JavaScript(https://www.sitepoint.com/javascript/) - May 24, 2017 - By Giulio Mainardi (https://www.sitepoint.com/author/gmainardi/)

What Is Event Bubbling in JavaScript? Event Propagation Explained

Event bubbling is a term you might have come across on your JavaScript travels. It relates to the order in which event handlers are called when one element is nested inside a second element, and both elements have registered a listener for the same event (a click, for example).

But event bubbling is only one piece of the puzzle. It is often mentioned in conjunction with event capturing and event propagation. And a firm understanding of all three concepts is essential for working with events in JavaScript — for example if you wish to implement the event delegation pattern (https://www.sitepoint.com/javascript-event-delegation-is-easier-than-you-think/).

In this post I will explain each of these terms and demonstrate how they fit together. I will also show you how a basic understanding of JavaScript event flow can give you fine-grained control over your application. Please note that this is not a primer on events, thus a familiarity with the topic is assumed. If you'd like to learn more about events in general, why not check out our book: JavaScript: Novice to Ninja (https://www.sitepoint.com/premium/books/javascript-novice-to-ninja).

What is the Event Propagation?

Let's start with event propagation. This is the blanket term for both event bubbling and event capturing. Consider the typical markup to build a list of linked images, for a thumbnails gallery for example:

```
<ul>
    <li><a href="..."><img src="..." alt=""></a>
    <li><a href="..."><img src="..." alt=""></a>
    ...
    <li><a href="..."><img src="..." alt=""></a>
</ul>
```

A click on an image does not only generate a `click` event for the corresponding `IMG` element, but also for the parent `A`, for the grandfather `LI` and so on, going all the way up through all the element's ancestors, before terminating at the `window` object.

In DOM terminology, the image is the *event target*, the innermost element over which the click originated. The event target, plus its ancestors, from its parent up through to the `window` object, form a branch in the DOM tree. For example, in the image gallery, this branch will be composed of the nodes: `IMG`, `A`, `LI`, `UL`, `BODY`, `HTML`, `document`, `window`.

> Note that `window` is not actually a DOM node but it implements the `EventTarget` interface, so, for simplicity, we are handling it like it was the parent node of the document object.

This branch is important because it is the path along which the events propagate (or flow). This propagation is the process of calling all the listeners for the given event type, attached to the nodes on the branch. Each listener will be called with an `event` object that gathers information relevant to the event (more on this later).

Remember that several listeners can be registered on a node for the same event type. When the propagation reaches one such node, listeners are invoked in the order of their registration.

It should also be noted that the branch determination is static, that is, it is established at the initial dispatch of the event. Tree modifications occurring during event processing will be ignored.

The propagation is bidirectional, from the window to the event target and back. This propagation can be divided into three phases:

1 From the window to the event target parent: this is the *capture phase*
2 The event target itself: this is the *target phase*
3 From the event target parent back to the window: the *bubble phase*

What differentiates these phases is the type of listeners that are called.

The Event Capture Phase

In this phase only the *capturer* listeners are called, namely, those listeners that were registered using a value of `true` for the third parameter of addEventListener (https://developer.mozilla.org/en-US/docs/Web/API/EventTarget/addEventListener):

```
el.addEventListener('click', listener, true)
```

If this parameter is omitted, its default value is false and the listener is not a capturer.

So, during this phase, only the capturers found on the path from the window to the event target parent are called.

The Event Target Phase

In this phase all the listeners registered on the event target will be invoked, regardless of the value of their capture flag.

