Handling Events in JavaScript

When an [event](https://www.javascripttutorial.net/javascript-dom/javascript-events/) occurs, you can create an event handler which is a piece of code that will execute to respond to that event. An event handler is also known as an event listener. It listens to the event and responds accordingly to the event fires.

An event listener is a [function](https://www.javascripttutorial.net/javascript-function/) with an explicit name if it is resuable or an anonymous function in case it is used one time.

An event can be handled by one or multiple event handlers. If an event has multiple event handlers, all the event handlers will be executed when the event is fired.

There are three ways to assign event handlers.

1) HTML event handler attributes

Event handlers typically have names that begin with on, for example, the event handler for the click event is onclick.

To assign an event handler to an event associated with an HTML element, you can use an HTML attribute with the name of the event handler. For example, to execute some code when a button is clicked, you use the following:

**<input type="button" value="Save" onclick="alert('Clicked!')">**

Code language: HTML, XML (xml)

In this case, when the button is clicked, the [alert](https://www.javascripttutorial.net/javascript-bom/javascript-alert/) box is shown.

When you assign JavaScript code as the value of the onclick attribute, you need to escape the HTML characters such as ampersand (&), double quotes ("), less than (<), etc., or you will get a syntax error.

An event handler defined in the HTML can call a function defined in a script. For example:

**<script>**

function showAlert() {

alert('Clicked!');

}

**</script>**

**<input type="button" value="Save" onclick="showAlert()">**

Code language: HTML, XML (xml)

In this example, the button calls the showAlert() function when it is clicked.

The showAlert() is a function defined in a separate <script> element, and could be placed in an external JavaScript file.

Important notes

The following are some important points when you use the event handlers as attributes of the HTML element:

First, the code in the event handler can access the event object without explicitly defining it:

**<input type="button" value="Save" onclick="alert(event.type)">**

Code language: HTML, XML (xml)

Second, the this value inside the event handler is equivalent to the event’s target element:

**<input type="button" value="Save" onclick="alert(this.value)">**

Code language: HTML, XML (xml)

Third, the event handler can access the element’s properties, for example:

**<input type="button" value="Save" onclick="alert(value)">**

Code language: HTML, XML (xml)

Disadvantages of using HTML event handler attributes

Assigning event handlers using HTML event handler attributes are considered as bad practices and should be avoided as much as possible because of the following reasons:

First, the event handler code is mixed with the HTML code, which will make the code more difficult to maintain and extend.

Second, it is a timing issue. If the element is loaded fully before the JavaScript code, users can start interacting with the element on the webpage which will cause an error.

For example, suppose that the following showAlert() function is defined in an external JavaScript file:

**<input type="button" value="Save" onclick="showAlert()">**

Code language: HTML, XML (xml)

And when the page is loaded fully and the JavaScript has not been loaded, the showAlert() function is undefined. If users click the button at this moment, an error will occur.

2) DOM Level 0 event handlers

Each element has event handler properties such as onclick. To assign an event handler, you set the property to a function as shown in the example:

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

btn.onclick = function() {

alert('Clicked!');

};

Code language: JavaScript (javascript)

In this case, the anonymous function becomes the method of the button element. Therefore, the this value is equivalent to the element. And you can access the element’s properties inside the event handler:

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

btn.onclick = function() {

alert(this.id);

};

Code language: JavaScript (javascript)

Output:

btn

By using the this value inside the event handler, you can access the element’s properties and methods.

To remove the event handler, you set the value of the event handler property to null:

btn.onclick = null;

Code language: JavaScript (javascript)

The DOM Level 0 event handlers are still being used widely because of its simplicity and cross-browser support.

3) DOM Level 2 event handlers

DOM Level 2 Event Handlers provide two main methods for dealing with the registering/deregistering event listeners:

* addEventListener() – register an event handler
* removeEventListener() – remove an event handler

These methods are available in all DOM nodes.

The addEventListener() method

The addEventListener() method accepts three arguments: an event name, an event handler function, and a Boolean value that instructs the method to call the event handler during the capture phase (true) or during the bubble phase (false). For example:

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

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

alert(event.type); *// click*

});

Code language: JavaScript (javascript)

It is possible to add multiple event handlers to handle a single event, like this:

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

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

alert(event.type); *// click*

});

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

alert('Clicked!');

});

Code language: JavaScript (javascript)

The removeEventListener() method

The removeEventListener() removes an event listener that was added via the addEventListener(). However, you need to pass the same arguments as were passed to the addEventListener(). For example:

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

*// add the event listener*

let showAlert = function() {

alert('Clicked!');

};

btn.addEventListener('click', showAlert);

*// remove the event listener*

btn.removeEventListener('click', showAlert);

Code language: JavaScript (javascript)

Using an anonymous event listener as the following will not work:

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

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

alert('Clicked!');

});

*// won't work*

btn.removeEventListener('click', function() {

alert('Clicked!');

});

Code language: JavaScript (javascript)

Summary

* There are three ways to assign an event handler: HTML event handler attribute, element’s event handler property, and addEventListener().
* Assign an event handler via the HTML event handler attribute should be avoided.