Which "href" value should I use for JavaScript links, "#" or "javascript:void(0)"?

Asked 16 years, 3 months ago Modified 6 days ago Viewed 2.5m times

4450

The following are two methods of building a link that has the sole purpose of running JavaScript code. Which is better, in terms of functionality, page load speed, validation purposes, etc.?

```
function myJsFunc() {
    alert("myJsFunc");
}
```

```
<a href="#" onclick="myJsFunc();">Run JavaScript Code</a>
```

Run code snippet Expand snippet

or

```
function myJsFunc() {
    alert("myJsFunc");
}
```

```
 <a href="javascript:void(0)" onclick="myJsFunc();">Run JavaScript Code</a>
```

Run code snippet Expand snippet

javascript html performance optimization href

Share Follow edited Feb 19, 2017 at 8:58 community wiki
18 revs, 12 users 36%
Peter Mortensen

14 Why use a link when you want a button? Then there is no issue with pseudo–protocols. – RobG Jun 11, 2022 at 12:57

Neither. If you really *must* use a link, then use a span styled as a link. No href to bother about. And why `void(0)` when `void 0` will do? – RobG Apr 16, 2023 at 7:43

55 Answers

Sorted by: Highest score (default)

1 2 Next

2317

+50

I use `javascript:void(0)`.

Three reasons. Encouraging the use of `#` amongst a team of developers inevitably leads to some using the return value of the function called like this:

```
function doSomething() {
    //Some code
    return false;
}
```

But then they forget to use `return doSomething()` in the onclick and just use `doSomething()`.

A second reason for avoiding `#` is that the final `return false;` will not execute if the called function throws an error. Hence the developers have to also remember to handle any error appropriately in the called function.

A third reason is that there are cases where the `onclick` event property is assigned dynamically. I prefer to be able to call a function or assign it dynamically without having to code the function specifically for one method of attachment or another. Hence my `onclick` (or on anything) in HTML markup look like this:

```
onclick="someFunc.call(this)"
```

OR

```
onclick="someFunc.apply(this, arguments)"
```

Using `javascript:void(0)` avoids all of the above headaches, and I haven't found any examples of a downside.

So if you're a lone developer then you can clearly make your own choice, but if you work as a team you have to either state:

Use `href="#"`, make sure `onclick` always contains `return false;` at the end, that any called function does not throw an error and if you attach a function dynamically to the `onclick` property make sure that as well as not throwing an error it returns `false`.

OR

Use `href="javascript:void(0)"`

The second is clearly much easier to communicate.

Share Follow

edited Feb 20, 2016 at 2:02

community wiki
6 revs, 5 users 76%
AnthonyWJones

---

174 Fast-forward to 2013: `javascript:void(0)` violates [Content Security Policy](#) on CSP-enabled HTTPS pages. One option would be then to use `href='#'` and `event.preventDefault()` in the handler, but I don't like this much. Perhaps you can establish a convention to use `href='#void'` and make sure no element on the page has `id="void"`. That way, clicking a link to non-existing anchor will not scroll the page. – jakub.g Oct 1, 2013 at 9:21

36 You can use "#" and then bind a click event to all links with "#" as the `href`. This would be done in jQuery with: `$('a[href="#"]').click(function(e) { e.preventDefault ? e.preventDefault() : e.returnValue = false; });` – Nathan Feb 14, 2014 at 4:58

7 `#void` would add an entry to the browser history nonetheless. Another way would be to use the URL of a resource that returns HTTP status 204 (and still use `preventDefault` - the 204 is just a fallback). – CherryDT Feb 15, 2021 at 22:35

@Nathan Non-jquery: `const voidLinks = document.querySelectorAll('a[href="#"') for (const voidLink of voidLinks) { voidLink.addEventListener('click', e => { e.preventDefault ? e.preventDefault() : e.returnValue = false }) voidLink.addEventListener('keypress', e => { if (e.keyCode === 13) e.preventDefault ? e.preventDefault() : e.returnValue = false }) }` That is including event listener on keypress Enter (code 13) for people navigation through page with keyboard. – s3c May 31, 2021 at 12:18

8 @HPWD This is [cargo cult programming](#). [`void`](#) is an *operator*, not a function. It's `void 0`; the parentheses in `void(0)` don't do anything. `void()` is just as invalid as, say, `return ();`. `()` is not a valid UnaryExpression. – Sebastian Simon Feb 5, 2022 at 20:07

1417

+100

Neither.

