COURSEWARE

Handling Events and Timed Events

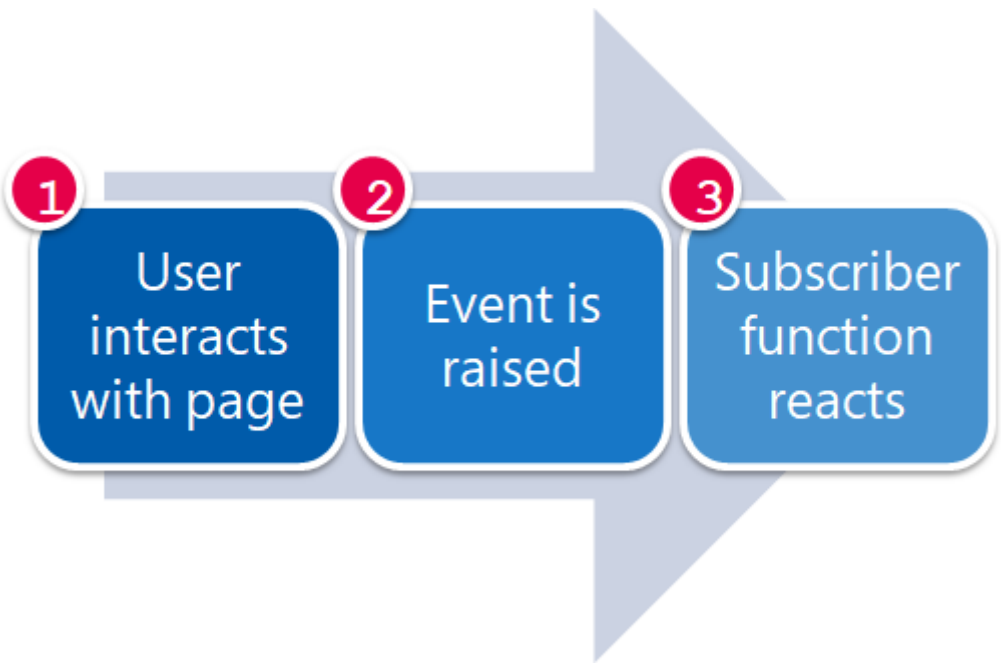
Contents

Overview

Events are the beating heart of any JavaScript page. Without events there are no scripts, take a look at any web page with JavaScript in it - in nearly all cases there will be an event that triggers the script. The reason is very simple. JavaScript is meant to add *interactivity* to our pages; the user does something and the page reacts. Therefore JavaScript needs a way of detecting user actions, so that it knows when to react. There are also some events that aren't directly caused by the user - the load event that fires when a page has been loaded, for instance.

Tutorial

The JavaScript event model has three stages.



The user (or browser event) occurs. This could be through a mouse clicking a button or a key being pressed. Every DOM object has a list of events it can execute (which we will investigate shortly).

An event may be subscribed to by associating the event to a function. When the event is raised the subscribed function executes.

There are three distinct event models which can be used in JavaScript:

Inline subscription model

The *inline subscription model* approach is quick, easy and works in all browsers.

Event handlers are added as attributes to the HTML element. The event is always on and always fires:

```
<button type="button" onclick="changeClass('container','div2');">
  Click Me
</button>
```

This approach is okay for testing but not recommended for release, as it will likely lead to hard to maintain and repetitive code.

Simple event registration model

The *single event registration model* approach is slightly less quick, but more secure - and works in all modern browsers.

Events are properties of DOM Objects, and in turn you can assign an event to a function:

```
myObject.onclick = functionName;

// OR

myObject.onclick = function () {
  // code to do stuff
};
```

However, this approach still limits one event to one behaviour (unless you use nested function calls, but that results in hard to read code... fast)

Event listener registration model

The *event listener registration model* approach pays careful attention to the problems of the other two models and offers a simple way to register as many event handlers as you like for the same event on one element.

The `addEventListener()` method is used, and it takes 3 parameters:

- The event type
- The function to be executed
- A an optional boolean value, defaulted to false

This third boolean parameter is the trickiest to understand. It states whether the event handler should be executed in the *capturing* phase (true) or in the *bubbling* phase (false).