The Event Bubbling Phase

During the event bubbling phase only the non-capturers will be called. That is, only the listeners registered with a value of `false` for the third parameter of `addEventListener()`:

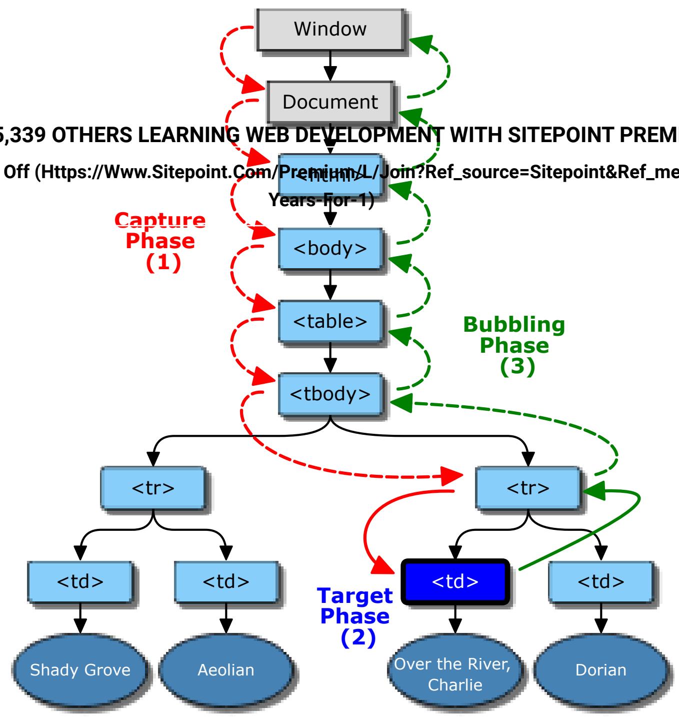
```
el.addEventListener('click', listener, false) // listener doesn't capture
el.addEventListener('click', listener) // listener doesn't capture
```

Note that while all events flow down to the event target with the capture phase, `focus`, `blur`, `load` and some others, don't bubble up. That is, their travel stops after the *target* phase.

Therefore, at the end of the propagation, each listener on the branch has been called exactly once.

Event bubbling does not take place for every kind of event. During propagation, it is possible for a listener to know if an event bubbles by reading the `.bubbles` Boolean property of the `event` object.

The three event flow phase are illustrated in the following diagram from the W3C UIEvents specification (https://www.w3.org/TR/DOM-Level-3-Events/).