If you can have an actual URL that makes sense use that as the HREF. The onclick won't fire if someone middle-clicks on your link to open a new tab or if they have JavaScript disabled.

If that is not possible, then you should at least inject the anchor tag into the document with JavaScript and the appropriate click event handlers.

I realize this isn't always possible, but in my opinion it should be striven for in developing any public website.

Check out *Unobtrusive JavaScript* and *Progressive enhancement* (both Wikipedia).

Share Follow

edited Mar 11, 2012 at 8:30

community wiki
3 revs, 2 users 76%
Aaron Wagner

53 *"If that is not possible, then" use a button instead.* That's how this answer should've ended. – Sebastian Simon May 29, 2021 at 0:52

and if we use a button, how would our lovely users will open their href in new tab using their shiny smartphones? – mohamed tebry Feb 11, 2022 at 4:57

6 @mohamedtebry—you've missed the point of the question: "*building a link that has the sole purpose of running JavaScript code*". The issue is using a link as a button, there is no href to open. – RobG Jun 11, 2022 at 12:59

@SebastianSimon Bootstrap dropdowns have anchor structure for styling. Sometimes it is necessary to use anchor getbootstrap.com/docs/5.3/components/dropdowns/#single-button – Cemstrian Aug 28, 2023 at 13:39

811

Doing `<a href="#" onclick="myJsFunc();">Link</a>` or `<a href="javascript:void(0)" onclick="myJsFunc();">Link</a>` or whatever else that contains an `onclick` attribute - was okay back five years ago, though now it can be a bad practice. Here's why:

1. It promotes the practice of obtrusive JavaScript - which has turned out to be difficult to maintain and difficult to scale. More on this in *Unobtrusive JavaScript*.
2. You're spending your time writing incredibly overly verbose code - which has very little (if any) benefit to your codebase.
3. There are now better, easier, and more maintainable and scalable ways of accomplishing the desired result.

The unobtrusive JavaScript way

Just don't have a `href` attribute at all! Any good CSS reset would take care of the missing default cursor style, so that is a non-issue. Then attach your JavaScript functionality using graceful and unobtrusive best practices - which are more maintainable as your JavaScript logic stays in JavaScript, instead of in your markup - which is essential when you start developing large scale JavaScript applications which require your logic to be split up into blackboxed components and templates. More on this in *Large-scale JavaScript Application Architecture*

Simple code example

```
// Cancel click event
$('.cancel-action').click(function(){
    alert('Cancel action occurs!');
});

// Hover shim for Internet Explorer 6 and Internet Explorer 7.
$(document.body).on('hover','a',function(){
    $(this).toggleClass('hover');
});
```

```
a { cursor: pointer; color: blue; }
a:hover,a.hover { text-decoration: underline; }
```

```
<script src="https://ajax.googleapis.com/ajax/libs/jquery/2.1.1/jquery.min.js">
</script>
<a class="cancel-action">Cancel this action</a>
```

Run code snippet   Expand snippet

A blackboxed *Backbone.js* example

For a scalable, blackboxed, Backbone.js component example - *see this working jsfiddle example here*. Notice how we utilize unobtrusive JavaScript practices, and in a tiny amount of code have a component that can be repeated across the page multiple times without side-effects or conflicts between the different component instances. Amazing!

Notes

- Omitting the `href` attribute on the `a` element will cause the element to not be accessible using `tab` key navigation. If you wish for those elements to be accessible via the `tab` key, you can set the `tabindex` attribute, or use `button` elements instead. You can easily style button elements to look like normal links as mentioned in *Tracker1's answer*.
- Omitting the `href` attribute on the `a` element will cause Internet Explorer 6 and Internet Explorer 7 to not take on the `a:hover` styling, which is why we have added a simple JavaScript shim to accomplish this via `a.hover` instead. Which is perfectly okay, as if you don't have a href attribute and no graceful degradation then your link won't work anyway - and you'll have bigger issues to worry about.
- If you want your action to still work with JavaScript disabled, then using an `a` element with a `href` attribute that goes to some URL that will perform the action manually instead of via an Ajax request or whatever should be the way to go. If you are doing this, then you want to ensure you do an `event.preventDefault()` on your

click call to make sure when the button is clicked it does not follow the link. This option is called graceful degradation.

