109 學年度 第 1 學期

JavaScript程式設計

蘇俊銘(Jun-Ming Su)
jmsu@mail.nutn.edu.tw

星期五，節次8、9、A (J304)

數位學習科技學系, 國立臺南大學

2020

工欲善其事，必先利其[器]

工欲善其事，必先利其器

- 開發環境規劃:
  - 如何在 **Windows** 打造 **JavaScript Web** 開發環境
  - **Ref:**
  - **https://www.happycoder.org/2017/12/19/javascript101-windows-dev-environment-setup-tutorial/**

- 安裝軟體步驟:
  - 下載安裝 **google chrome** 瀏覽器
  - 下載安裝 **Visual Studio Code** 或 **Sublime text** 文字編輯器
  - 下載安裝 **cmder terminal** 終端機程式 **(**請下載含 **git** 的 **full** 完整版本**)**
  - 下載安裝 **Node.JS** 選擇左邊穩定版本，按照指令安裝完成
  - 在終端機輸入 **node -v** 若成功顯示版本，代表安裝完成
  - 安裝 **http-server** 套件：**npm install http-server -g**

工欲善其事，必先利其器

- 為什麼我從 **Sublime Text** 跳槽 **Visual Studio Code**？
  - **Ref:**
  - **https://medium.com/hungys-blog/why-i-switched-from-sublime-to-vscode-ea030b3ff1d9**

npm (Node Package Manager)

- **node包管理器:**
  - 是Node.js預設的、
  - 以JavaScript編寫的軟體套件管理系統
  - Wiki:
    - https://zh.wikipedia.org/wiki/Npm

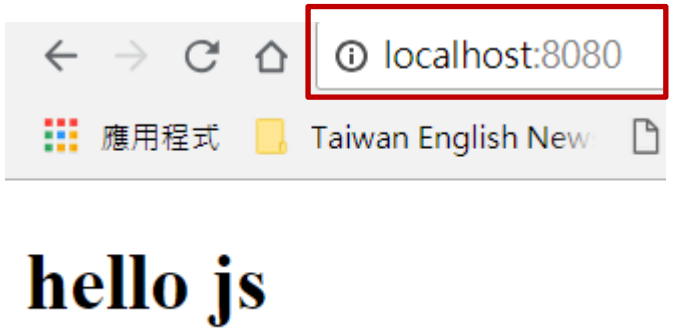
撰寫第一個 JavaScript 程式

```
<!DOCTYPE html>
<html>
    <head>
    <meta charset="utf-8">
    <meta name="viewport" content="width=device-width">
    <title>Test</title>
    <script type="text/javascript">
        console.log('hello js')
    </script>

    </head>
<body>
    <h1> hello js </h1>
</body>
</html
```

啟動伺服器

- 終端機移動到該檔案資料夾下
  - http-server -p 8080
  - 指令 –p(設定port) port編號
- Cmder

```
D:\Works\KALE Lab\Course\Web Programming\2018-08\temp\JS_project
λ http-server -p 8080
Starting up http-server, serving ./
Available on:
  http://192.168.43.55:8080
  http://127.0.0.1:8080
```



啟動伺服器

- Windows預設 cmd環境亦可:

命令提示字元 - http-server -p 9099

```
Microsoft Windows [版本 6.3.9600]
(c) 2013 Microsoft Corporation. 著作權所有，並保留一切權利。

C:\Users\user>cd D:\Works\KALE Lab\Course\Web Programming\2018-08\temp\JS_projec
t

C:\Users\user>d:

D:\Works\KALE Lab\Course\Web Programming\2018-08\temp\JS_project>http-server -p
9099
Starting up http-server, serving ./
Available on:
  http://192.168.43.55:9099
  http://127.0.0.1:9099
Hit CTRL-C to stop the server
```

環境安裝練習

- 依照安裝步驟建置開發環境
- 正確在Chrome中出現以下畫面:

工欲善其事，必先利其[技]

了解JavaScript Debug技巧

For browser-based JavaScript debugging,

Not for Node.js

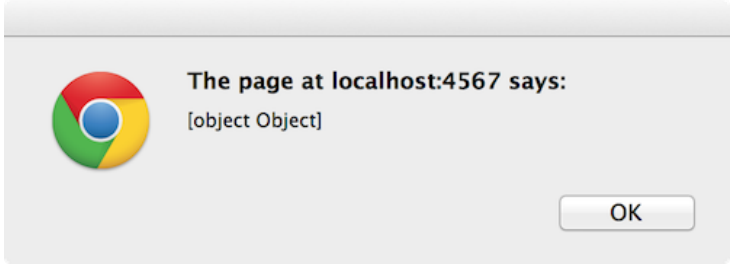
http://juliepagano.com/blog/2014/05/18/javascript-debugging-for-beginners/

Alert

- It displays a dialog with the specified string value and an “ok” button to dismiss it.
- It will stop the JavaScript from continuing until you click “ok”.
- alert("Hello! I am an alert.");

Alert

- This is useful for debugging because you can set the alert value to something meaningful.
- *// I want to know if I reach this part of the code.*
- alert("I am here!");
- *// I want to know the value of foo in this part of the code.*
- alert("Foo: " + foo);
- *// I heard you like alerts...*
- **for** (i = 0; i < 100; i++) { alert(i); }
- the alert is a very limited tool and It can only display **strings**.
- *// I want to see all the H2s on the page.*
- alert($('h2'));



Chrome Developer Tools

- are invaluable for JavaScript debugging
- open it via keyboard shortcuts or F12:

Console

- It is an incredibly useful part of JavaScript development.
- In many programming languages, you do debugging and logging in the terminal.
- The console is the equivalent for JavaScript in the browser.

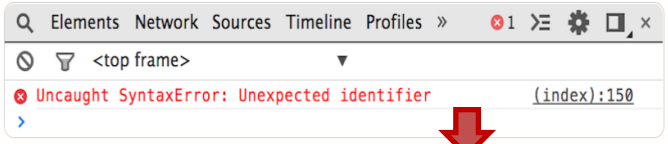
Console-error

Console

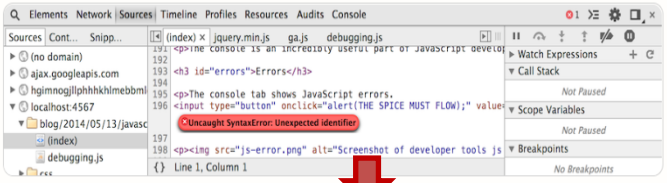
The console is an incredibly useful part of JavaScript development. In many programming languages, you do debugging and logging in the terminal. The console is the equivalent for JavaScript in the browser.

Errors

The console tab shows JavaScript errors. Click to create an error



It also shows you where the error originated. Click the underlined file name and line count to the right of the error to be taken to the origin.



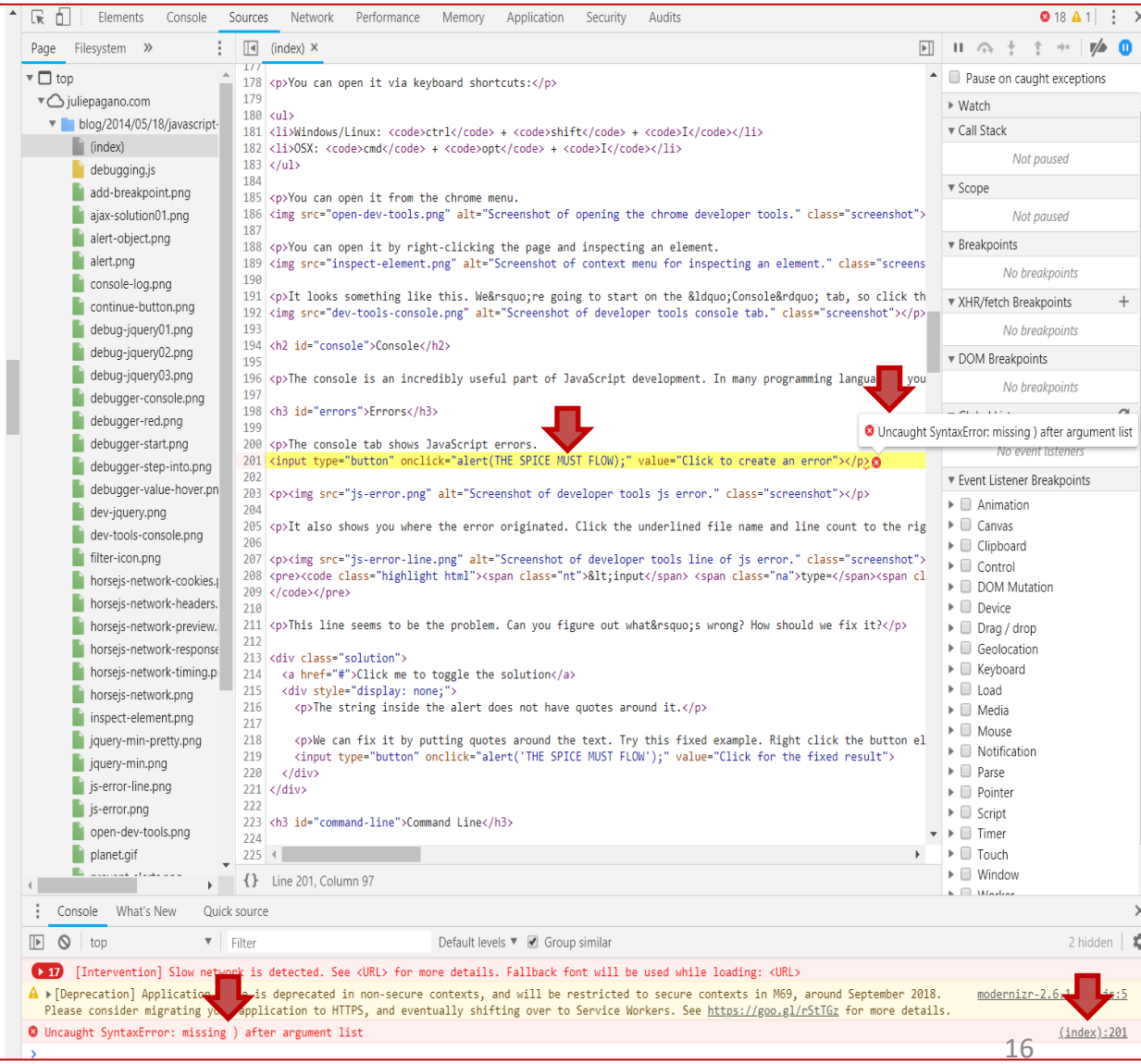

```
<input type="button" onclick="alert(THE SPICE MUST FLOW);"
value="Click to create an error">
```

