.max(argsLength, argNames.length)
buffer.push(argNames[i] || '.');
if (i < argsLength) {
    buffer.push('=%o');
}
if (i < argNames.length - 1) {
    buffer.push(' ');
}
```

```
> captureCalls('document.getElementById');
undefined
> document.getElementById('lga');
document.getElementById(.*="lga") -> null
((window && window.console && window.console._commandLineAPI) undefined:1
```

Persistence Extensions

- You can make changes inside the Tools (to scripts and styles) which are then automatically saved to your source files. Similarly, you can make changes to your source files (CSS/JavaScript) which result in a browser reload showing your changes.



Lab:

Exercise 1: Authoring and Development Workflow

4254_javascript_tools_chrome.zip

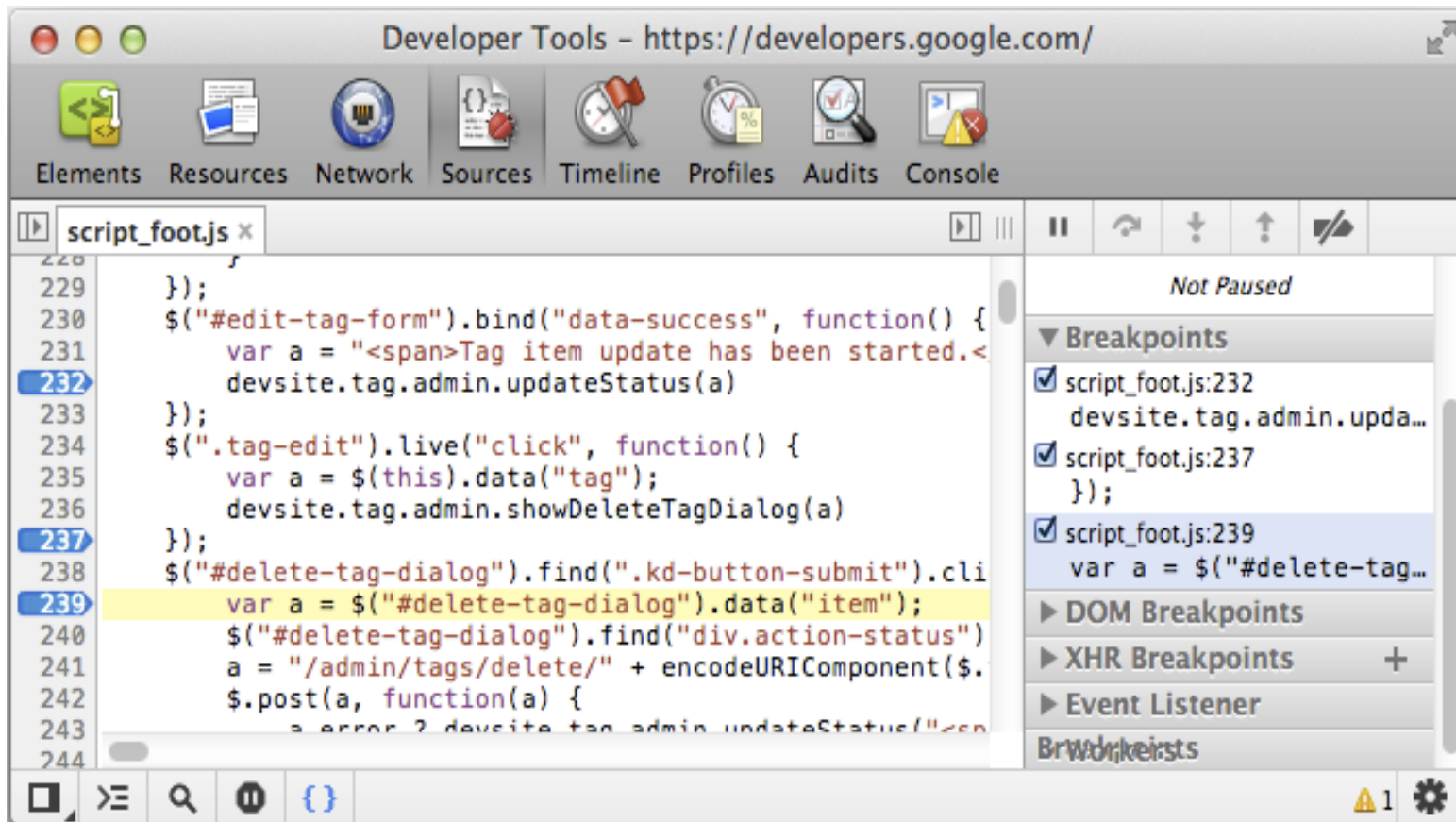


Debugging JavaScript

Debugging with Breakpoints

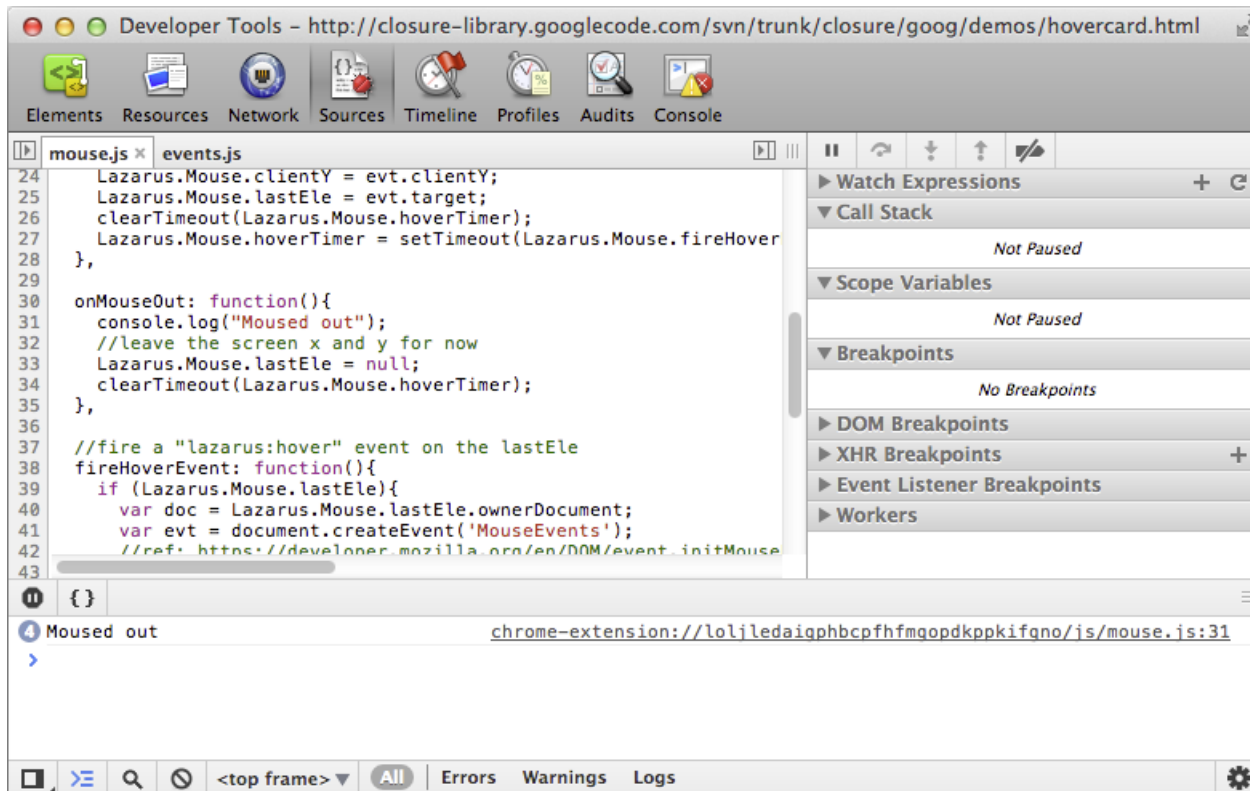
- Source Panel
- Debugging with Breakpoints
- Live editing
- Handling exceptions
- Pretty print
- Working with Source Maps

Debugging With Breakpoints



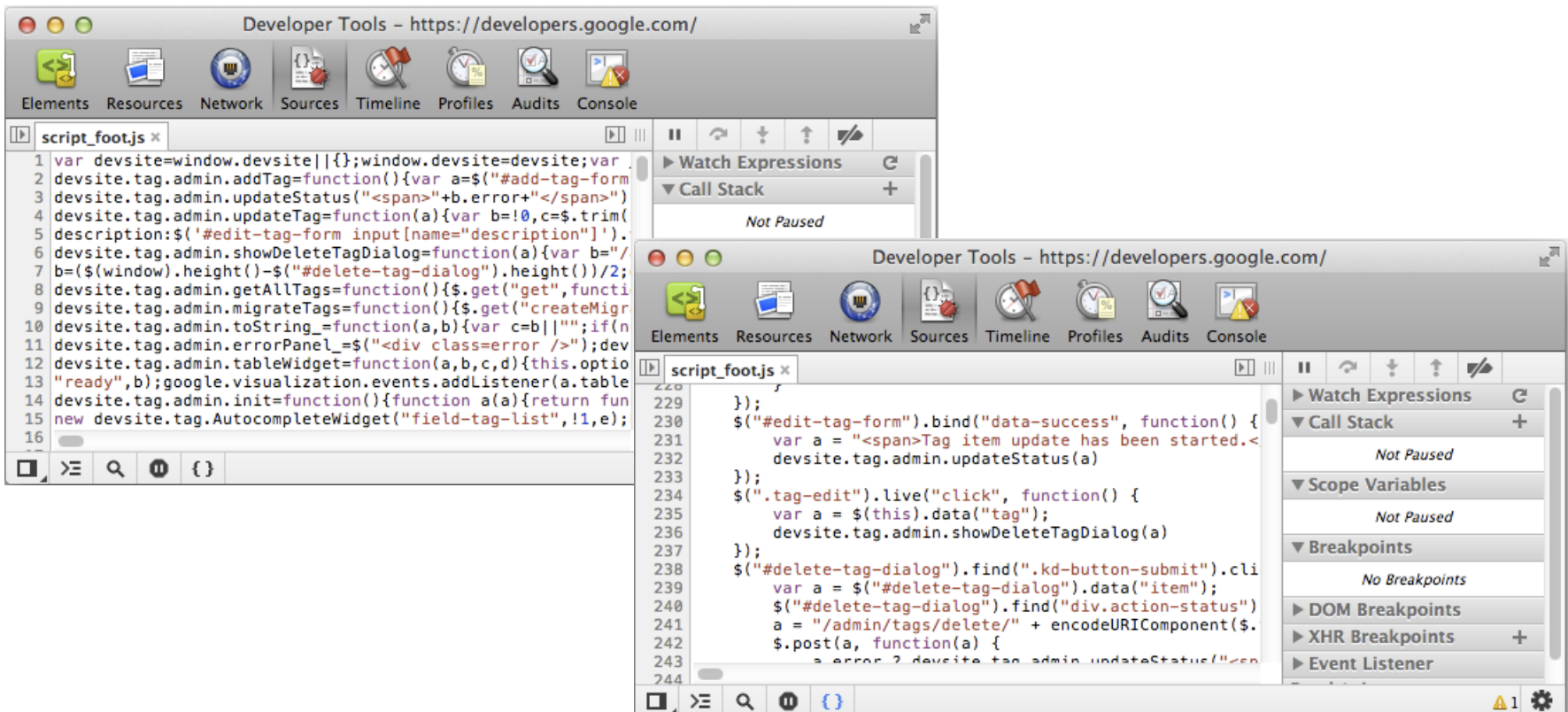
Live Editing at a Breakpoint

- While at a breakpoint, it's also possible to live edit scripts



Pretty Print

- JavaScript is transformed into a more human readable form



Source Maps

- Motivation
 - > Have you ever found yourself wishing you could keep your client-side code readable and more importantly debuggable even after you've combined and minified it, without impacting performance?
- What is it?
 - > Source Maps are a generic mapping format (that are JSON-based) which can be used by any processed file to create relations between files that are pre-processed and those that are post-processed
 - > Of most relevance to us is that they can be used to map combined/minified scripts back to an unbuilt state for debugging.