Share Follow

edited May 23, 2017 at 12:10

community wiki
17 revs, 5 users 69%
balupton

And what about passing a parameter tot he final JS method? Is it possible? – DoomerDGR8 Nov 22, 2022 at 8:16

360

`'#'` will take the user back to the top of the page, so I usually go with `void(0)`.

`javascript:;` also behaves like `javascript:void(0);`

Share Follow

edited May 11, 2017 at 4:33

community wiki
4 revs, 4 users 43%
Gabe Rogan

72 The way to avoid that is to return false in the onclick event handler. – Guvante Sep 25, 2008 at 21:22

39 Returning false in the event handler doesn't avoid that if JavaScript the JS doesn't run successfully. – Quentin Sep 26, 2008 at 8:10

34 using "#someNonExistantAnchorName" works well because it has nowhere to jump to. – scunliffe Sep 27, 2008 at 2:46

28 If you have a base href, then `#` or `#something` will take you to that anchor *on the base href* page, instead of on the current page. – Abhi Beckert Mar 28, 2012 at 6:39

15 The shebang (`#!`) does the trick but it's definitely bad practice. – Neel Apr 17, 2014 at 0:23

355

I would honestly suggest neither. I would use a stylized `<button></button>` for that behavior.

```
button.link {
  display: inline-block;
  position: relative;
  background-color: transparent;
  cursor: pointer;
  border: 0;
  padding: 0;
  color: #00f;
  text-decoration: underline;
  font: inherit;
}
```

```
<p>A button that looks like a <button type="button" class="link">link</button>.
</p>
```

Run code snippet Expand snippet

This way you can assign your onclick. I also suggest binding via script, not using the `onclick` attribute on the element tag. The only gotcha is the pseudo 3d text effect in older IEs that cannot be disabled.

If you *MUST* use an A element, use `javascript:void(0);` for reasons already mentioned.

- Will always intercept in case your onclick event fails.
- Will not have errant load calls happen, or trigger other events based on a hash change
- The hash tag can cause unexpected behavior if the click falls through (onclick throws), avoid it unless it's an appropriate fall-through behavior, and you want to change the navigation history.

NOTE: You can replace the `0` with a string such as `javascript:void('Delete record 123')` which can serve as an extra indicator that will show what the click will actually do.

Share Follow

edited Dec 14 at 12:19

community wiki
6 revs, 5 users 57%
Tracker1

54 This answer should be up top. If you are having this dilemma, chances are you are actually in need of a `button`. And IE9 is quickly losing market shares, fortunately, and the 1px active effect should not prevent us to use semantic markup. `<a>` is a link, it's meant to send you somewhere, for actions we have `<button>`'s. – mjsarfatti Mar 14, 2016 at 15:00

If the action in the onclick will act as a link (navigate to other page or open something in new window) then it should be semantically be an `<a>` tag. If the action is some in-page popup or some other functionality on same page, it should be a `<button>`. – awe Sep 29, 2023 at 7:52

I will respectfully disagree with @awe in that it should depend on default and fallback behavior, and especially destructive behaviors. Anything that should not have a fallback to a url should not be an anchor. Anything potentially destructive to data should *NEVER* be an anchor. – Tracker1 Apr 26 at 19:33

The first one, ideally with a real link to follow in case the user has JavaScript disabled. Just make sure to return false to prevent the click event from firing if the JavaScript

153 executes.

```
<a href="#" onclick="myJsFunc(); return false;">Link</a>
```

If you use Angular2, this way works:

`<a [routerLink]="" (click)="passTheSalt()">Click me</a>`.