This line seems to be the problem. Can you figure out what's wrong? How should we fix it?

Click me to toggle the solution

Command Line

The console has an interactive command line that can be useful for debugging.



Command Line

- The console has an interactive command line:
  - can be useful for debugging
- basic JavaScript:
  - *// Try some math.*
  - 2 + 2
  - *// How about some string manipulation*
  - "the golden " + "path"
  - *// You can even create an alert* alert("Muad'Dib!");
- complicated JavaScript:
  - *// You can create variables.*
  - **var** arr = [1, 2, 3];
  - *// You can add multiple lines by pressing shift + enter.*
  - **for** (**var** i = 0; i < arr.length; i++) { arr[i] = arr[i] * 2; }
  - arr;

```
▼(3) [2, 4, 6]
    0: 2
    1: 4
    2: 6
    length: 3
  ▶ __proto__: Array(0)
```

Command Line

- provides a bunch of helper functions:
  - *// You can look up elements via* *css selectors* $$('h2');
  - *// Or*
  - *xpath* $x('//h2');
- command line api reference:
  - contains a collection of convenience functions for performing common tasks:
    - selecting and inspecting DOM elements,
    - displaying data in readable format,
    - stopping and starting the profiler,
    - monitoring DOM events.

command line api reference

- $_
  - returns the value of the most recently evaluated expression.
  - 2+2
  - [“A”,”B”,”C”]
  - $_

```
["A","B","C"]
▶ (3) ["A", "B", "C"]
$_.length
3
```

- **$0, $1, $2, $3, $4:**
  - a historical reference to the last five DOM elements inspected within the Elements panel
  - or the last five JavaScript heap objects selected in the Profiles panel.
  - **$0:** returns the most recently selected element or JavaScript object,
  - **$1:** returns the second most recently selected one,
  - **$2, $3, $4:** and so on.

Test by: http://juliepagano.com/blog/2014/05/18/javascript-debugging-for-beginners/

```
> $0
  <p>The string inside the alert does not
> $1
  <input type="button" onclick="alert(THE
> $2
  <input type="button" onclick="alert('Hello! I am an alert.');" value="Click for an alert">
```

Elements onsole Sources Network Performance

```
<!DOCTYPE html>
<html lang="en" data-react-helmet="lang">
▶<head>…</head>
…▶<body class="pastel" data-react-helmet="class">…</body> == $0
</html>
```

$(selector, [startNode])

- $(selector):
  - returns the reference to the first DOM element with the specified CSS selector.
  - is an alias for the document.querySelector().
- Test by: https://www.w3schools.com/css/css_website_layout.asp
- $('img') :

```
$('img')
<img src="https://www.google.com/images/cleardot.gif" class="goog-te-gadget-icon" alt style="background-image: url("https://translate.googleapis.com/translate_static/img/
te_ctrl3.gif"); background-position: -65px 0px;">
```

- $('img').src :

```
$('img').src
"https://www.google.com/images/cleardot.gif"
```

- $('img', document.querySelector('background-image')).src :

```
> $('img', document.querySelector('background-image')).src
<· "https://www.google.com/images/cleardot.gif"
```

$$(selector, [startNode])

- $$(selector):
  - returns an **array of elements** that match the given **CSS selector**.
  - is equivalent to calling document.querySelectorAll().
  - var images = $$('img');
  - for (each in images) {
  - console.log(images[each].src);}

```
> var images = $$('img');
    for (each in images) {
        console.log(images[each].src);
    }
3 https://www.google.com/images/cleardot.gif
  https://www.w3schools.com/images/colorpicker.gif
  https://www.w3schools.com/images/w3schoolscom_gray.gif
  https://www.gstatic.com/images/branding/product/1x/translate_24dp.png
```

  - var images = $$('img', document.querySelector('background-image'));
  - for (each in images) {
  - console.log(images[each].src); }

```
> var images = $$('img', document.querySelector('background-image'));
    for (each in images) {
        console.log(images[each].src);
    }
3 https://www.google.com/images/cleardot.gif
  https://www.w3schools.com/images/colorpicker.gif
  https://www.w3schools.com/images/w3schoolscom_gray.gif
  https://www.gstatic.com/images/branding/product/1x/translate_24dp.png
```

$x(path, [startNode])

- $x(path):
  - returns an **array of DOM elements** that match the given XPath expression.
  - $x("//p"): //returns all the <p> elements on the page:

```
> $x("//p")
<· ▼(32) [p, p, p, p, p, p, p, p, p.w3-hide-large.w3-hide-medium, p, p.w3-hide-large.w3-hide-medium, p, p, p, p, p, p, p, p, p, p, p, p, p, p, p, p, p, p, p, p, p, p, p, p, p, p]
    ▶0: p
    ▶1: p
    ▶2: p
    ▶3: p
    ▶4: p
    ▶5: p
    ▶6: p
```

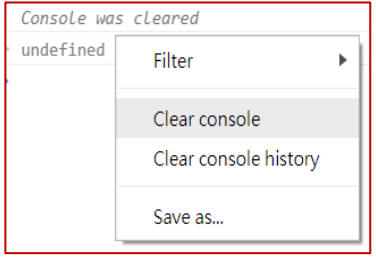
  - $x("//p[a]") : returns all **the <p> elements** that contain **<a> elements**

```
> $x("//p[a]")
<· ▼(4) [p, p, p, p]
    ▶0: p
    ▶1: p
    ▶2: p
    ▶3: p
     length: 4
    ▶__proto__: Array(0)
```

