Event bubbling & Capturing Model

**addEventListener()**

The addEventListener() method is used to attach an event handler to a particular element. It does not override the existing event handlers. Events are said to be an essential part of the JavaScript. A web page responds according to the event that occurred. Events can be user-generated or generated by API's. An event listener is a JavaScript's procedure that waits for the occurrence of an event.

The addEventListener() method is an inbuilt function of JavaScript. We can add multiple event handlers to a particular element without overwriting the existing event handlers.

Syntax

element.addEventListener(event, function, useCapture);

Although it has three parameters, the parameters event and function are widely used. The third parameter is optional to define. The values of this function are defined as follows.

Parameter Values

1. event: It is a required parameter. It can be defined as a string that specifies the event's name.
2. function: It is also a required parameter. It is a JavaScript function which responds to the event occur.
3. useCapture: It is an optional parameter. It is a Boolean type value that specifies whether the event is executed in the bubbling or capturing phase. Its possible values are true and false. When it is set to true, the event handler executes in the capturing phase. When it is set to false, the handler executes in the bubbling phase. Its default value is false.

So, in **Bubbling**, the event of paragraph element is handled first, and then the div element's event is handled. It means that in bubbling, the inner element's event is handled first, and then the outermost element's event will be handled.

In **Capturing** the event of div element is handled first, and then the paragraph element's event is handled. It means that in capturing the outer element's event is handled first, and then the innermost element's event will be handled.

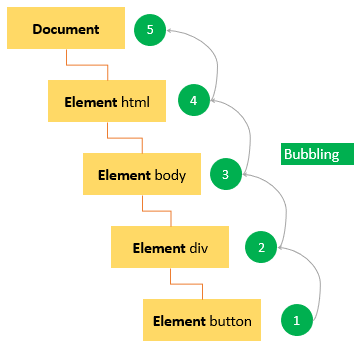
In the event bubbling model, an event starts at the most specific element and then flows upward toward the least specific element (the document or even [window](https://www.javascripttutorial.net/javascript-bom/javascript-window/)).

When you click the button, the click event occurs in the following order:

1. button
2. div with the id container
3. body
4. html
5. document

The click event first occurs on the button which is the element that was clicked. Then the click event goes up the DOM tree, firing on each node along its way until it reaches the document object.

The following picture illustrates the event bubbling effect when users click the button:

[](https://www.javascripttutorial.net/wp-content/uploads/2020/02/JavaScript-event-bubbling.png)

Note that modern web browsers bubble the event up to the window object.

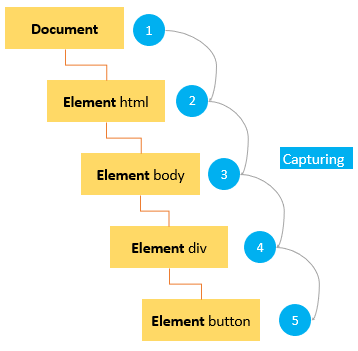
### Event capturing

In the event capturing model, an event starts at the least specific element and flows downward toward the most specific element.

When you click the button, the click event occurs in the following order:

1. document
2. html
3. body
4. div with the id container
5. button

The following picture illustrates the event capturing effect:

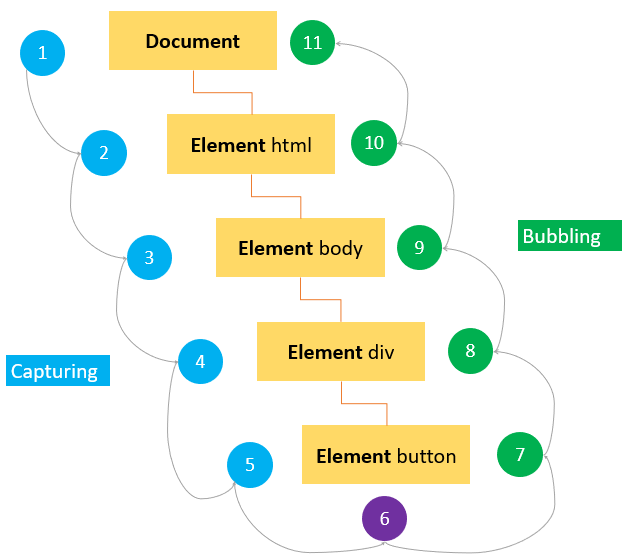
[](https://www.javascripttutorial.net/wp-content/uploads/2020/02/JavaScript-event-capturing.png)

### DOM Level 2 Event flow

DOM level 2 events specify that event flow has three phases:

* First, event capturing occurs, which provides the opportunity to intercept the event.
* Then, the actual target receives the event.
* Finally, event bubbling occurs, which allows a final response to the event.

The following picture illustrates the DOM Level 2 event model when users click the button:

[](https://www.javascripttutorial.net/wp-content/uploads/2020/02/JavaScript-DOM-Level-2-Event.png)

## **Event object**

When the event occurs, the web browser passed an Event object to the event handler:

let btn = document.querySelector('#btn');

btn.addEventListener('click', function(event) {

console.log(event.type);

});Code language: JavaScript (javascript)

Output:

'click'Code language: JavaScript (javascript)

The following table shows the most commonly-used properties and methods of the event object:

| **Property / Method** | **Description** |
| --- | --- |
| bubbles | true if the event bubbles |
| cancelable | true if the default behavior of the event can be canceled |
| currentTarget | the current element on which the event is firing |
| defaultPrevented | return true if the preventDefault() has been called. |
| detail | more informatio nabout the event |
| eventPhase | 1 for capturing phase, 2 for target, 3 for bubbling |
| preventDefault() | cancel the default behavior for the event. This method is only effective if the cancelable property is true |
| stopPropagation() | cancel any further event capturing or bubbling. This method only can be used if the bubbles property is true. |
| target | the target element of the event |
| type | the type of event that was fired |

Note that the event object is only accessible inside the event handler. Once all the event handlers have been executed, the event object is automatically destroyed.

### preventDefault()

To prevent the default behavior of an event, you use the preventDefault() method.

For example, when you click a link, the browser navigates you to the URL specified in the href attribute:

**<a href="https://www.javascripttutorial.net/">**JS Tutorial**</a>**Code language: HTML, XML (xml)

However, you can prevent this behavior by using the preventDefault() method of the event object:

let link = document.querySelector('a');

link.addEventListener('click',function(event) {

console.log('clicked');

event.preventDefault();

});Code language: JavaScript (javascript)

Note that the preventDefault() method does not stop the event from bubbling up the DOM. And an event can be canceled when its cancelable property is true.

### stopPropagation()

The stopPropagation() method immediately stops the flow of an event through the DOM tree. However, it does not stop the browers default behavior.

See the following example:

let btn = document.querySelector('#btn');

btn.addEventListener('click', function(event) {

console.log('The button was clicked!');

event.stopPropagation();

});

document.body.addEventListener('click',function(event) {

console.log('The body was clicked!');

});

Code language: JavaScript (javascript)

Without the stopPropagation() method, you would see two messages on the Console window.

However, the click event never reaches the body because the stopPropagation() was called on the click event handler of the button.