Lab:

Exercise 1: Debugging JavaScript
4254_javascript_tools_chrome.zip



Using Console

Using Console API

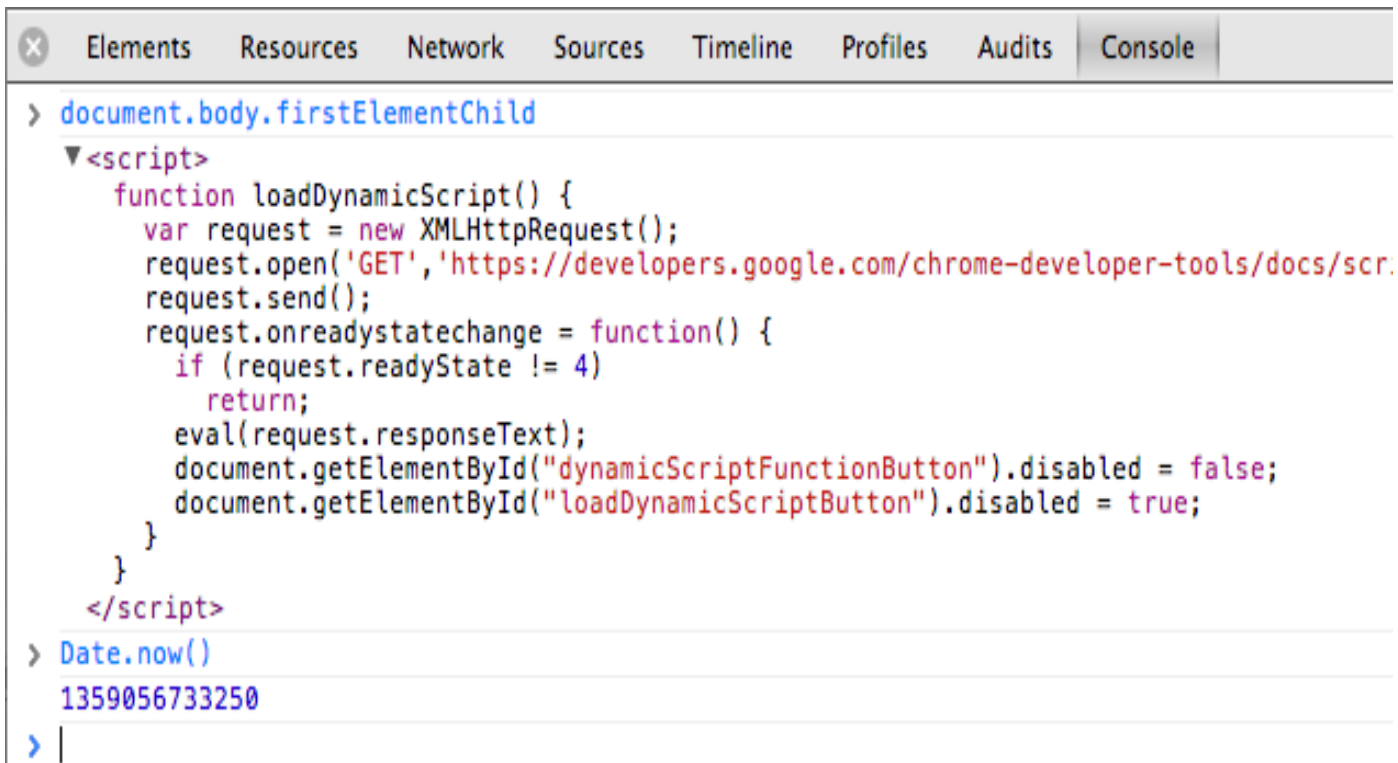
- `console.log()`
- `console.error()`
- `console.assert()`

Using Command Line API

- Console is also a shell prompt where you can directly evaluate expressions or issue commands provided by the Command Line API
 - > Convenience functions for selecting DOM elements
 - > Methods for controlling the CPU profiler
 - > Aliases for a number of Console API methods
 - > Monitoring events
 - > View event listeners registered on objects

Evaluating expressions

- The Console attempts to evaluate any JavaScript expression you enter at the shell prompt, upon pressing the Return or Enter key

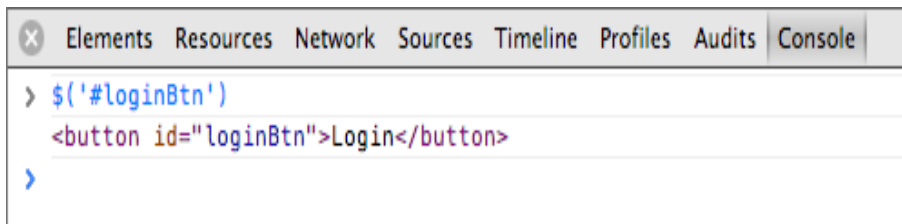


The screenshot shows the Chrome DevTools Console with the 'Console' tab selected. The console displays the following content:

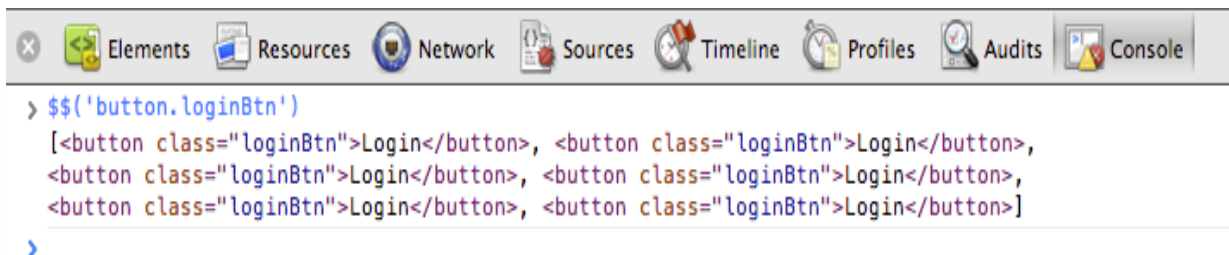
```
> document.body.firstChild
▼ <script>
  function loadDynamicScript() {
    var request = new XMLHttpRequest();
    request.open('GET', 'https://developers.google.com/chrome-developer-tools/docs/scr');
    request.send();
    request.onreadystatechange = function() {
      if (request.readyState != 4)
        return;