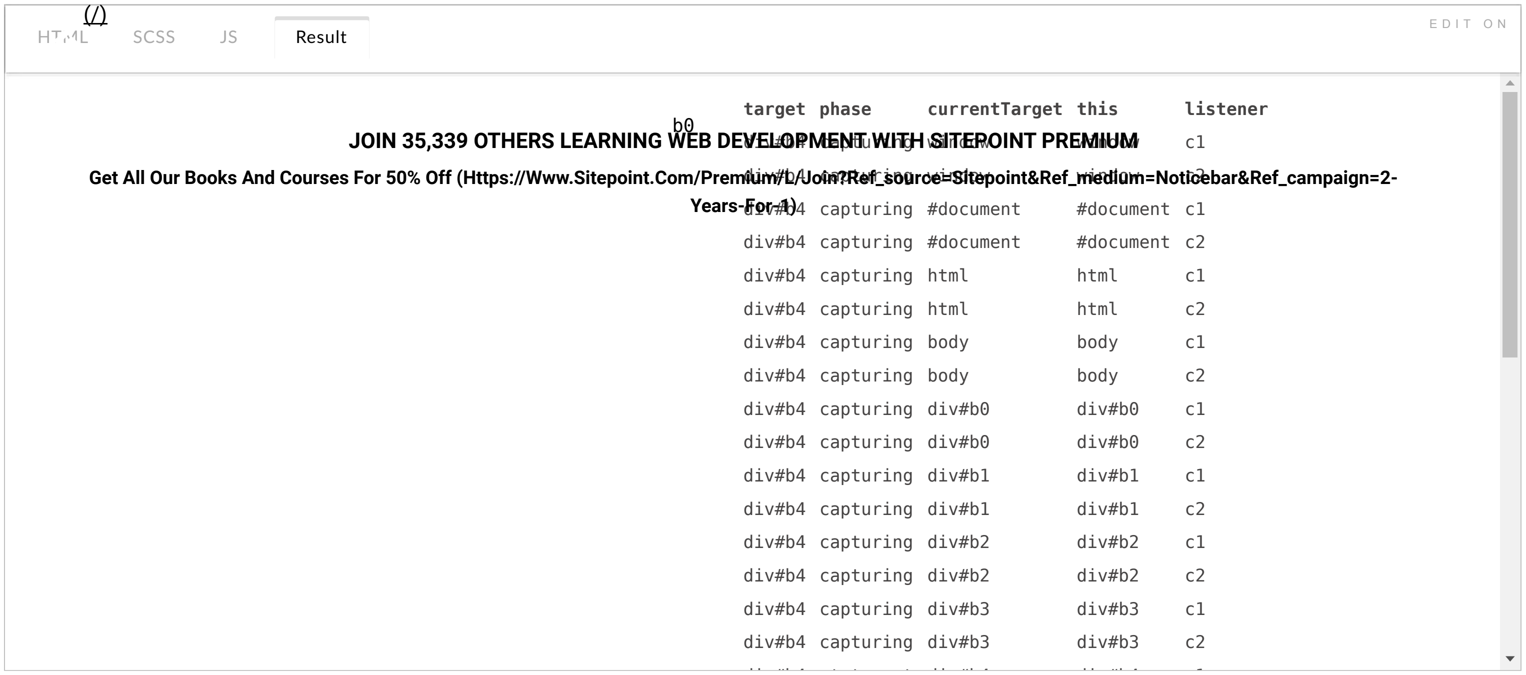
Accessing Propagation Information

I already mentioned the `.bubbles` property of the `event` object. There are a number of other properties provided by this object that are available to the listeners to access information relative to the propagation.

- e.target (https://developer.mozilla.org/en-US/docs/Web/API/Event/target) references the event target.
- e.currentTarget (https://developer.mozilla.org/en-US/docs/Web/API/Event/currentTarget) is the node on which the running listener was registered on. This is the same value of the listener invocation context, i.e, the value referenced by the `this` keyword.
- We can even find out the current phase with e.eventPhase (https://developer.mozilla.org/en-US/docs/Web/API/Event/eventPhase). It is an integer that refers to one the three `Event` constructor constants `CAPTURING_PHASE`, `BUBBLING_PHASE` and `AT_TARGET`.

Putting it into Practice

Let's see the above concepts into practice. In the following pen, there are five nested square boxes, named `b0`…`b4`. Initially, only the outer box `b0` is visible; the inner ones will show when the mouse pointer hovers over them. When we click on a box, a log of the propagation flow is shown on the table to the right.



```
target  phase      currentTarget  this       listener
div#b4  capturing  window         window     c1
div#b4  capturing  window         window     c2
div#b4  capturing  #document      #document  c1
div#b4  capturing  #document      #document  c2
div#b4  capturing  html           html       c1
div#b4  capturing  html           html       c2
div#b4  capturing  body           body       c1
div#b4  capturing  body           body       c2
div#b4  capturing  div#b0         div#b0     c1
div#b4  capturing  div#b0         div#b0     c2
div#b4  capturing  div#b1         div#b1     c1
div#b4  capturing  div#b1         div#b1     c2
div#b4  capturing  div#b2         div#b2     c1
div#b4  capturing  div#b2         div#b2     c2
div#b4  capturing  div#b3         div#b3     c1
div#b4  capturing  div#b3         div#b3     c2
```

It is even possible to click outside the boxes: in this case, the event target will be the `BODY` or the `HTML` element, depending on the click screen location.

Stopping Propagation

The event propagation can be stopped in any listener by invoking the stopPropagation (https://developer.mozilla.org/en-US/docs/Web/API/Event/stopPropagation) method of the event object. This means that all the listeners registered on the nodes on the propagation path that follow the current target will not be called. Instead, all the other remaining listeners attached on the current target will still receive the event.