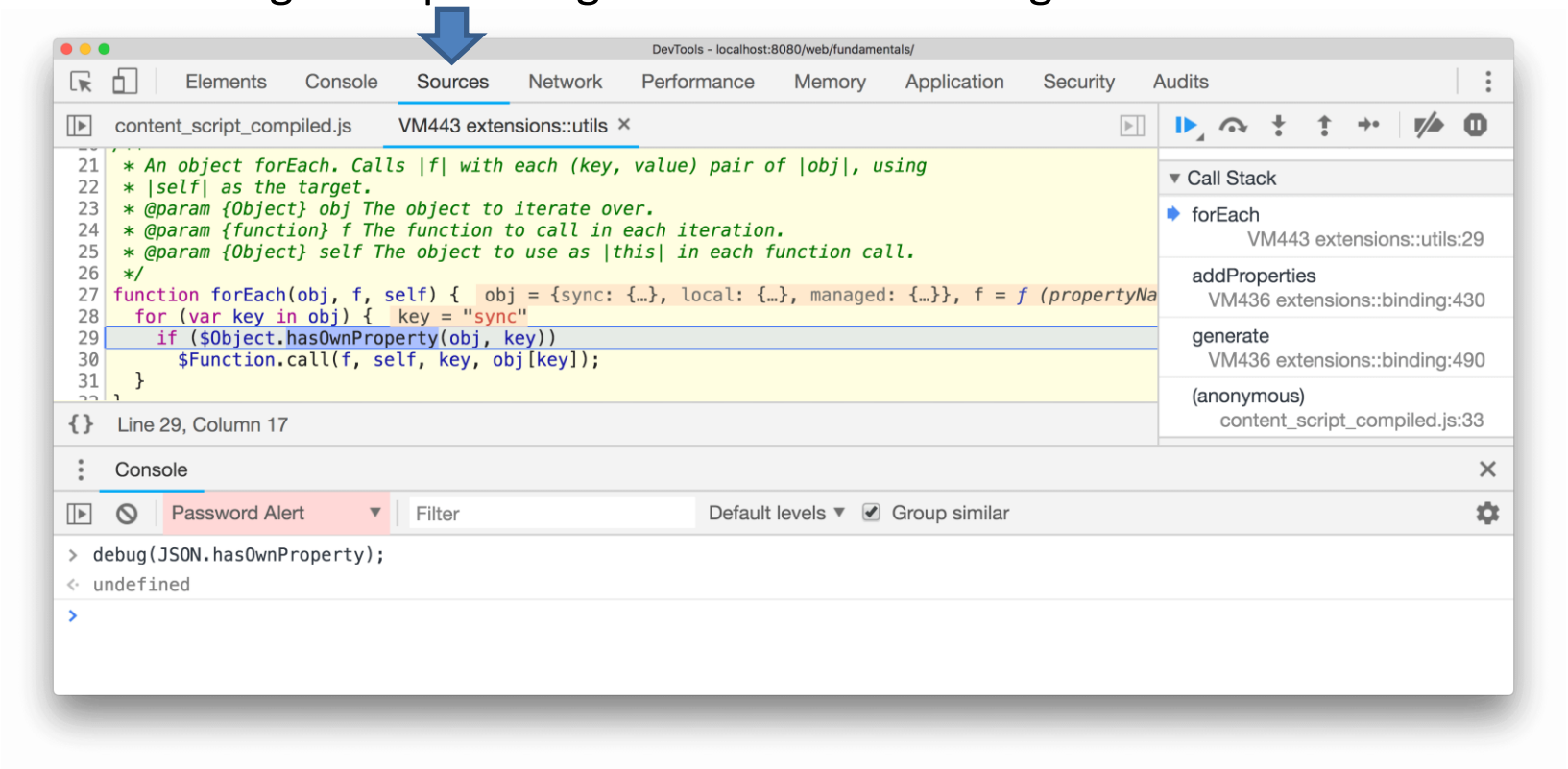
  - $x("//p", document): specifies an element or Node from which to search for elements.

```
$x("//iframe", document)
(31) [iframe#google_ads_iframe_/22152718/sws-hb//w3schools.com//main_leaderboard_1,
```

- clear(): clears the console of its history.



- copy(object):
  - copies a string representation of the specified object to the clipboard
  - copy($0);
- **debug(function)** vs **undebug(function)**:
  - the debugger is invoked and breaks inside the function on the Sources panel allowing to step through the code and debug it.



- dir(object)
  - displays an object-style listing of all the specified object's properties.
    - = Console API's console.dir() method.
  - document.body;
  - dir(document.body);

```
> document.body;
  dir(document.body);
  ▼ body
      aLink: ""
      accessKey: ""
      assignedSlot: null
    ▶ attributeStyleMap: StylePropertyMap {size: 3}
    ▶ attributes: NamedNodeMap {0: style, style: style, length: 1}
      autocapitalize: ""
      background: ""
      baseURI: "https://www.w3schools.com/css/css_website_layout.asp"
      bgColor: ""
      childElementCount: 14
```

- dirxml(object):
  - prints an **XML representation** of the specified object, as seen in the **Elements** tab.
  - = console.dirxml() method.
- inspect(object/function):
  - opens and selects the specified element or object in the appropriate panel:
    - either the Elements panel for DOM elements or the Profiles panel for JavaScript heap objects
  - inspect(document.body);

```
<!doctype html>
<html lang="en-US" style="height: 100%;">
▶ <head>…</head>
▼ <body style="position: relative; min-height: 100%; top: 0px;"> == $0
  ▶ <div class="w3-container top">…</div>
  ▶ <div style="display: none; position: absolute; z-index: 4; right: 52px; height: 44px; background-color: rgb(95, 95, 95);
    letter-spacing: normal;" id="googleSearch">…</div>
  ▶ <div style="display: none; position: absolute; z-index: 3; right: 111px; height: 44px; background-color: rgb(95, 95, 95);
    text-align: right; padding-top: 9px;" id="google_translate_element">…</div>
  ▶ <div class="w3-card-2 topnav" id="topnav" style="position: relative; top: 0px;">…</div>
  ▶ <div class="w3-sidebar w3-collapse" id="sidenav" style="top: 144px;">…</div>
```

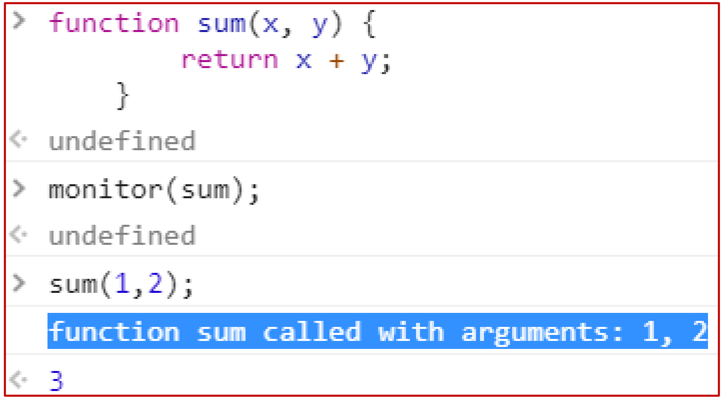
- getEventListeners(object)
  - returns the **event listeners registered** on the specified object.
  - The return value is an object that contains **an array for each registered event type**:
    - EX: click or keydown
  - getEventListeners(document);

```
> getEventListeners(document);
<· ▼{visibilitychange: Array(1)}
  ▼visibilitychange: Array(1)
    ▶0: {listener: ƒ, useCapture: false, passive: false, once: false, type: "visibilitychange"}
     length: 1
    ▶__proto__: Array(0)
  ▶__proto__: Object
```

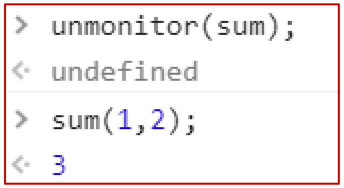
- **keys(object):**
  - returns an array containing the names of the properties belonging to the **specified object**.
  - **values():** get the associated values of the same properties
  - var player1 = { "name": "Ted", "level": 42 }
  - keys(player1)
  - values(player1)

```
Console   What's New   Quick source
top                   Filter
> var player1 = { "name": "Ted", "level": 42 }
<· undefined
> keys(player1)
<· ▶(2) ["name", "level"]
> values(player1)
<· ▶(2) ["Ted", 42]
>
```

- monitor(function) vs unmonitor(function):
  - When the function specified is called:
    - a message is logged to the console that indicates the function name along with the arguments that are passed to the function when it was called.
  - function sum(x, y) {
  - return x + y;
  - }
  - > **monitor(sum);**
  - > **sum(1,2);**

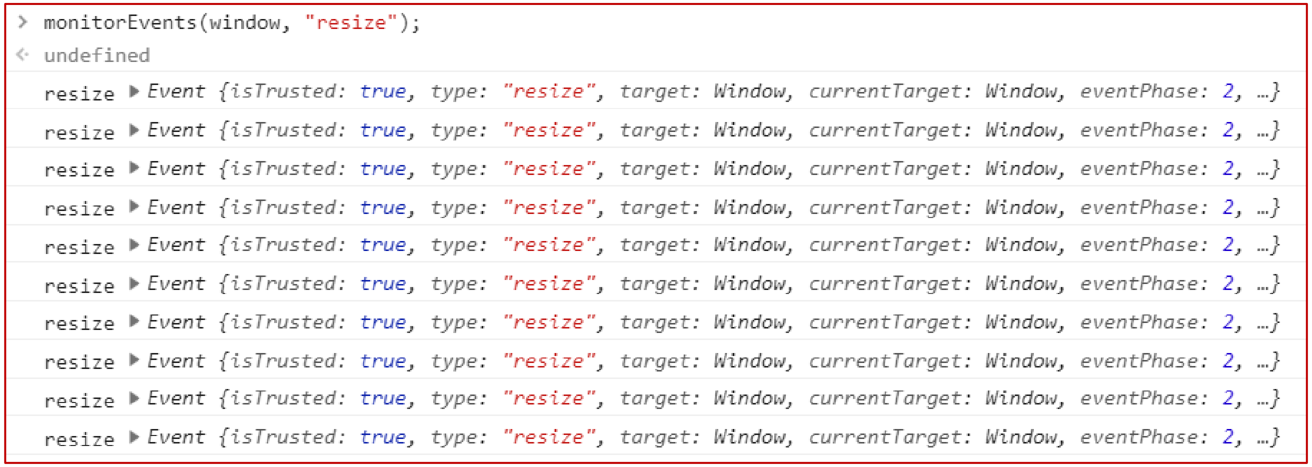


```
> function sum(x, y) {
          return x + y;
      }
<· undefined
> monitor(sum);
<· undefined
> sum(1,2);
  function sum called with arguments: 1, 2
<· 3
```



```
> unmonitor(sum);
<· undefined
> sum(1,2);
<· 3
```

- monitorEvents(object[, events]) vs unmonitorEvents(object[, events]):
  - When one of the specified events occurs on the specified object, the Event object is logged to the console.
  - monitorEvents(window, "resize");
    - monitors all resize events on the window object.