      eval(request.responseText);
      document.getElementById("dynamicScriptFunctionButton").disabled = false;
      document.getElementById("loadDynamicScriptButton").disabled = true;
    }
  }
</script>
> Date.now()
1359056733250
> |
```

Selecting Elements

- The Command Line API provides several methods to access DOM elements in your application. For example, the `$()` method returns the first element that matches the specified CSS selector



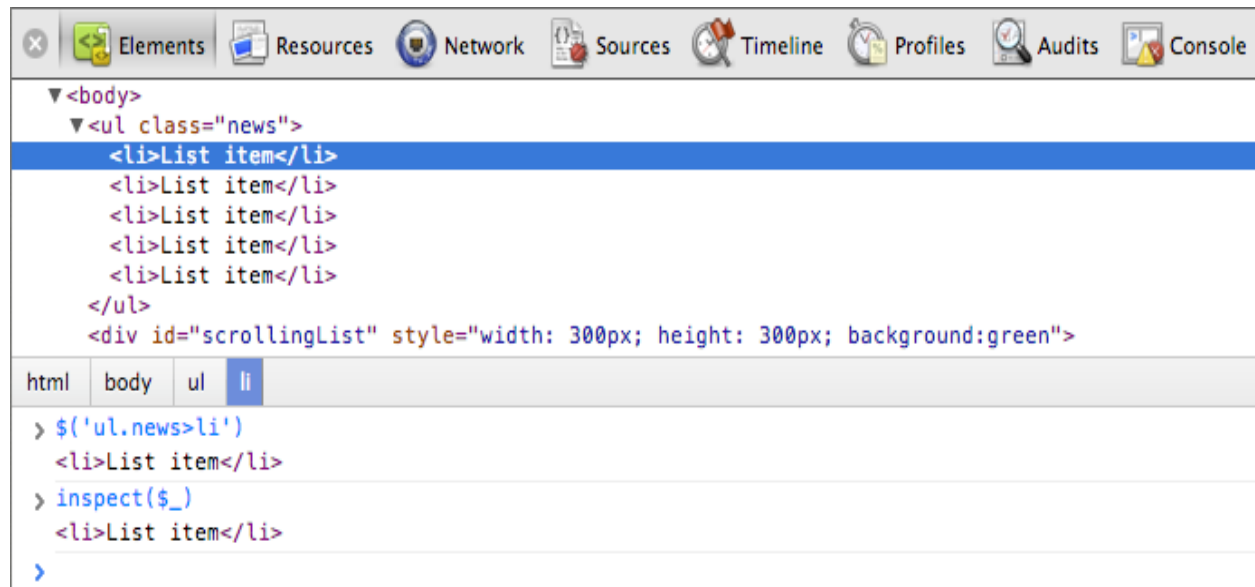
```
Elements Resources Network Sources Timeline Profiles Audits Console
> $('#loginBtn')
<button id="loginBtn">Login</button>
>
```



```
Elements Resources Network Sources Timeline Profiles Audits Console
> $$('button.loginBtn')
[<button class="loginBtn">Login</button>, <button class="loginBtn">Login</button>,
<button class="loginBtn">Login</button>, <button class="loginBtn">Login</button>,
<button class="loginBtn">Login</button>, <button class="loginBtn">Login</button>]
>
```


Inspecting DOM Elements & JavaScript Heap Objects

- The `inspect()` method takes a DOM element reference (or JavaScript reference) as a parameter and displays the element or object in the appropriate panel—the Elements panel for DOM elements, or the Profile panel for a JavaScript object.