We can check this behavior with a simple fork of the previous demo (https://codepen.io/SitePoint/pen/aWPXQg), just inserting a call to `stopPropagation()` in one of the listeners. Here we have prepended this new listener as a capturer to the list of callbacks registered on `window`:

```
 window.addEventListener('click', e => { e.stopPropagation(); }, true);
window.addEventListener('click', listener('c1'), true);
window.addEventListener('click', listener('c2'), true);
window.addEventListener('click', listener('b1'));
window.addEventListener('click', listener('b2'));
```

This way, whatever box is clicked, the propagation halts early, reaching only the capturer listeners on `window`.

Stopping Immediate Propagation

As indicated by its name, stopImmediatePropagation (https://developer.mozilla.org/en-US/docs/Web/API/Event/stopImmediatePropagation) throws the brakes on straight away, preventing even the siblings of the current listener from receiving the event. We can see this with a minimal change to the last pen (https://codepen.io/SitePoint/pen/jmXddK):

```
 window.addEventListener('click', e => { e.stopImmediatePropagation(); }, true);
window.addEventListener('click', listener('c1'), true);
window.addEventListener('click', listener('c2'), true);
window.addEventListener('click', listener('b1'));
window.addEventListener('click', listener('b2'));
```

Now, nothing is output in the log table, neither the `c1` and `c2` window capturers rows, because the propagation stops after the execution of the new listener.

Event Cancellation

Some events are associated with a default action that the browser executes at the end of the propagation. For instance, the click on a link element or the click on a form submit button causes the browser to navigate to a new page, or submit the form respectively.

It is possible to avoid the execution of such default actions with the event cancellation, by calling yet another method of the event object, e.preventDefault (https://developer.mozilla.org/en-US/docs/Web/API/Event/preventDefault), in a listener.

Conclusion

With that, I hope to have shown you how event bubbling and event capturing work in JavaScript. If you have any questions or comments, I'd be glad to hear them in the discussion below.

References

Document Object Model (DOM) Level 2 Events Specification (https://www.w3.org/TR/DOM-Level-2-Events/events.html#Events-flow)
W3C DOM4 – Events (https://www.w3.org/TR/dom/#events)
DOM – Living Standard – Events (https://dom.spec.whatwg.org/#events)
W3C UI Events – DOM Event Architecture (https://www.w3.org/TR/DOM-Level-3-Events/#dom-event-architecture)
MSN – Events and the DOM (https://developer.mozilla.org/en-US/docs/Web/API/Document_Object_Model/Events)

This article was peer reviewed by Yaphi Berhanu (https://www.sitepoint.com/author/yberhanu) and Dominic Myers (https://github.com/annoyingmouse). Thanks to all of SitePoint's peer reviewers for making SitePoint content the best it can be!

Meet the author
Giulio Mainardi (https://www.sitepoint.com/author/gmainardi/) (https://twitter.com/mgiulio) (https://www.facebook.com/giulio.mainardi.1) (http://codepen.io/mgiulio/) (https://github.com/mgiulio)

Dabbling in front end development.

(/)

Maxime Soulie 6d
Hello,Just after editing the package.json file,...

Surja 23 Aug
Thank you for writing this helpful article. G...

Keith 2 Sep
Excellent examples of the struggles of...

Dev Agrawal 9 Sep
Excellent guide... definitely needed thi...

reiallenramos... 2d
Good tutorial for Rails beginners. While this...

JOIN 35,339 OTHERS LEARNING WEB DEVELOPMENT WITH SITEPOINT PREMIUM

Get All Our Books And Courses For 50% Off (Https://Www.Sitepoint.Com/Premium/L/Join?Ref_source=Sitepoint&Ref_medium=Noticebar&Ref_campaign=2-Years-For-1)

Conversation (18)

Sort by **Oldest**

Add a comment...