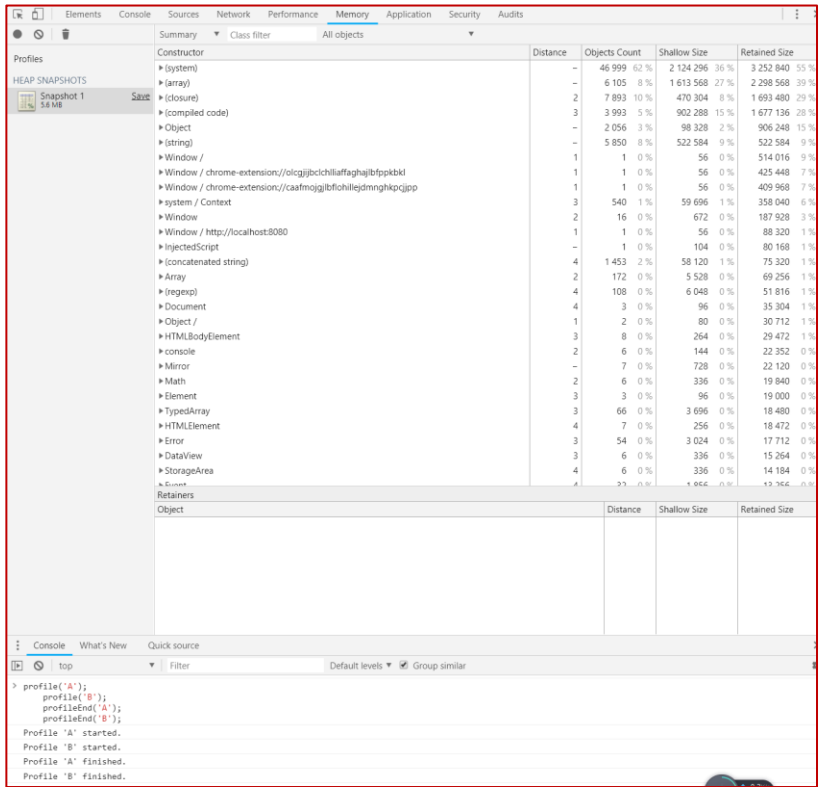


```
> monitorEvents(window, "resize");
<· undefined
  resize ▶ Event {isTrusted: true, type: "resize", target: Window, currentTarget: Window, eventPhase: 2, …}
  resize ▶ Event {isTrusted: true, type: "resize", target: Window, currentTarget: Window, eventPhase: 2, …}
  resize ▶ Event {isTrusted: true, type: "resize", target: Window, currentTarget: Window, eventPhase: 2, …}
  resize ▶ Event {isTrusted: true, type: "resize", target: Window, currentTarget: Window, eventPhase: 2, …}
  resize ▶ Event {isTrusted: true, type: "resize", target: Window, currentTarget: Window, eventPhase: 2, …}
  resize ▶ Event {isTrusted: true, type: "resize", target: Window, currentTarget: Window, eventPhase: 2, …}
  resize ▶ Event {isTrusted: true, type: "resize", target: Window, currentTarget: Window, eventPhase: 2, …}
  resize ▶ Event {isTrusted: true, type: "resize", target: Window, currentTarget: Window, eventPhase: 2, …}
  resize ▶ Event {isTrusted: true, type: "resize", target: Window, currentTarget: Window, eventPhase: 2, …}
  resize ▶ Event {isTrusted: true, type: "resize", target: Window, currentTarget: Window, eventPhase: 2, …}
```

- monitorEvents(window, ["resize", "scroll"]):
  - defines an array to monitor both "**resize**" and "**scroll**" events on the window object

Event type & Corresponding mapped events	
mouse	"mousedown", "mouseup", "click", "dblclick", "mousemove", "mouseover", "mouseout", "mousewheel"
key	"keydown", "keyup", "keypress", "textInput"
touch	"touchstart", "touchmove", "touchend", "touchcancel"
control	"resize", "scroll", "zoom", "focus", "blur", "select", "change", "submit", "reset"

- profile([name]) vs profileEnd([name]):
  - **profile():** starts a JavaScript CPU profiling session with an optional name.
  - **profileEnd():** completes the profile and displays the results in the Profile panel.
  - profile('A');
  - profile('B');
  - profileEnd('A');
  - profileEnd('B');



- table(data[, columns])
  - Log object data with table formatting by passing in a data object in with optional column headings.
  - var names = {
  - 0: { firstName: "John", lastName: "Smith" },
  - 1: { firstName: "Jane", lastName: "Doe" }
  - };
  - table(names);

```
> var names = {
        0: { firstName: "John", lastName: "Smith" },
        1: { firstName: "Jane", lastName: "Doe" }
    };
<· undefined
> table(names)
```

VM384:1

(index)	firstName	lastName
0	"John"	"Smith"
1	"Jane"	"Doe"

▶ Object

練習時間

(練習上述操作)

console.log

- The console.log method outputs a message in the console.
- console.log("I am logging to the console.");
  - it is basically the JavaScript equivalent of using a print method for debugging
  - use it to output values or check that we reach certain places in the code.
  - *// I want to know if I reach this part of the code.*
  - console.log("I am here!");
  - ***// I want to know the value of foo in this part of the code.***
  - **console.log("Foo: " + foo);**

```
> // I want to know if I reach this part of the code.
  console.log("I am here!");
  // I want to know the value of foo in this part of the code.
  console.log("Foo: " + foo);

  I am here!

  ▸Uncaught ReferenceError: foo is not defined
      at <anonymous>:4:23
```

思考: 如何讓此處正確顯示?

  - **for** (i = 0; i < 100; i++) { console.log(i); }
    - Unlike an alert, console.log does not stop the JavaScript from continuing.

- Alerts could only output strings, Console.log has no such limitation
- *// It can output dom elements.*
- console.log($('h2'));

- *// It can output objects.*
- console.log({ book: "Dune", characters: ["Paul", "Leto", "Jessica", "Chani", "sandworms"] });

```
> console.log({
    book: "Dune",
    characters: ["Paul", "Leto", "Jessica", "Chani", "sandworms"]
  });
  ▼{book: "Dune", characters: Array(5)}
     book: "Dune"
    ▶characters: (5) ["Paul", "Leto", "Jessica", "Chani", "sandworms"]
    ▶__proto__: Object
```

Console API Reference:

to write information to the console, create JavaScript profiles, and start a debugging session.

- console.assert(expression, object):
  - **Writes an error** to the console when the evaluated expression is false.
    - function greaterThan(a,b) {
    - console.assert(a > b, {"message":"a is not greater than b","a":a,"b":b});
    - }
    - greaterThan(5,6);

Elements Console Sources Network Performance Memory Application Security Audits

Page Filesystem » VM457 VM517 VM523 VM530 ×

top
localhost:8080
(index)

```
1 function greaterThan(a,b) {   a = 5, b = 6
2   console.assert(a > b, {"message":"a is not greater than b","a":a,"b":b});
3 }
4 greaterThan(5,6);
```

{} Line 2, Column 11

Console What's New Quick source

top Filter Default levels Group similar

```
> function greaterThan(a,b) {
    console.assert(a > b, {"message":"a is not greater than b","a":a,"b":b});
  }
  greaterThan(5,6);
✖ ▸Assertion failed: ▸{message: "a is not greater than b", a: 5, b: 6}
>
```

```
> greaterThan(7,6);
<· undefined
```

- console.clear(): Clears the console.
- console.count(label)
  - Writes the number of times that count() has been invoked at the same line and with the same label.
    - function login(name) {
    - console.count(name + ' logged in');
    - }
    - **login("A");**
    - **login('A');**

```
> function login(name) {
    console.count(name + ' logged in');
  }
<· undefined
> login("A")
  A logged in: 1
<· undefined
> login('A')
  A logged in: 2
<· undefined
```

- console.debug(object [, object, ...])
  - **=console.log()**
- console.dir(object)
- console.dirxml(object)
- **console.error**(object [, object, ...])
  - **Prints a message similar to console.log(),** styles the message like an error, and includes a stack trace from where the method was called.

```
> function logName(obj) {
    if (obj.name === undefined) {
      console.error('name is undefined');
    }
  }
<· undefined
> logName({});
✕ ▼name is undefined                                  VM2249:3
      logName              @ VM2249:3
      (anonymous function) @ VM2250:1
<· undefined
```

- console.warn(object [, object, ...]): like console.log(), but also displays a yellow warning icon next to the logged message.
  - console.warn('user limit reached!');

```
> console.warn('user limit reached!');
⚠ user limit reached!                                  VM1013:1
```