See here https://stackoverflow.com/a/45465728/2803344

Share Follow

edited Oct 12, 2017 at 12:00

community wiki
3 revs, 2 users 55%
Belter

20 So in user agents with JavaScript enabled and the function supported this run a JavaScript function, falling back (for user agents where the JS fails for whatever reason) to a link to the top of the page? This is rarely a sensible fallback. – Quentin Sep 26, 2008 at 8:09

26 "ideally with a real link to follow in case the user has JavaScript disabled", it should be going to a useful page not #, even if it's just an explanation that the site needs JS to work. as for failing, I would expect the developer to use proper browser feature detection, etc before deploying. – Zach Sep 26, 2008 at 15:41

110

Neither if you ask me;

If your "link" has the sole purpose of running some JavaScript code it doesn't qualify as a link; rather a piece of text with a JavaScript function coupled to it. I would recommend to use a `<span>` tag with an `onclick handler` attached to it and some basic CSS to immitate a link. Links are made for navigation, and if your JavaScript code isn't for navigation it should not be an `<a>` tag.

Example:

```
function callFunction() { console.log("function called"); }
```

```
.jsAction {
    cursor: pointer;
    color: #00f;
    text-decoration: underline;
}
```

```
<p>I want to call a JavaScript function <span class="jsAction"
onclick="callFunction();">here</span>.</p>
```

Run code snippet   Expand snippet

Share Follow   edited Nov 5, 2017 at 13:52   community wiki
4 revs, 4 users 52%
fijter

139 This approach restricts the 'link' to a mouse only operation. An anchor can be visited via the keyboard and its 'onclick' event is fired when the enter key is pressed. – AnthonyWJones Sep 26, 2008 at 7:09

46 Hardcoding colors in your CSS would prevent the browser from using custom colors the user may define, which can be a problem with accessibility. – Hosam Aly Feb 28, 2009 at 19:21

40 `<span>` s are not meant to *do* anything. `<A>` nchors and `<buttons>` are used for that! – redShadow Dec 14, 2011 at 23:15

30 Using `buttons` is a better choice here while using a `span` is not. – apnerve Dec 28, 2012 at 8:33

1 *“Links are made for navigation, and if your JavaScript code isn't for navigation it should not be an* `<a>` *tag.”*—Even if the JavaScript code is “for navigation”, a link may not be appropriate every time. – Sebastian Simon May 29, 2021 at 1:04

108

Ideally you'd do this:

```
<a href="javascriptlessDestination.html" onclick="myJSFunc(); return
false;">Link text</a>
```

Or, even better, you'd have the default action link in the HTML, and you'd add the onclick event to the element unobtrusively via JavaScript after the DOM renders, thus ensuring that if JavaScript is not present/utilized you don't have useless event handlers riddling your code and potentially obfuscating (or at least distracting from) your actual content.

Share Follow   edited May 31, 2011 at 8:04   community wiki
2 revs, 2 users 80%
Steve Paulo

79

Using just `#` makes some funny movements, so I would recommend to use `#self` if you would like to save on typing efforts of `JavaScript bla, bla,`.

Share Follow   edited Nov 10, 2015 at 4:10   community wiki
3 revs, 3 users 40%
Spammer Joe

36 For reference, `#self` doesn't appear to be special. Any fragment identifier that doesn't match the name or id of any element in the document (and isn't blank or "top") should have the same effect. – cHao May 17, 2013 at 16:16

has the same effect, as cHao said. You just need to return false on element `a` when clicked (`onclick`) – Kafaa Billahi Syahida May 19, 2022 at 22:20

73

I use the following

```
<a href="javascript:;" onclick="myJsFunc();">Link</a>
```

instead

```
<a href="javascript:void(0);" onclick="myJsFunc();">Link</a>
```

Share Follow

edited Aug 9, 2018 at 11:14

community wiki
4 revs, 4 users 70%
se_pavel

`href="javascript:"` and `href="javascript:void 0;"` are all equivalent as well, but they're all equally bad. – Sebastian Simon May 29, 2021 at 1:10

This answer offers no justification whatsoever. Bald assertions aren't helpful. – isherwood Jan 19, 2023 at 17:59

71

I recommend using a `<button>` element instead, *especially* if the control is supposed to produce a change in the data. (Something like a POST.)