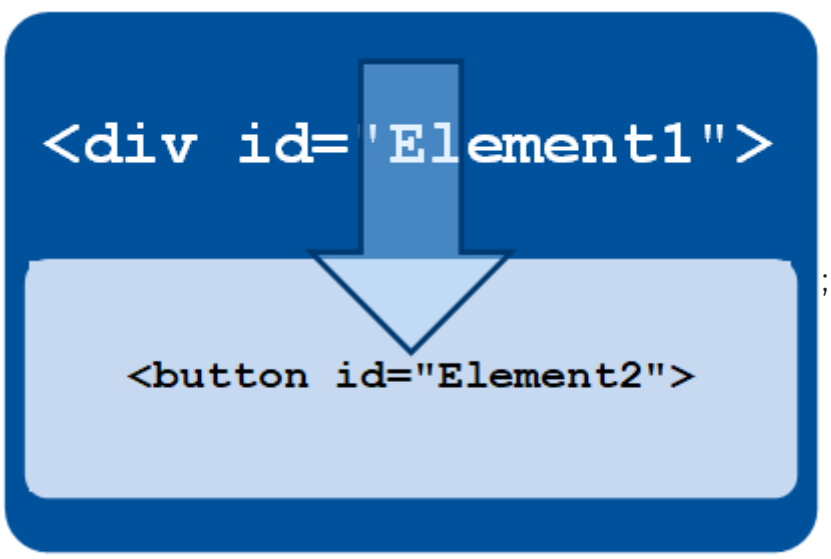
(*note: In older browsers (IE8 and before), only bubbling is applicable.)

Capturing vs Bubbling

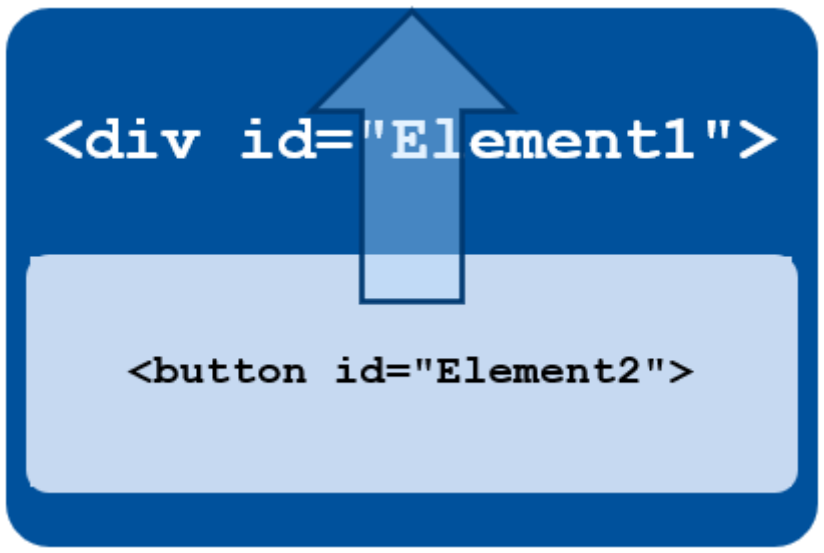
Events in JavaScript can bubble or be caught. Consider the problem of an element nested inside another element (e.g. a `button` in a `<div>`). Both have an `onclick` event; if the user clicks on the `button` they implicitly click on the `<div>`.

Which event handler should fire first?

With **event capturing** the event handler of `Element1` fires first:



With **event bubbling** the event handler of `Element2` fires first:

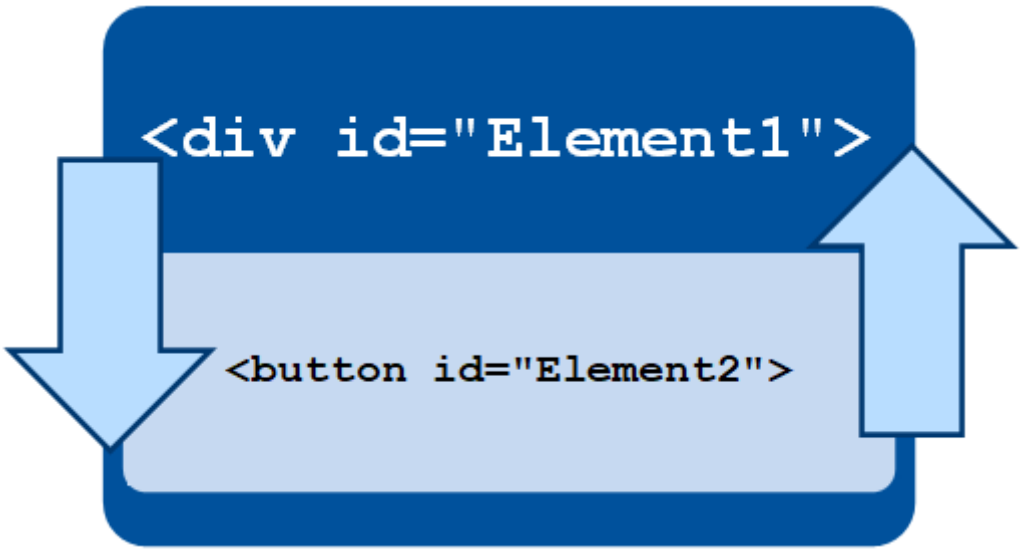


;

W3C Model

The W3C model could use either approach. The third parameter in `addEventListener()` sets how the event works.