- console.group(object[, object, ...])
  - Starts a new logging group with an optional title.
    - function name(obj) {
    - console.group('name');
    - console.log('first: ', obj.first);
    - console.log('middle: ', obj.middle);
    - console.log('last: ', obj.last);
    - console.groupEnd();
    - }
    - name({"first":"Wile","middle":"E","last":"Coyote"});

```
> function name(obj) {
    console.group('name');
    console.log('first: ', obj.first);
    console.log('middle: ', obj.middle);
    console.log('last: ', obj.last);
    console.groupEnd();
  }

  name({"first":"Wile","middle":"E","last":"Coyote"});
▼ name
    first:  Wile
    middle:  E
    last:  Coyote
```

  - nest groups:
    - function name(obj) {
    - console.group('name');
    - console.log('first: ', obj.first);
    - console.log('middle: ', obj.middle);
    - console.log('last: ', obj.last);
    - console.groupEnd();
    - }

    - function doStuff() {
    - console.group('doStuff()');
    - name({"first":"Wile","middle":"E","last":"coyote"});
    - console.groupEnd();
    - }

    - doStuff();

```
> function name(obj) {
    console.group('name');
    console.log('first: ', obj.first);
    console.log('middle: ', obj.middle);
    console.log('last: ', obj.last);
    console.groupEnd();
  }

  function doStuff() {
    console.group('doStuff()');
    name({"first":"Wile","middle":"E","last":"coyote"});
    console.groupEnd();
  }

  doStuff();
▼ doStuff()
  ▼ name
      first:  Wile
      middle:  E
      last:  coyote
```

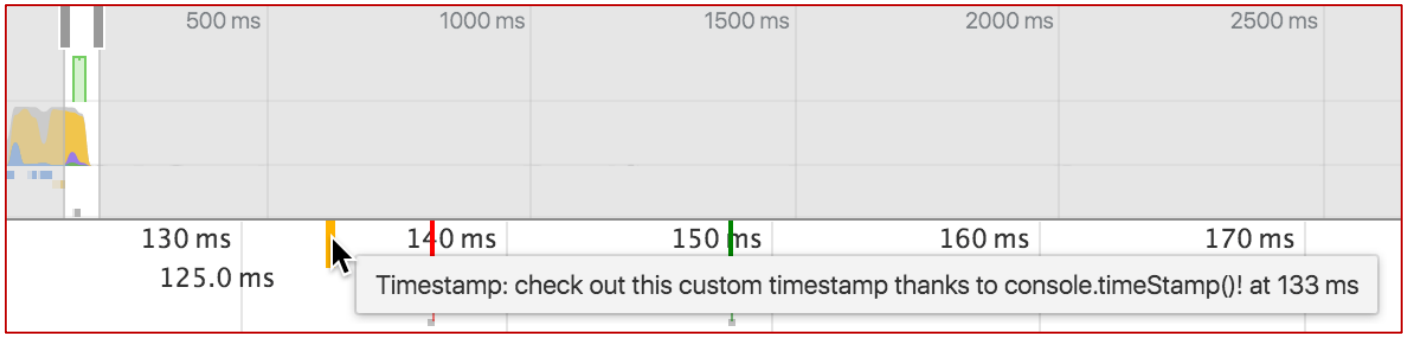
- console.groupCollapsed(object[, object, ...])
  - Creates a new logging group that is initially collapsed instead of open.
  - console.**groupCollapsed**('status');
  - console.log("peekaboo, you can't see me");
  - console.**groupEnd**();

```
> console.groupCollapsed('status');
  console.log("peekaboo, you can't see me");
  console.groupEnd();
▼ status
    peekaboo, you can't see me
```

- console.time([label]) vs console.timeEnd():
  - Starts a new timer vs Stop the timer and print the elapsed time to the Console.
    - console.**time**();
    - var arr = new Array(10000);
    - for (var i = 0; i < arr.length; i++) {
    - arr[i] = new Object();
    - }
    - console.**timeEnd**();

```
console.time();
var arr = new Array(10000);
for (var i = 0; i < arr.length; i++) {
  arr[i] = new Object();
}
console.timeEnd();
default: 14.81982421875ms
```

- console.timeStamp([label])
  - Adds an event to the **Timeline** during a recording session.
  - console.timeStamp('check out this custom timestamp thanks to console.timeStamp()!');



- console.trace(object)
  - Prints a stack trace from the point **where the method was called**.
  - function add(num){
  - if(num>0){
  - console.trace('recursion is fun:',num);
  - return num+add(num-1);
  - }else{
  - return 0;
  - }
  - }
  - add(3)

```
> function add(num){
   if(num>0){
      console.trace('recursion is fun:',num);
      return num+add(num-1);
   }else{
      return 0;
      }
  }
<· undefined
> add(3)
  ▼recursion is fun: 3
   add         @ VM566:3
   (anonymous) @ VM570:1
   greaterThan @ VM530:2
   (anonymous) @ VM530:4
  ▼recursion is fun: 2
   add         @ VM566:3
   add         @ VM566:4
   (anonymous) @ VM570:1
   greaterThan @ VM530:2
   (anonymous) @ VM530:4
  ▼recursion is fun: 1
   add         @ VM566:3
   add         @ VM566:4
   add         @ VM566:4
   (anonymous) @ VM570:1
   greaterThan @ VM530:2
   (anonymous) @ VM530:4
<· 6
```

```
var car;
var func1 = function() {func2();}
var func2 = function() {func4();}
var func3 = function() {}
var func4 = function() {car = new Car();car.funcX();}
var Car = function() {
        this.brand = 'volvo';
        this.color =  'red';
        this.funcX = function() {this.funcY();}
        this.funcY = function() {this.funcZ();}
        this.funcZ = function() {console.trace('trace car')}
}
func1();
```

```
▼trace car
Car.funcZ          @ VM10257:11
Car.funcY          @ VM10257:10
Car.funcX          @ VM10257:9
func4              @ VM10257:5
func2              @ VM10257:3
func1              @ VM10257:2
(anonymous)        @ VM10257:13
_.Ql.send          @ /_/scs/social-static…ukX67Q/m=_b,_tp:285
_.Sl               @ /_/scs/social-static…ukX67Q/m=_b,_tp:283
_.$t               @ m=sy5o,sy5p,XAzchc,s…c,Uas9Hd,LbJKvc:414
_.au.flush         @ m=sy5o,sy5p,XAzchc,s…c,Uas9Hd,LbJKvc:420
(anonymous)        @ m=sy5o,sy5p,XAzchc,s…,Uas9Hd,LbJKvc:1156
e.T                @ /_/scs/social-static…sukX67Q/m=_b,_tp:94
Tba                @ /_/scs/social-static…sukX67Q/m=_b,_tp:97
Pba                @ /_/scs/social-static…sukX67Q/m=_b,_tp:97
_.la.ua            @ /_/scs/social-static…sukX67Q/m=_b,_tp:97
Fba                @ /_/scs/social-static…sukX67Q/m=_b,_tp:89
Promise.then (async)
Ae                 @ /_/scs/social-static…sukX67Q/m=_b,_tp:89
Ee                 @ /_/scs/social-static…sukX67Q/m=_b,_tp:89
Qba                @ /_/scs/social-static…sukX67Q/m=_b,_tp:97
Fe                 @ /_/scs/social-static…sukX67Q/m=_b,_tp:96
(anonymous)        @ /_/scs/social-static…sukX67Q/m=_b,_tp:90
(anonymous)        @ m=sy5y,_latency,sy3q,FCpbqb,WhJNk:8
td.execute         @ /_/scs/social-static…sukX67Q/m=_b,_tp:72
c                  @ /_/scs/social-static…ukX67Q/m=_b,_tp:273
vd                 @ /_/scs/social-static…sukX67Q/m=_b,_tp:73
iba                @ /_/scs/social-static…sukX67Q/m=_b,_tp:73
_.q                @ /_/scs/social-static…ukX67Q/m=_b,_tp:115
(anonymous)        @ m=sy5y,_latency,sy3q,FCpbqb,WhJNk:9
(anonymous)        @ m=sy5y,_latency,sy3q,FCpbqb,WhJNk:29
```

https://raygun.com/javascript-debugging-tips

Find the important things in complex debugging

- Use CSS and make your own structured console messages
  - console.**todo** = function(msg) {
  - console.log('%c %s %s %s', **'color**: yellow; **background-color**: black;', '–', msg, '–');
  - }
  - console.**important** = function(msg) {
  - console.log('%c %s %s %s', **'color**: brown; **font-weight**: bold; **text-decoration**: underline;', '–', msg, '–');
  - }
  - console.**todo**("This is something that's need to be fixed");
  - console.**important**('This is an important message');

```
> console.todo = function(msg) {
      console.log('%c %s %s %s', 'color: yellow; background-color: black;', '-', msg, '-');
  }

  console.important = function(msg) {
      console.log('%c %s %s %s', 'color: brown; font-weight: bold; text-decoration: underline;', '-', msg, '-');
  }

  console.todo("This is something that's need to be fixed");
  console.important('This is an important message');
  - This is something that's need to be fixed -
  - This is an important message -
```


https://raygun.com/javascript-debugging-tips

練習時間

(練習上述操作)