It's even better if you inject the elements unobtrusively, a type of progressive enhancement. (See this comment.)

Share Follow

edited Nov 29, 2019 at 8:14

community wiki
3 revs, 2 users 93%
treat your mods well

this depend if you need button on your bem (block element model) of html. – user8331407 Jul 5, 2021 at 8:43

I agree with suggestions elsewhere stating that you should use regular URL in `href` attribute, then call some JavaScript function in onclick. The flaw is, that they automaticaly add `return false` after the call.

59

The problem with this approach is, that if the function will not work or if there will be any problem, the link will become unclickable. Onclick event will always return `false`, so the normal URL will not be called.

There's very simple solution. Let function return `true` if it works correctly. Then use the returned value to determine if the click should be cancelled or not:

JavaScript

```
function doSomething() {
    alert( 'you clicked on the link' );
    return true;
}
```

HTML

```
<a href="path/to/some/url" onclick="return !doSomething();">link text</a>
```

Note, that I negate the result of the `doSomething()` function. If it works, it will return `true`, so it will be negated (`false`) and the `path/to/some/URL` will not be called. If the function will return `false` (for example, the browser doesn't support something used within the function or anything else goes wrong), it is negated to `true` and the `path/to/some/URL` is called.

Share Follow

edited Mar 11, 2012 at 8:36

community wiki
2 revs, 2 users 93%
Fczbkk

56

`#` is better than `javascript:anything`, but the following is even better:

HTML:

```
<a href="/gracefully/degrading/url/with/same/functionality.ext" class="some-
selector">For great justice</a>
```

JavaScript:

```
$(function() {
    $(".some-selector").click(myJsFunc);
});
```

You should always strive for graceful degradation (in the event that the user doesn't have JavaScript enabled...and when it is with specs. and budget). Also, it is considered bad form to use JavaScript attributes and protocol directly in HTML.

Share Follow

answered Nov 23, 2009 at 20:38

community wiki
Justin Johnson

2 @Muhd: Return should activate `click` on links… – Ry- ♦ Apr 10, 2014 at 23:47

47

Unless you're writing out the link using JavaScript (so that you know it's enabled in the browser), you should ideally be providing a proper link for people who are browsing with JavaScript disabled and then prevent the default action of the link in your onclick event handler. This way those with JavaScript enabled will run the function and those with JavaScript disabled will jump to an appropriate page (or location within the same page) rather than just clicking on the link and having nothing happen.

Share Follow

answered Sep 25, 2008 at 18:02

community wiki
Simon Forrest

40

Definitely hash (`#`) is better because in JavaScript it is a pseudoscheme:

1. pollutes history
2. instantiates new copy of engine
3. runs in global scope and doesn't respect event system.

Of course "#" with an onclick handler which prevents default action is [much] better. Moreover, a link that has the sole purpose to run JavaScript is not really "a link" unless you are sending user to some sensible anchor on the page (just # will send to top) when something goes wrong. You can simply simulate look and feel of link with stylesheet and forget about href at all.

In addition, regarding cowgod's suggestion, particularly this: `...href="javascript_required.html" onclick="...` This is good approach, but it doesn't distinguish between "JavaScript disabled" and "onclick fails" scenarios.

Share Follow

edited Feb 20, 2017 at 6:52

community wiki
5 revs, 4 users 45%
Free Consulting

33

I usually go for

```
<a href="javascript:;" onclick="yourFunction()">Link description</a>
```

It's shorter than javascript:void(0) and does the same.

Share Follow answered Aug 13, 2015 at 1:01 community wiki dnetix

30

I choose use `javascript:void(0)`, because use this could prevent right click to open the content menu. But `javascript:;` is shorter and does the same thing.

Share Follow edited Aug 9, 2018 at 11:16 community wiki 3 revs, 3 users 40% user564706

29

I would use:

```
<a href="#" onclick="myJsFunc();return false;">Link</a>
```

Reasons:

1. This makes the `href` simple, search engines need it. If you use anything else (such as a string), it may cause a `404 not found` error.
2. When mouse hovers over the link, it doesn't show that it is a script.
3. By using `return false;`, the page doesn't jump to the top or break the `back` button.