```
Elements Resources Network Sources Timeline Profiles Audits Console
▼ <body>
  ▼ <ul class="news">
    <li>List item</li>
    <li>List item</li>
    <li>List item</li>
    <li>List item</li>
    <li>List item</li>
  </ul>
  <div id="scrollingList" style="width: 300px; height: 300px; background:green">
html body ul li
> $('ul.news>li')
  <li>List item</li>
> inspect($_)
  <li>List item</li>
>
```

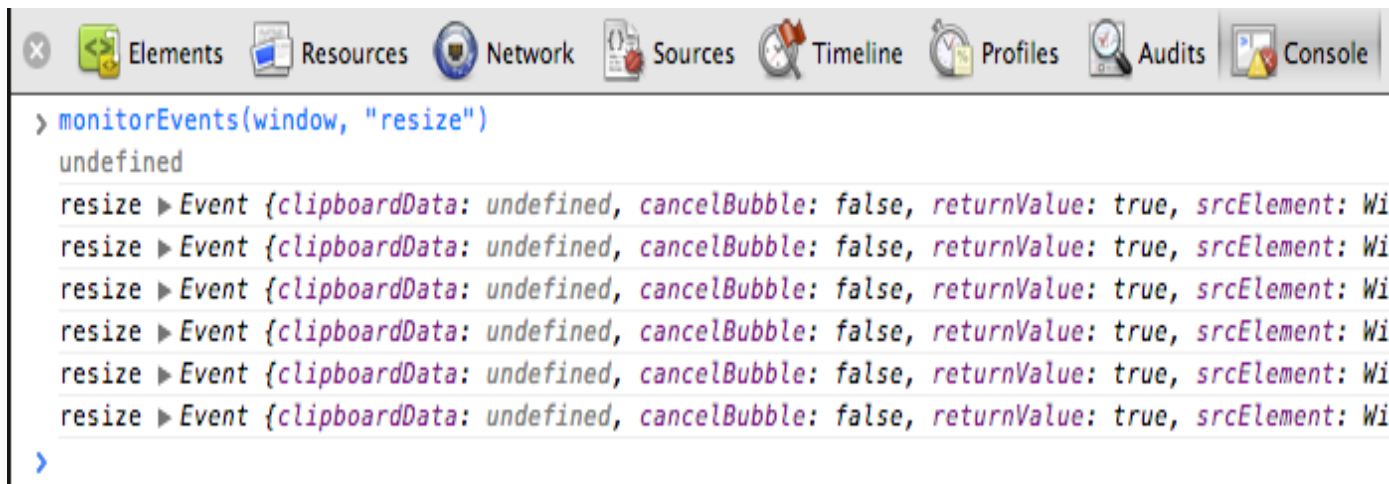
Accessing Recently Selected Elements & Objects

- The Console remembers the last five elements (or heap objects) you've selected and makes them available as properties named \$0, \$1, \$2, \$3 and \$4

```
html#vbulletin_html  body  div#everything  div.body_wrapper  div#tmxmenu
> $0
▶ <div id="tmxmenu">...</div>
> $1
▶ <div class="bread_and_butter">...</div>
> $2
▶ <div id="postlist" class="postlist restrain">...</div>
```

Monitoring Events

- The `monitorEvents()` command monitors an object for one or more specified events. When an event occurs on the monitored object, the corresponding Event object is logged to the Console.



The screenshot shows the Chrome DevTools Console with the following content:

```
> monitorEvents(window, "resize")
undefined
resize ▶ Event {clipboardData: undefined, cancelBubble: false, returnValue: true, srcElement: Wi
resize ▶ Event {clipboardData: undefined, cancelBubble: false, returnValue: true, srcElement: Wi
resize ▶ Event {clipboardData: undefined, cancelBubble: false, returnValue: true, srcElement: Wi
resize ▶ Event {clipboardData: undefined, cancelBubble: false, returnValue: true, srcElement: Wi
resize ▶ Event {clipboardData: undefined, cancelBubble: false, returnValue: true, srcElement: Wi
resize ▶ Event {clipboardData: undefined, cancelBubble: false, returnValue: true, srcElement: Wi
>
```

Lab:

**Exercise 3: Debugger
4253_javascript_tools.zip**



Learn with Passion!
JPassion.com