Interactive Debugger

- provides a rich toolset useful for debugging your code.
- **debugger;**
  - start the debugger at a specific point in your code by calling **debugger;**
  - *// I want to start debugging here.*
  - **debugger**;
- Test by: http://juliepagano.com/blog/2014/05/18/javascript-debugging-for-beginners/
- console:> testDebugging(); or Click to try out debugging

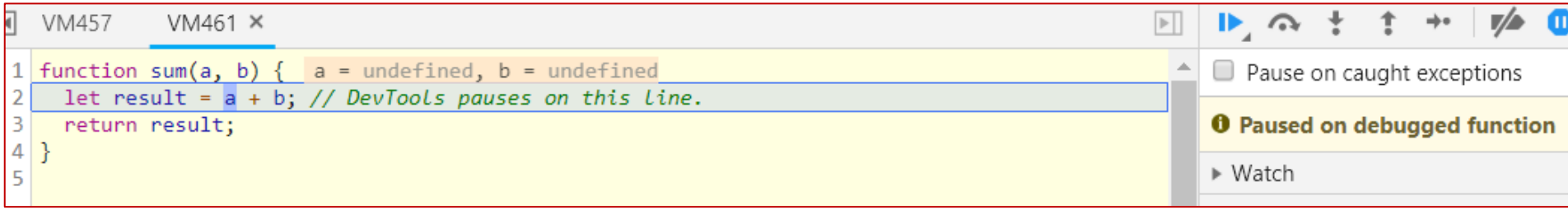
(index) debugging.js × content.js

```
1  $('.solution').on('click', 'a', function() {
2    $(this).siblings().toggle();
3    return false;
4  });
5
6  var foo = "bar";
7  var exampleVariable = ["walk", "without", "rhythm"];
8
9  function addJQuery(isDevelopment) {
10   var script = $("script[src *= 'jquery']");
11   var src = script.attr('src');
12   var path = src.substr(0, src.lastIndexOf('/') + 1);
13   var file = isDevelopment ? 'jquery.js' : 'jquery.min.js';
14   var newScript = $('<script src="' + path + file + '" type="text/javascript"></script>')
15   $("body").append(newScript);
16 }
17
18 function testDebugging() {
19   debugger;
20   makeItColor('red');
21   makeItColor('green');
22   makeItColor('red');
23   makeItColor('');
24 };
25
26 function makeItColor(color) {
27   $('body').css('background-color', color);
28 };
```

Debugger paused

- Watch
- Call Stack
  - testDebugging debugging.js:19
  - (anonymous) VM737:1
- Scope
- Local
  - this: Window
- Global Window
- Breakpoints
  - *No breakpoints*
- XHR/fetch Breakpoints
- DOM Breakpoints
- Global Listeners
- Event Listener Breakpoints

- **Function breakpoints:**
  - Call **debug(functionName)**, where functionName is the function you want to debug
  - function sum(a, b) {
  - let result = a + b; // DevTools pauses on this line.
  - return result;
  - }
  - debug(sum); // Pass the function object, not a string.



  - **Make sure the target function is in scope:**
    - (function () {
    - function hey() {
    - console.log('hey');
    - }
    - function yo() {
    - console.log('yo');
    - }
    - debug(yo); // This works.
    - yo();
    - })();
    - **debug(hey);** // This doesn't work. hey() is out of scope.

```
> (function () {
    function hey() {
      console.log('hey');
    }
    function yo() {
      console.log('yo');
    }
    debug(yo); // This works.
    yo();
  })();
> debug(hey); // This doesn't work. hey() is out of scope.
✕ ▸Uncaught ReferenceError: hey is not defined
      at eval (eval at yo ((index):6), <anonymous>:1:7)
      at yo (<anonymous>:6:5)
      at <anonymous>:9:3
      at <anonymous>:10:3
```

https://developers.google.com/web/tools/chrome-devtools/javascript/breakpoints

- Line-of-code breakpoints
  - when you know the exact region of code that you need to investigate
- Line-of-code breakpoints in your code
  - Call debugger from your code to pause on that line
    - console.log('a');
    - console.log('b');
    - debugger;
    - console.log('c');

```
1 console.log('a');
2 console.log('b');
3 debugger;
4 console.log('c');
```

- Conditional line-of-code breakpoints
  - to pause only when some other condition is true
  - i=1;
  - j=2;
  - k=i+j;
  - console.log(k);
  - debugger;

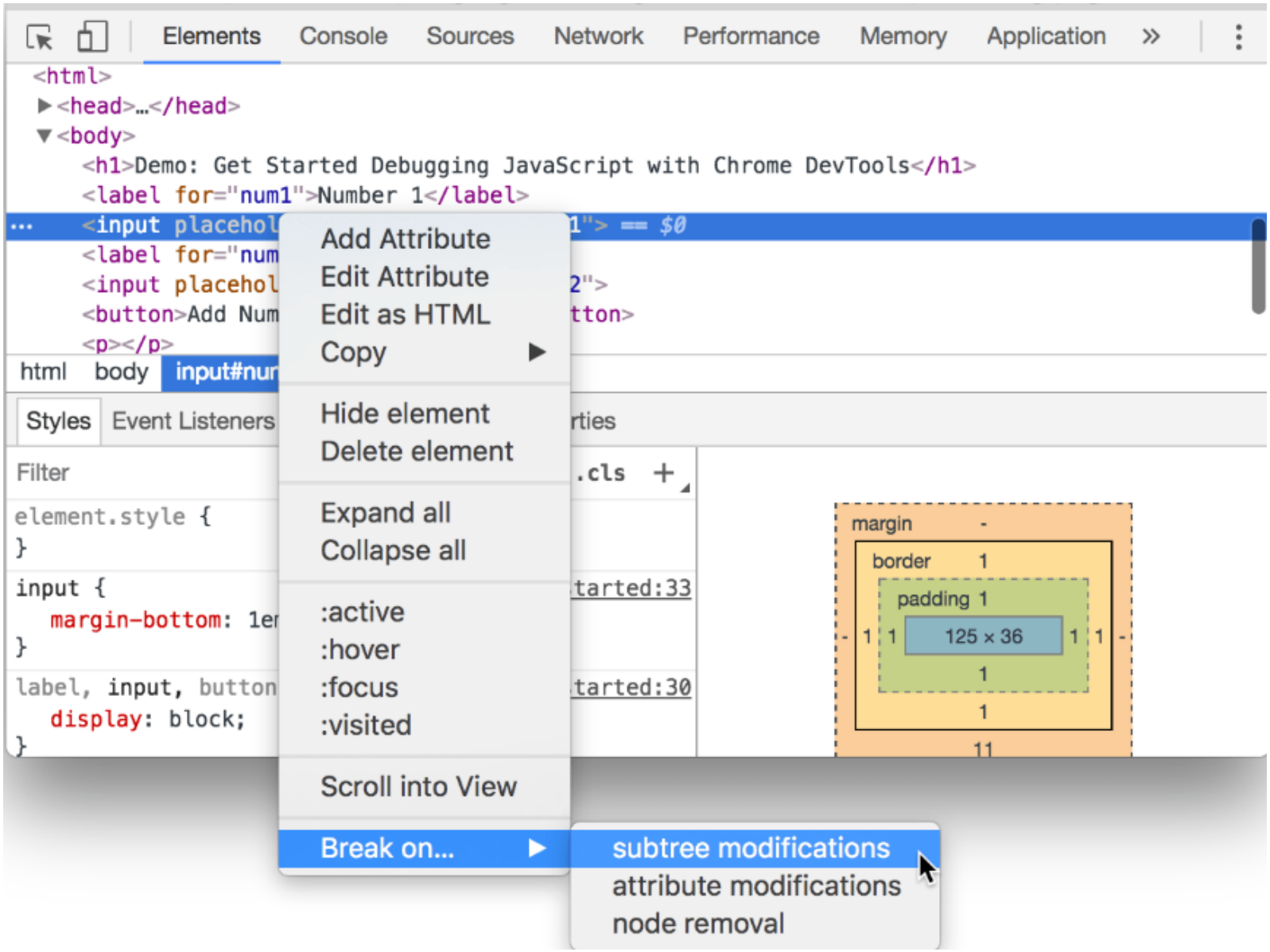
```
1 i=1;
2 j=2;
3 k=i+j;
  The breakpoint on line 3 will stop only if this expression is tr
  k=3
4 console.log(k); debugger;
```

