What is the best way to add an event in JavaScript?

Asked 16 years, 3 months ago Modified 8 years, 10 months ago Viewed 11k times

9

I see 2 main ways to set events in JavaScript:

1. Add an event directly inside the tag like this:

   ```
   <a href="" onclick="doFoo()">do foo</a>
   ```

2. Set them by JavaScript like this:

   ```
   <a id="bar" href="">do bar</a>
   ```

and add an event in a `<script>` section inside the `<head>` section or in an external JavaScript file, like that if you're using **prototypeJS**:

```
Event.observe(window, 'load', function() {
    $('bar').observe('click', doBar);
}
```

I think the first method is easier to read and maintain (because the JavaScript action is directly bound to the link) but it's not so clean (because users can click on the link even if the page is not fully loaded, which may cause JavaScript errors in some cases).

The second method is cleaner (actions are added when the page is fully loaded) but it's more difficult to know that an action is linked to the tag.

Which method is the best?

A killer answer will be fully appreciated!

javascript html events event-binding

Share
Improve this question
Follow

edited Jun 1, 2013 at 19:08

Erik Schierboom
16.6k ● 10 ● 66 ● 81

asked Aug 29, 2008 at 7:35

paulgreg
18.9k ● 18 ● 48 ● 56

6 Answers

Sorted by: Highest score (default)

9

> I think the first method is easier to read and maintain

I've found the opposite to be true. Bear in mind that sometimes more than one event handler will be bound to a given control.

Declaring all events in one central place helps to organize the actions taking place on the site. If you need to change something you don't have to search for all places making a call to a function, you simply have to change it in one place. When adding more elements that should have the same functionality you don't have to remember to add the handlers to them; instead, it's often enough to let them declare a class, or even not change them at all because they logically belong to a container element of which all child elements get wired to an action. From an actual code:

```
$$('#itemlist table th > a').invoke('observe', 'click', performSort);
```

This wired an event handler to all column headers in a table to make the table sortable. Imagine the effort to make all column headers sortable separately.

Share Improve this answer Follow

answered Aug 29, 2008 at 7:50

Konrad Rudolph
545k ● 139 ● 956 ● 1.2k

4

In my experience, there are two major points to this:

1) The most important thing is to be consistent. I don't think either of the two methods is necessarily easier to read, as long as you stick to it. I only get confused when both methods are used in a project (or even worse on the same page) because then I have to start searching for the calls and don't immediately know where to look.

2) The second kind, i.e. `Event.observe()` has advantages when the same or a very similar action is taken on multiple events because this becomes obvious when all those calls are in the same place. Also, as Konrad pointed out, in some cases this can be handled with a single call.

Share
Improve this answer
Follow

edited Feb 8, 2016 at 14:36

Alexander Elgin
6,946 ● 4 ● 42 ● 52

answered Aug 29, 2008 at 8:10

Thomas Lötzer
25.4k ● 16 ● 70 ● 56

I believe the second method is generally preferred because it keeps information about action (i.e. the JavaScript) separate from the markup in the same way CSS separates presentation from markup.

3

I agree that this makes it a little more difficult to see what's happening in your page, but good tools like firebug will help you with this a lot. You'll also find much better IDE support available if you keep the mixing of HTML and Javascript to a minimum.

This approach really comes into its own as your project grows, and you find you want to attach the same javascript event to a bunch of different element types on many different pages. In that case, it becomes much easier to have a single pace which attaches events, rather than having to search many different HTML files to find where a particular function is called.

Share Improve this answer Follow

answered Aug 29, 2008 at 9:05

Matt Sheppard

118k ● 46 ● 113 ● 134

1

You can also use addEventListener (not in IE) / attachEvent (in IE).

Check out: http://www.quirksmode.org/js/events_advanced.html

These allow you to attach a function (or multiple functions) to an event on an existing DOM object. They also have the advantage of allowing un-attachment later.

In general, if you're using a serious amount of javascript, it can be useful to make your *javascript* readable, as opposed to your html. So you could say that `onclick=X` in the html is very clear, but this is both a lack of separation of the code -- another syntactic dependency between pieces -- and a case in which you have to read both the html and the javascript to understand the dynamic behavior of the page.

Share

Improve this answer

Follow

edited Feb 8, 2016 at 14:37

Alexander Elgin

6,946 ● 4 ● 42 ● 52

answered Aug 29, 2008 at 8:29

Tyler

28.9k ● 12 ● 93 ● 108

0

My personal preference is to use jQuery in external js files so the js is completely separate from the html. Javascript should be unobtrusive so inline (ie, the first example) is not really the best choice in my opinion. When looking at the html, the only sign that you are using js should be the script includes in the head.

An example of attaching (and handling) events might be something like this

```
var myObject = {

    allLinkElements: null,

    init: function()
    {
```

```
        // Set all the elements we need
        myObject.setElements();

        // Set event handlers for elements
        myObject.setEventHandlers();
    },

    clickedLink: function()
    {
        // Handle the click event
        alert('you clicked a link');
    },

    setElements: function()
    {
        // Find all <a> tags on the page
        myObject.allLinkElements = $('a');

        // Find other elements...
    },

    setEventHandlers: function()
    {
        // Loop through each link
        myObject.allLinkElements.each(function(id)
        {
            // Assign the handler for the click event
            $(this).click(myObject.clickedLink);
        });

        // Assign handlers for other elements...
    }
}

// Wait for the DOM to be ready before initialising
$(document).ready(myObject.init);
```

I think this approach is useful if you want to keep all of your js organised, as you can use specific objects for tasks and everything is nicely contained.

Of course, the huge benefit of letting jQuery (or another well known library) do the hard work is that cross-browser support is (largely) taken care of which makes life much easier

Share Improve this answer Follow

answered Aug 29, 2008 at 9:56

harry.w

0

Libraries like YUI and jQuery provide methods to add events only once the DOM is ready, which can be before window.onload. They also ensure that you can add multiple event handlers so that you can use scripts from different sources without the different event handlers overwriting each other.

So your practical choices are;

One. If your script is simple and the only one that will ever run on the page, create an init function like so:

```
window.onload = function () {
    init();
}
function init() {
    // actual function calls go here
    doFoo();
}
```

Two. If you have many scripts or plan to mashup scripts from different sources, use a library and its `onDOMReady` method to safely add your event handlers

Share

Improve this answer

Follow

edited Feb 8, 2016 at 14:38

Alexander Elgin

6,946 ● 4 ● 42 ● 52

answered Aug 29, 2008 at 7:41

Polsonby

22.9k ● 19 ● 60 ● 74