Share Follow edited Aug 22, 2013 at 11:37 community wiki 2 revs, 2 users 73% Eric Yin

i dont agree with "1." cause it gives error when u put ur script link when scripts are not allowed. so that kind of links should be added with the js code. that way people can avoid those links while script is not allowed and see no errors at all. – Berker Yüceer Dec 30, 2011 at 11:32

25

Don't use links for the sole purpose of running JavaScript.

The use of href="#" scrolls the page to the top; the use of void(0) creates navigational problems within the browser.

Instead, use an element other than a link:

```
<span onclick="myJsFunc()" class="funcActuator">myJsFunc</span>
```

And style it with CSS:

```
.funcActuator {
  cursor: default;
}

.funcActuator:hover {
  color: #900;
}
```

Share Follow

answered Jan 26, 2016 at 19:54

community wiki
Garrett

21 Use a button, not a span. Buttons naturally fall in the focus order so can be accessed without a mouse / trackpad / etc. – Quentin Jun 15, 2016 at 12:12

5 Adding ton Quentin's comment: as currently written, keyboard users won't reach the `span` element because it is a non-focusable element. That's why you need a button. – Tsundoku Jul 25, 2017 at 13:09

3 You can make it focusable by adding `tabindex="0"` to the span. That said, using button is better because it gives you the desired functionality for free. To make it accessible using a span you not only need to attach a click handler, but a keyboard event handler that looks for presses of space bar or enter key and then fires the normal click handler. You would also want to change the second CSS selector to `.funcActuator:hover, .funcActuator:focus` so the fact that the element has focus is apparent. – Useless Code Nov 22, 2017 at 11:45

24

So, when you are doing some JavaScript things with an `<a />` tag and if you put `href="#"` as well, you can add **return false** at the end of the event *(in case of inline event binding)* like:

```
<a href="#" onclick="myJsFunc(); return false;">Run JavaScript Code</a>
```

*Or you can change the **href** attribute with JavaScript like:*

```
<a href="javascript://" onclick="myJsFunc();">Run JavaScript Code</a>
```

or

```
<a href="javascript:void(0)" onclick="myJsFunc();">Run JavaScript Code</a>
```

But semantically, all the above ways to achieve this are wrong *(it works fine though)*. If any element is not created to navigate the page and that have some JavaScript things associated with it, then it should not be a `<a>` tag.

You can simply use a `<button />` instead to do things or any other element like b, span or whatever fits there as per your need, because you are allowed to add events on all the elements.

So, **there is one benefit** to use `<a href="#">`. You get the cursor pointer by default on that element when you do `a href="#"`. For that, I think you can use CSS for this like `cursor:pointer;` which solves this problem also.

And at the end, if you are binding the event from the JavaScript code itself, there you can do `event.preventDefault()` to achieve this if you are using `<a>` tag, but if you are not using a `<a>` tag for this, there you get an advantage, you don't need to do this.

So, if you see, it's better not to use a tag for this kind of stuff.

Share Follow edited Oct 26, 2014 at 12:32 community wiki
2 revs, 2 users 84%
Ashish Kumar

22

Usually, you should always have a fallback link to make sure that clients with JavaScript disabled still have some functionality. This concept is called unobtrusive JavaScript.

Example... Let's say you have the following search link:

```
<a href="search.php" id="searchLink">Search</a>
```

You can always do the following:

```
var link = document.getElementById('searchLink');

link.onclick = function() {
    try {
        // Do Stuff Here
    } finally {
        return false;
    }
};
```

That way, people with JavaScript disabled are directed to `search.php` while your viewers with JavaScript view your enhanced functionality.

Share Follow edited May 28, 2023 at 19:13 community wiki

4 revs, 4 users 84%
Andrew Moore

21

It would be better to use jQuery,

```
$(document).ready(function() {
    $("a").css("cursor", "pointer");
});
```

and omit both `href="#"` and `href="javascript:void(0)"`.