橘色標記

```
1 i=1;
2 j=2;
3 k=i+j;
4 console.log(k); debugger;
5
```

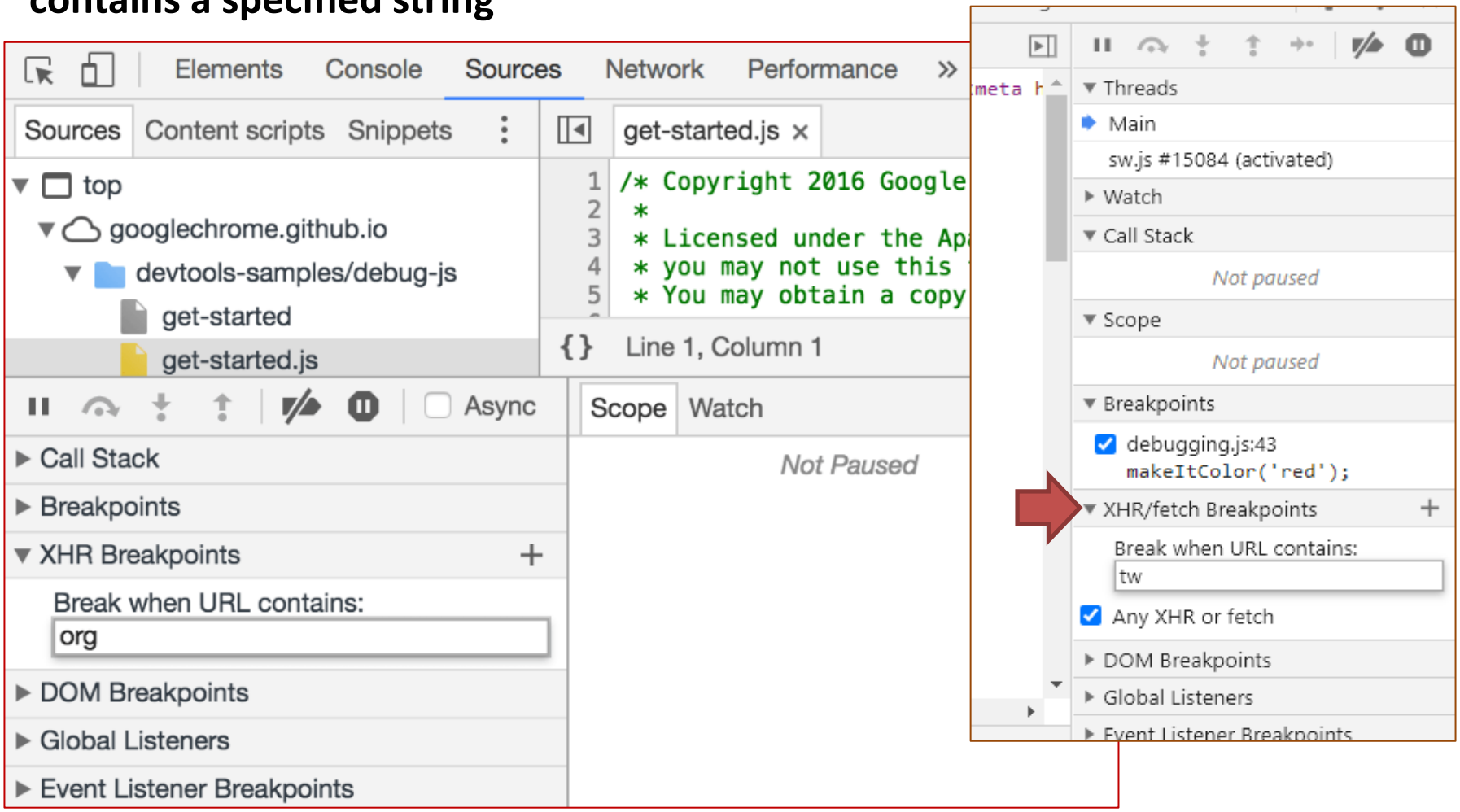
• DOM change breakpoints

– Use a **DOM change breakpoint** when you want to pause on the code that changes a DOM node or its children.



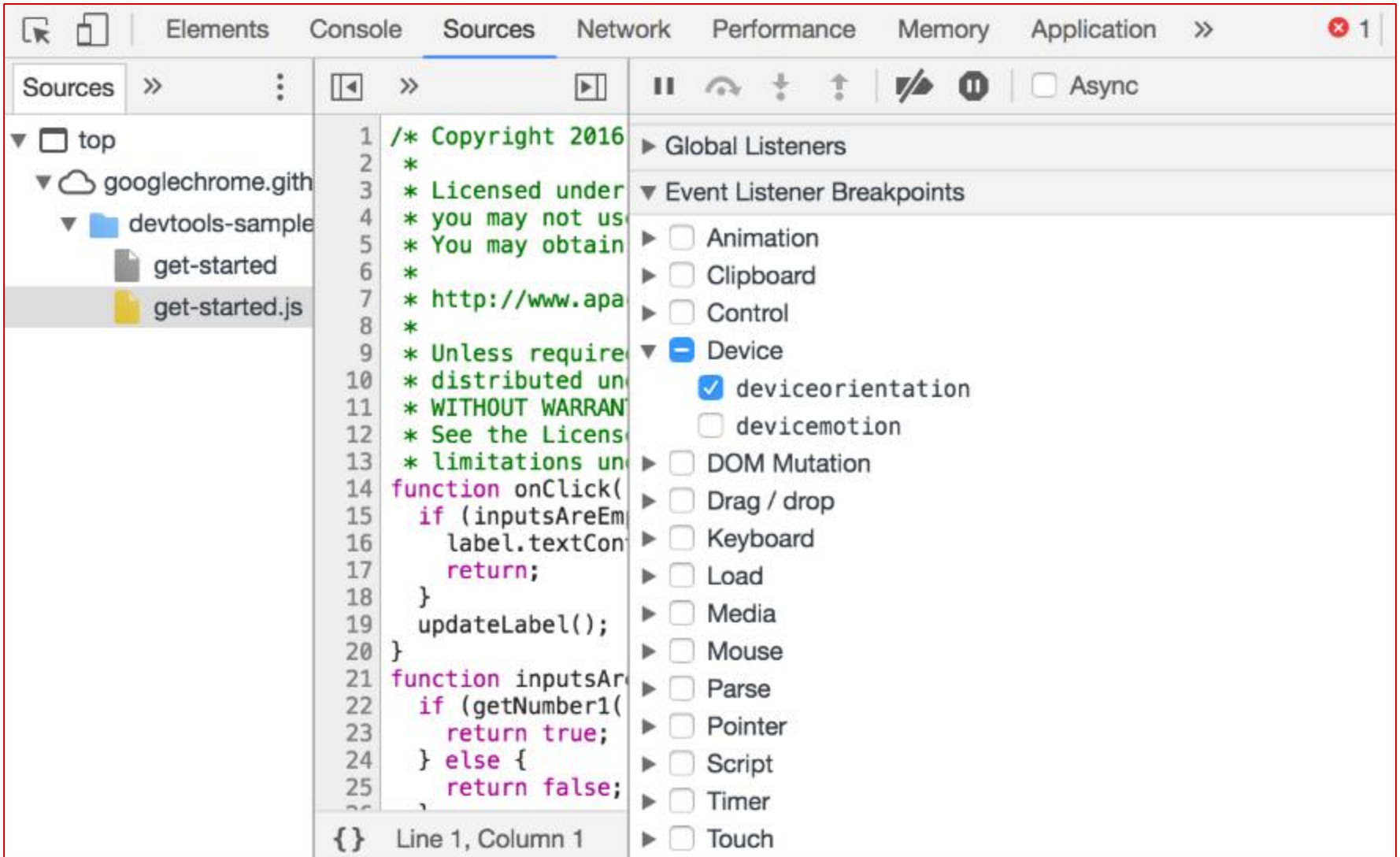
XHR/Fetch breakpoints (XMLHttpRequest)

- an **XHR breakpoint** when you want to break when the **request URL of an XHR contains a specified string**



Event listener breakpoints

- event listener breakpoints when you want to pause on the event listener code that runs after an event is fired



Exception breakpoints

- **exception breakpoints** when you want to pause on the line of code that's throwing a caught or uncaught exception

Elements Console Sources Network Performance Memory »

Sources »

top
googlechrome.github.io
devtools-samples/debug-
get-started
get-started.js

Fetch content_script_compiled.js get-started.js ×

```
17      return;
18    }
19    throw "whoops";
20    updateLabel();
21  }
```

{} Line 19, Column 3

Async

Pause On Caught Exceptions

Paused on exception
whoops

Call Stack
Breakpoints
XHR Breakpoints
DOM Breakpoints

Scope Watch

Local
Exception: "whoops"
this: button
Global Window

Libraries and Minified Code

- This code is minified.

Elements Network Sources Timeline Profiles Resources Audits Console

Sources Cont... Snipp...

- (no domain)
- ajax.googleapis.com
  - ajax/libs/jquery/2.1.1
    - jquery.min.js
- hgimnogjllphhhkhlmebbml
- localhost:4567
- use.typekit.net
- www.google-analytics.com

jquery.min.js ×

```
/*! jQuery v2.1.1 | (c) 2005, 2014 jQuery Foundation, Inc. | jquery.org/license */
!function(a,b){"object"==typeof module&&"object"==typeof module.exports?module.export
},_data:function(a,b,c){return L.access(a,b,c)},_removeData:function(a,b){L.remove(a,
},removeAttr:function(a,b){var c,d,e=0,f=b&&b.match(E);if(f&&1===a.nodeType)while(c=f
```

{} Line 1, Column 1

Elements Network Sources Timeline Profiles Resources Audits Console

Sources Cont... Snipp...