Bruno Scopelliti
· 25 May
This was a nice intro!

Reply · Share ·

Giulio Mainardi Bruno Scopelliti
· 25 May
Thanks Bruno, good to hear you liked it.

Reply · Share ·

Patrick Van Bergen
· 25 May
Thanks, Giulio! Very insightful!

Reply · Share ·

Giulio Mainardi Patrick Van Bergen
· 26 May
Thank you Patrick, happy you found it useful!

Reply · Share ·

Pradeep Chavan
· 27 May
Thank You. Very Helpful.

Reply · Share ·

Giulio Mainardi Pradeep Chavan
· 27 May
:)

Reply · Share ·

Pradeep Chavan
· 27 May
How to generate auto drop down of a combobox(datalist) using the keyboard?

Reply · Share ·

Giulio Mainardi Pradeep Chavan
· 29 May
Sorry, this is a too broad question :)

Reply · Share ·

quazecoatl
· 28 May
The three event flow illustration, I wish you created it yourself based on the thumbnails gallery code example you gave in the previous section. It's confusing that you started with a code example but the event flow illustration actually refers to something else.

Reply · Share ·

Giulio Mainardi quazecoatl
· 29 May
Hello, thanks for your feedback.

The thumbnails snippet is only an initial example and after the introduction it is never used again. Even the demo pen doesn't refer to that markup.
The W3C diagram was included because it is a nice illustration of the event propagation flow.

Reply · Share ·

Daniele Tabanella
· 29 May
Nice reading! Do you have any "real world scenario" example when Capture is preferred over Bubbling ?

Reply · Share ·

(/)

Giulio Mainardi Daniele Tabanella
· 29 May
Some examples are presented on https://stackoverflow.com/questions/24786769/real-world-example-where-event-capturing-is-necessary-preferred

Reply · Share ·

Vladimir Mityukov
· 1 Jun
This statement requires elaboration, at least for me.. "all the listeners registered on the nodes on the propagation path that follow the current target will not be called. Instead, all the other remaining listeners attached on the current target will still receive the event."

What are those "other remaining listeners"? Without comprehending this one, I could not get immediate propagation either =)

Reply · Share ·

Giulio Mainardi Vladimir Mityukov
· 2 Jun
Hello Vladimir, thanks for your feedback.

Each node on the propagation path can have zero, one or more registered listeners.

Let's say that n1 and n2 are two consecutive nodes on the propagation path, that is, the event flows first in n1 and then through n2.
Suppose that n1 has the listeners l_1_1, l_1_2 and l_1_3, and that n2 has its own list of listeners too.

If l_1_1 invokes the e.stopPropagation() method, none of the listeners on n2 will be called, but l_1_2 and l_1_3("the 'remaining listeners attached on the current target") will be called.

See more

Reply · Share · *1 Like* ·

DimKa Klusevich
· 3 Jun
Cool. Thank You.

Reply · Share ·

Giulio Mainardi DimKa Klusevich
· 3 Jun
:)

Reply · Share ·

Jabar Sadiq Syed Ibrahim
· 3 Jun
nice explanation Guilio ..looks cool stuff

Reply · Share ·

Nilesh Ganu
· 28 Jun
Nice article.

Reply · Share ·

Terms · Privacy

Add Spot.IM to your site ·

Stuff We Do

- Premium (/premium/)
- Versioning (/versioning/)
- Themes (/themes/)
- Forums (/community/)
- References (/html-css/css/)

About

- Our Story (/about-us/)
- Press Room (/press/)

Contact

- Contact Us (/contact-us/)
- FAQ (https://sitepoint.zendesk.com/hc/en-us)
- Write for Us (/write-for-us/)
- Advertise (/advertise/)

Legals

- Terms of Use (/legals/)
- Privacy Policy (/legals/#privacy)