The anchor tag markup will be like

```
<a onclick="hello()">Hello</a>
```

Simple enough!

Share Follow

edited Mar 9, 2014 at 10:43

community wiki
3 revs, 3 users 88%
naveen

5 This is what I was going to say. If a link has a fallback url that makes sense, use that. Otherwise, just omit the href or use something more semantically appropriate than an <a>. If the only reason everyone is advocating including the href is to get the finger on hover, a simple "a { cursor: pointer; }" will do the trick. – Matt Kantor Jul 22, 2009 at 13:40

May I say this is the option that SO decided to go with. Check the "flag" links, for instance. – mtyaka Nov 24, 2009 at 10:19

13 That gives terrible accessibility. Try it in SO: you can't flag a post without using the mouse. The "link" and "edit" links are accessible by tabbing, but "flag" isn't. – Nicolás Jul 7, 2010 at 3:51

6 I agree with this option. If the anchor has no purpose other than JavaScript, it shouldn't have a href. @Fatih: Using jQuery means that if JavaScript is disabled, the link will NOT have a pointer. – Scott Rippey Jun 17, 2011 at 19:32

3 If you are going to go this route, why not bind the click using jQuery as well? Part of the great thing about using jQuery is the ability to seperate your javascript from your markup. – Muhd Dec 14, 2011 at 22:39

19

If you happen to be using **AngularJS**, you can use the following:

```
<a href="">Do some fancy JavaScript</a>
```

Which will not do anything.

In addition

- It will not take you to the top of the page, as with (#)
    - Therefore, you don't need to explicitly return `false` with JavaScript
- It is short an concise

Share Follow edited Jul 30, 2013 at 10:45 community wiki
2 revs
whirlwin

6 But this would cause the page to reload, and since we're always using javascript to modify the page, this is unacceptable. – Henry Hu Jul 30, 2013 at 0:16

@HenryHu I figured out that the reason it did not reload was because of AngularJS. See my updated answer. – whirlwin Jul 30, 2013 at 10:46

18

Depending on what you want to accomplish, you could forget the onclick and just use the href:

```
<a href="javascript:myJsFunc()">Link Text</a>
```

It gets around the need to return false. I don't like the `#` option because, as mentioned, it will take the user to the top of the page. If you have somewhere else to send the user if they don't have JavaScript enabled (which is rare where I work, but a very good idea), then Steve's proposed method works great.

```
<a href="javascriptlessDestination.html" onclick="myJSFunc(); return
false;">Link text</a>
```

Lastly, you can use `javascript:void(0)` if you do not want anyone to go anywhere and if you don't want to call a JavaScript function. It works great if you have an image you want a mouseover event to happen with, but there's not anything for the user to click on.

Share Follow edited Mar 11, 2012 at 8:57 community wiki
3 revs, 3 users 80%
Will Read

3 The only downside with this (from memory, I may be wrong) is that IE doesn't consider an A to be an A if you don't have a href inside it. (So CSS rules won't work) – Benjol Sep 26, 2008 at 12:01

16

I personally use them in combination. For example:

HTML

```
<a href="#">Link</a>
```

---

with little bit of jQuery

```
$('a[href="#"]').attr('href','javascript:void(0);');
```

or

```
$('a[href="#"]').click(function(e) {
   e.preventDefault();
});
```

But I'm using that just for preventing the page jumping to the top when the user clicks on an empty anchor. I'm rarely using onClick and other `on` events directly in HTML.

My suggestion would be to use `<span>` element with the `class` attribute instead of an anchor. For example:

```
<span class="link">Link</span>
```

Then assign the function to `.link` with a script wrapped in the body and just before the `</body>` tag or in an external JavaScript document.

```
<script>
    (function($) {
        $('.link').click(function() {
            // do something
        });
    })(jQuery);
</script>
```

***Note:** For dynamically created elements, use:

```
$('.link').on('click', function() {
    // do something
});
```

And for dynamically created elements which are created with dynamically created elements, use:

```
$(document).on('click','.link', function() {
    // do something
});
```

Then you can style the span element to look like an anchor with a little CSS:

```
.link {
    color: #0000ee;
    text-decoration: underline;
    cursor: pointer;
}
.link:active {
    color: red;
}
```

Here's **a jsFiddle** example of above aforementioned.