- (no domain)
- ajax.googleapis.com
  - ajax/libs/jquery/2.1.1
    - jquery.min.js
- hgimnogjllphhhkhlmebbml
- localhost:4567
- use.typekit.net
- www.google-analytics.com

jquery.min.js ×

```
 /*! jQuery v2.1.1 | (c) 2005, 2014 jQuery Foundation, Inc. | jquery.org/license */
!function(a, b) {
    "object" == typeof module && "object" == typeof module.exports ? module.exports
        if (!a.document)
            throw new Error("jQuery requires a window with a document");
        return b(a)
    } : b(a)
}("undefined" != typeof window ? window : this, function(a, b) {
    var c = [], d = c.slice, e = c.concat, f = c.push, g = c.indexOf, h = {}, i = h.
        return new n.fn.init(a, b)
    }, o = /^[\s\uFEFF\xA0]+|[\s\uFEFF\xA0]+$/g, p = /^-ms-/, q = /-([\da-z])/gi, r
        return b.toUpperCase()
    };
    n.fn = n.prototype = {jquery: m,constructor: n,selector: "",length: 0,toArray: f
            return d.call(this)
        },get: function(a) {
            return null != a ? 0 > a ? this[a + this.length] : this[a] : d.call(this
```

{} Line 1, Column 2

Libraries and Minified Code

Paused in debugger

Let's try it out! Click to try out debugging

Go to the line in `makeItColor` where we are using jQuery functions.

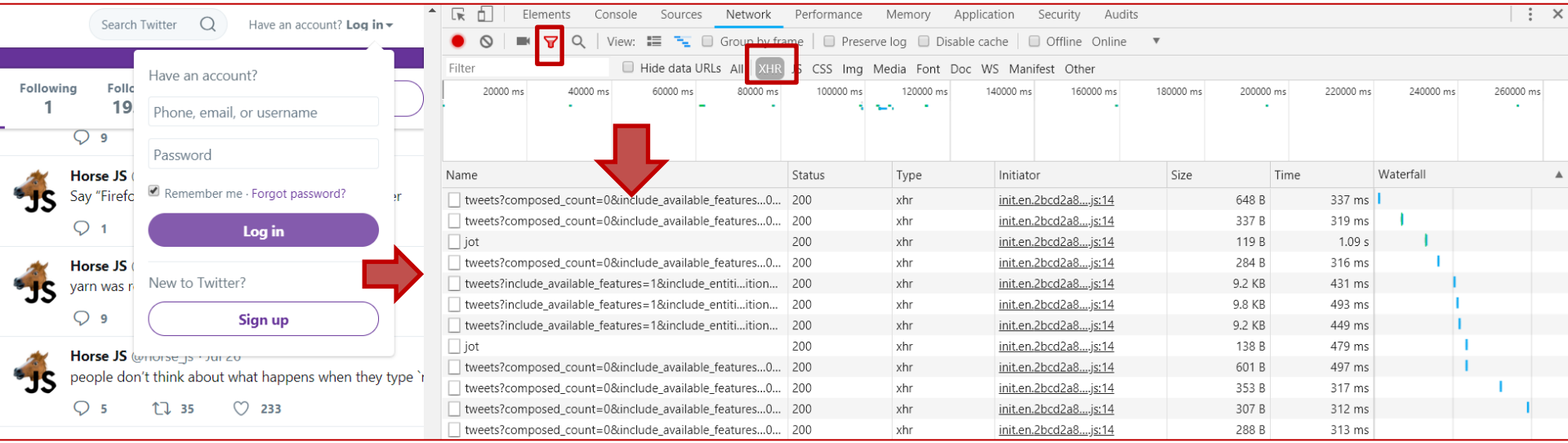
Step into the code. This is stepping into `$('body')` to find and return the body element.

```
$('.solution').on('click', 'a', function() {
  $(this).siblings().toggle();
  return false;
});

var foo = "bar";
var exampleVariable = ["walk", "without", "rhythm"];

function addJQuery(isDevelopment) {
  var script = $("script[src *= 'jquery']");
  var src = script.attr('src');
  var path = src.substr(0, src.lastIndexOf('/') + 1);
  var file = isDevelopment ? 'jquery.js' : 'jquery.min.js';
  var newScript = $('<script src="' + path + file + '" type="text/javascript"></script>')
  $("body").append(newScript);
}

function testDebugging() {
  debugger;
  makeItColor('red');
  makeItColor('green');
  makeItColor('red');
  makeItColor('');
};

function makeItColor(color) {
  $('body').css('background-color', color);
};
```

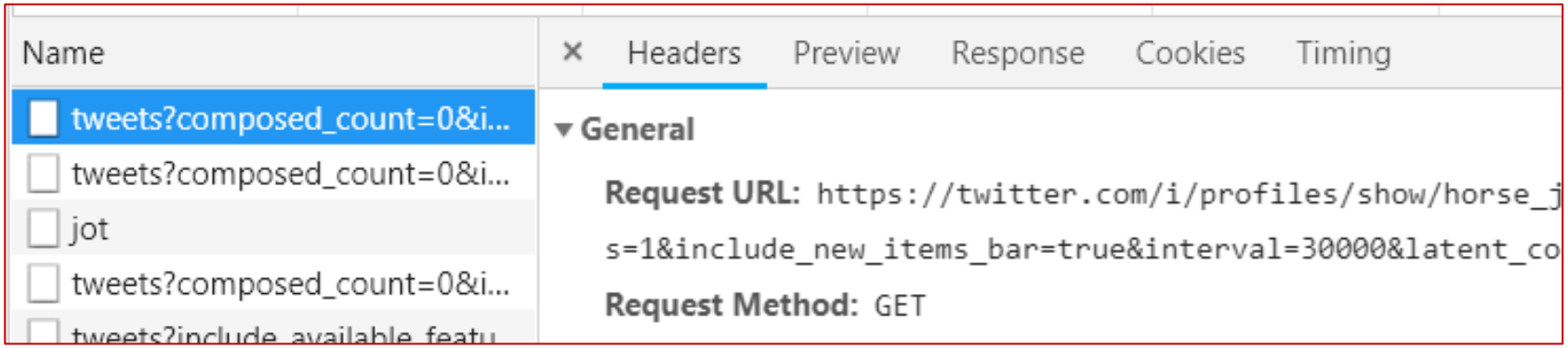
```
var testAjax = {};

testAjax.toggleLoader = function(show) {
  $('#dune-quotes .loader').toggle(show);
};

testAjax.ajax = function(source) {
  $('#dune-quotes .container').empty();
  testAjax.toggleLoader(true);

  window.setTimeout(function() {
    $.ajax({
```

AJAX Requests

- The network tab of the developer tools is really helpful for testing issues related to ajax requests.
  - Go to @horse_js (or some other twitter user you like better).
  - Open the Chrome developer tools.
  - Go to the network tab.
  - Click on the filter icon
  - Click the **XHR filter** option (XMLHttpRequest):
    - will limit us to the ajax request we're looking for.



下拉產生ajax request



- **Headers** tab:
  - provides you with information about the headers sent and received in the ajax request.

- **Preview** tab:
  - lets you preview information about the ajax response.

- **Response** tab:
  - lets you see the ajax response as text without the developer tools doing anything special to it.

- **Cookies** tab:
  - shows you the cookies associated with the request.

Try all the sizes

iPad Pro ▼ 1024 × 1366 50% ▼ Online ▼

- Responsive
- Galaxy S5
- Pixel 2
- Pixel 2 XL
- iPhone 5/SE
- iPhone 6/7/8
- iPhone 6/7/8 Plus
- iPhone X
- iPad
- ✓ iPad Pro
- Edit...

```
// It
consol
// It
consol
  book
  characters: ["Paul", "Leto", "Jessica", "Chani", "sandworms"]
});
```

This is just the beginning. There is a lot more you can do. I recommend reviewing the console api reference for more information.

The console logging provides enough for you to get by, and some developers stop there with their JavaScript debugging skills. However, it's not always the best tool for the job. Debugging complex JavaScript using console logging can be slow and painful. Thankfully, there are even more debugging tools available to you.

Interactive Debugger

The Chrome Developer tools has an interactive JavaScript debugger that provides a rich toolset useful for debugging your code. I find this especially useful for complex JavaScript code and code that interacts with other libraries.

You can start the debugger at a specific point in your code by calling `debugger`.

```
// I want to start debugging here.
debugger;
```

The debugger will only do something when the developer tools are open. Close the developer tools and click the button. Hmm, it looks like nothing happened. Now try opening the developer tools and clicking it.

Click to try out debugging

You should see something like this. Now you can start debugging!

練習時間

(練習上述操作)

兩個好用的前端網頁開發IDE

- jsfiddle:
  - https://jsfiddle.net/
    - 簡易教學

- Codepen:
  - https://codepen.io
    - 簡易教學

THE END

Q&A