JavaScript
TUTORIAL

JavaScript Events

Summary: in this tutorial, you will learn about JavaScript events, event models, and how to handle events.

Introduction to JavaScript Events

An event is an action that the web browser can detect and respond to, like a mouse click or a page load.

For example, you might want to display an alert when a user clicks a button.

An event may have an event handler, a function that runs when the event occurs. An event handler, also known as an event listener, listens for the event and executes when it happens.

Suppose you have a button with the id `btn` :

```
<button id="btn">Click Me!</button>
```

To define a function that will be executed when the button is clicked, you need to register an event handler using the `addEventListener()` method:

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

function handleClick() {
    alert('It was clicked!');
}

btn.addEventListener('click', handleClick);
```

How it works.

- First, select the button with the id `btn` by using the `querySelector()` method.

- Second, define a function called `handleClick()` as an event handler.
- Third, register an event handler using the `addEventListener()` so that when users click the button, the `display()` function will execute.

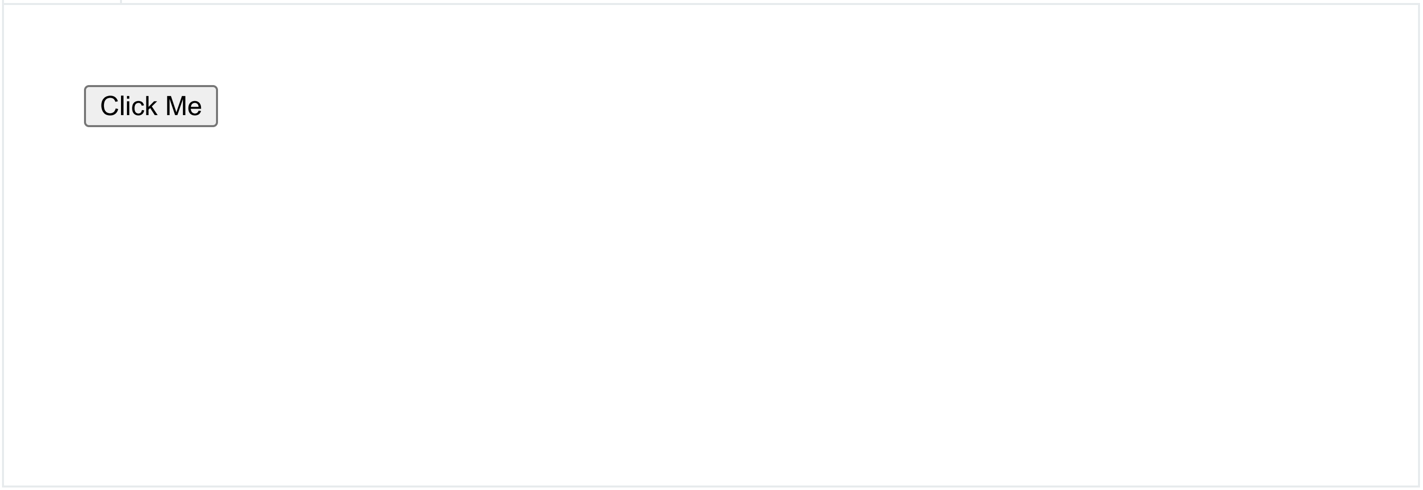
A shorter way to register an event handler is to place all code in an anonymous function, like this:

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

btn.addEventListener('click',function() {
    alert('It was clicked!');
});
```

Alternatively, you can use an arrow function:

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

btn.addEventListener('click',() => {
    alert('It was clicked!');
});
```

Output



Event flow

Assuming that you have the following HTML document:

```html
<!DOCTYPE html>
<html>
<head>
    <title>JS Event Demo</title>
</head>
<body>
    <div id="container">
        <button id='btn'>Click Me!</button>
    </div>
</body>
```

When you click the button, the click event occurs not only on the button but also on the button's container, the `div`, and the whole webpage.

Event flow explains the order in which events are received on the page from the element where the event occurs and propagated through the DOM tree.

There are two main event models:

- Event bubbling
- Event capturing

Event bubbling

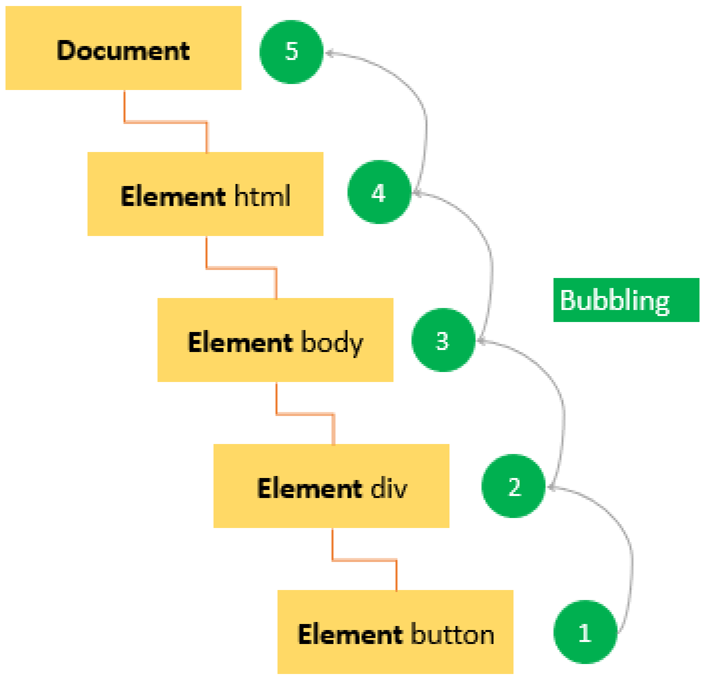
In the event bubbling model, an event starts at the most specific element and then flows upward toward the least specific element (the `document` or even `window`).

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:



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

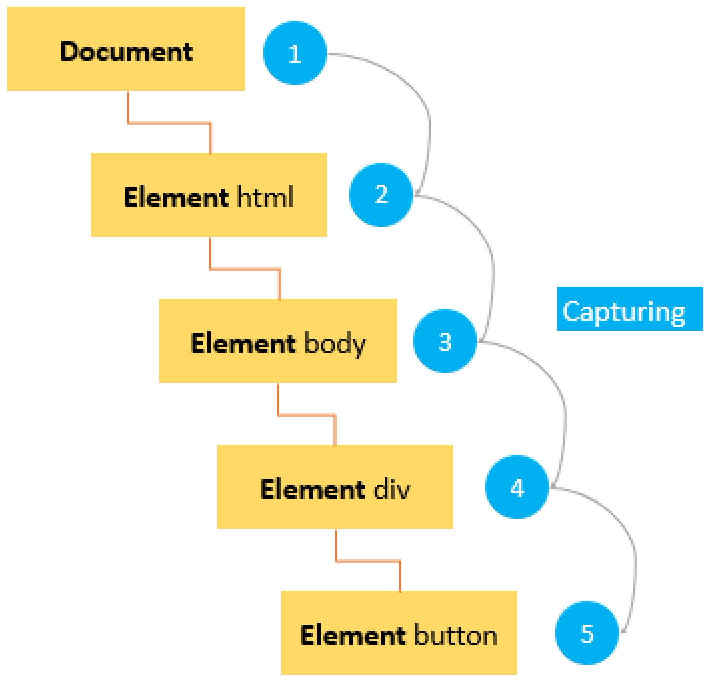
Event capturing

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:



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:

Event object

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

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

btn.addEventListener('click', function(event) {
    console.log(event.type);
});
```

Output:

```
'click'
```

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 information about 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:

```html
<a href="https://www.javascripttutorial.net/">JS Tutorial</a>
```

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

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

link.addEventListener('click',function(event) {
    console.log('clicked');
    event.preventDefault();
});
```

Note that the `preventDefault()` method does not stop the event from bubbling up the DOM. 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 browser's default behavior.

See the following example:

```javascript
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!');
});
```

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.

Summary

- An event is an action occurred in the web browser e.g., a mouse click.
- Event flow has two main models: event bubbling and event capturing.
- DOM Level 2 Event specifies that the event flow has three phases: event bubbling, the event occurring at the exact element, and event capturing.
- Use `addEventListener()` to register an event that connects an event to an event listener
- The `event` object is accessible only within the event listener.
- Use `preventDefault()` method to prevent the default behavior of an event, but does not stop the event flow.
- Use `stopPropagation()` method to stop the flow of an event through the DOM tree, but does not cancel the browser default behavior.

Quiz