Share Follow

edited Jan 24, 2019 at 23:25

community wiki
4 revs, 3 users 82%
mdesdev

16

I believe you are presenting a false dichotomy. These are not the only two options.

I agree with Mr. D4V360 who suggested that, even though you are using the anchor tag, you do not truly have an anchor here. All you have is a special section of a document that should behave slightly differently. A `<span>` tag is far more appropriate.

Share Follow

edited May 28, 2023 at 19:16

community wiki
3 revs, 3 users 67%
CleverPatrick

1 Also if you were to replace an `a` with a `span`, you'll need to remember to make it focusable via keyboard. – AndFisher Jan 18, 2017 at 14:42

1 No, a span should not be used. There's an element specifically for things that are clickable but not links, and that is `<button>`. – Sean Mar 20 at 21:05

15

When I've got several faux-links, I prefer to give them a class of 'no-link'.

Then in jQuery, I add the following code:

```
$(function(){
   $('.no-link').click(function(e){
       e.preventDefault();
   });
});
```

And for the HTML, the link is simply

```
<a href="/" class="no-link">Faux-Link</a>
```

I don't like using Hash-Tags unless they're used for anchors, and I only do the above when I've got more than two faux-links, otherwise I go with javascript:void(0).

```
<a href="javascript:void(0)" class="no-link">Faux-Link</a>
```

Typically, I like to just avoid using a link at all and just wrap something around in a span and use that as a way to active some JavaScript code, like a pop-up or a content-reveal.

Share Follow edited Oct 26, 2014 at 12:28 community wiki
2 revs, 2 users 88%
Stacks on Stacks on Stacks

15

I tried both in google chrome with the developer tools, and the `id="#"` took 0.32 seconds. While the `javascript:void(0)` method took only 0.18 seconds. So in google chrome, `javascript:void(0)` works better and faster.

Share Follow edited Feb 20, 2017 at 15:46 community wiki
2 revs, 2 users 67%
user6460587

Actually they don't do the same. # makes you jump to top of the page. – Jochen Schultz Sep 21, 2017 at 10:58

15

On a modern website the use of href should be avoided if the element is only doing JavaScript functionality (not a real link).

Why? The presence of this element tells the browser that this is a link with a destination. With that, the browser will show the Open In New Tab / Window function (also triggered when you use shift+click). Doing so will result in opening the same page without the desired function triggered (resulting in user frustration).

In regards to IE: As of IE8, element styling (including hover) works if the doctype is set. Other versions of IE are not really to worry about anymore.

Only Drawback: Removing HREF removes the tabindex. To overcome this, you can use a button that's styled as a link or add a tabindex attribute using JS.

Share Follow

edited Feb 6, 2019 at 1:32

community wiki
2 revs
djpalme

14

It's nice to have your site be accessible by users with JavaScript disabled, in which case the href points to a page that performs the same action as the JavaScript being executed. Otherwise I use "#" with a "`return false;`" to prevent the default action (scroll to top of the page) as others have mentioned.

Googling for "`javascript:void(0)`" provides a lot of information on this topic. Some of them, like this one mention reasons to **NOT use void(0)**.

Share Follow

edited Aug 15, 2018 at 9:38

community wiki
4 revs, 4 users 42%
Peter Mortensen

2 The blog entry does not cite the reference as to why javascript:void(0) should be avoided. – AnthonyWJones Sep 26, 2008 at 7:20

3 The link is (effectively) broken now. – Peter Mortensen May 31, 2011 at 8:06

2 Not good solution for accessibility: dequeuniversity.com/rules/axe/3.0/href-no-hash – Hrvoje Golcic Nov 26, 2019 at 15:13

`javascript:void(0);` is also not good if you want to use a strict Content Security Policy that disables inline JavaScript. – Shannon Scott Schupbach Apr 7, 2020 at 18:59

1 2 Next

Highly active question. Earn 10 reputation (not counting the association bonus) in order to answer this question. The reputation requirement helps protect this question from spam and non-answer activity.