You can mix both in the same page if appropriate:



▶ Event Capturing and Bubbling Example

Let's have a look at an example:

```
<!-- Inside HTML document -->
<button id="butt1" type="button">Click Me</button>
```

```
// Inside JavaScript document
let button = document.querySelector("#butt1");

function callMe() {
  alert("Hi there Friends");
}

function alsoCallMe() {
  alert("How can I help?");
}
// SYNTAX: button.addEventListener(event-type, function-to-execute, bubbling?)
button.addEventListener("click", callMe, false);
button.addEventListener("click", alsoCallMe);
```

When we click on the object associated to these events, we will fire off a call to both functions. It's important to note that the W3C model does not state which event handler is fired first and that can not be assumed.

Anonymous functions in event listeners

In many occasions we might want to conceal event-raising functions, so only the event handler can raise them. The event listener approach also allows us to continue this approach - sounds like a job for anonymous functions!

This can be extremely useful when we need to pass parameters to the listener. Other than the event itself, no parameter is passed when the event is raised (although it does maintain scope).

Instead, we can wrap the functions we want to call inside an anonymous function block and pass the parameter from the DOM element that raises the event:

```
// <<Insert HTML from before>>

button.addEventListener("click", function () {
  alert("Do Stuff");
});

// we can also nest functions
button.addEventListener("click", function () {
  changeClass(e, "div2");
});
```

Removing Event Listeners

Subscribed functions can be removed from an event at any time using the `removeEventListener()` function:

- Event type must be the same
- Event handler function must be the same
- Any options, including bubbling/capturing must be the same

```
<!-- Inside HTML document -->
<button id="butt1" type="button">Click Me</button>
```

```
// button.addEventListener("click", callMe, false); //Added previously
button.removeEventListener("click", callMe, false);
```

(The arguments must be exactly the same as the arguments used to add the event in the first place.)

When the above code is ran you should only see one alert appear in the browser.

This level of granularity allows us to choose which method will be unsubscribed.

Timed Events

We can also have timed events, in which we run a function or expect something to happen within a certain time frame. The `setTimeout()` method calls a function or evaluates an expression after a specified number of milliseconds:

```
setTimeout(function() {
  ...
}, milliseconds, param1, param2, ...)
```

For instance:

```
setTimeout(function () {
  alert("You will see this after 3 seconds");
}, 3000);
```

```
let myWindow = window.open("", "", "width=300,height=100");
myWindow.document.write("<p> Isn't this cool? </p>");
setTimeout(function () {
  myWindow.close();
}, 3000);
```

The function is only executed once. If you need to repeat execution, use `setInterval()` method.

Use the `clearTimeout()` method to prevent the function from running.

The possibilities are endless - we can use timed events inside other functions, or we can expect to call a timed event when a user first visits our Web page for instance.

Resources

For more information on events please refer to MDN Documentation.

Exercises

Create multiple event listeners which:

- Changes the first paragraph's text colour from blue to red.
- Changes the second paragraph's green background to pink.
- Change the font style of the last paragraph from Times New Roman to Arial.

1. Create a `index.js` file and write your JavaScript code to meet the above requirements.
2. Create a timed event of your choice for the Web page.
3. Create an event listener for a mouse/keyboard action.

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <meta http-equiv="X-UA-Compatible" content="ie=edge" />
    <title>QuickLab 11 - JavaScript Events</title>
    <style>
      #blueText {
        color: blue;
      }
      .greenBg {
        background-color: green;
      }
      .hotpinkBg {
        background-color: hotpink;
      }
    </style>
  </head>
  <body>
    <p id="blueText">I am a paragraph with blue text</p>
    <button id="textColour">Change blue to red</button>
    <p class="greenBg">
      I am a paragraph with a green background set by my CSS class
    </p>
    <p class="greenBg">
      I am another paragraph with a green background set by my CSS class
    </p>
    <p class="greenBg">
      I am another paragraph with a green background set by my CSS class
    </p>
    <button id="bgColour">Change green to pink</button>
    <p id="tnrParagraph">
      I am a paragraph with Times New Roman as its font style
    </p>
    <button id="fonts">Change Times New Roman to Arial</button>
    <script src="./index.js"></script>
  </body>
</html>
```

▶ Solution to 1.