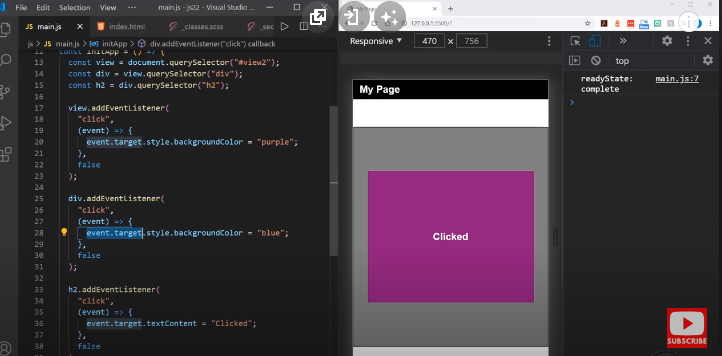
Events Properties

The target property returns the element where the event occured.

The target property is read-only.

The target property returns the element on which the event occurred, opposed to the currentTarget property, which returns the element whose event listener triggered the event.



<<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta http-equiv="X-UA-Compatible" content="IE=edge">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>event.target</title>

</head>

<body>

<input type="text" name="" id="inp">

<script>

let inputBox = document.getElementById("inp");

inputBox.addEventListener("input", (event)=>{

console.log(event.target) //Gets the element

console.log(event.target.value) //Gets a property of the element

event.target.style.color = "red"; //Modifies the css property of the element

})

</script>

</body>

</html>

**textContent property**

The textContent property of the Node interface represents the text content of the node and its descendants.

The textContent property sets or returns the text content of the specified node, and all its descendants.

Note: textContent and HTMLElement.innerText are easily confused, but the two properties are different in important ways.

JS

let note = document.getElementById('note');

note.textContent = 'This is a note';

**Differences from innerText**

Don't get confused by the differences between Node.textContent and HTMLElement.innerText. Although the names seem similar, there are important differences:

textContent gets the content of all elements, including <script> and <style> elements. In contrast, innerText only shows "human-readable" elements.

textContent returns every element in the node. In contrast, innerText is aware of styling and won't return the text of "hidden" elements.

Moreover, since innerText takes CSS styles into account, reading the value of innerText triggers a reflow to ensure up-to-date computed styles. (Reflows can be computationally expensive, and thus should be avoided when possible.)

**Differences from innerHTML**

Element.innerHTML returns HTML, as its name indicates. Sometimes people use innerHTML to retrieve or write text inside an element, but textContent has better performance because its value is not parsed as HTML.

Moreover, using textContent can prevent XSS attacks.

Examples

Start with this HTML fragment.

HTML

<div id="divA">This is <span>some</span> text!</div>

You can use textContent to get the element's text content:

JS

let text = document.getElementById("divA").textContent;

// The text variable is now: 'This is some text!'

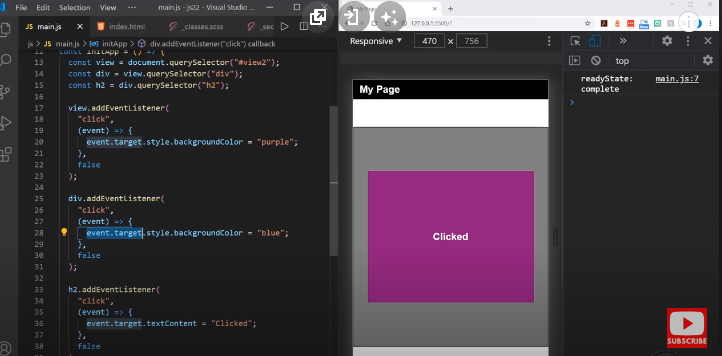
If you prefer to set the element's text content, you can do:

JS

document.getElementById("divA").textContent = "This text is different!";

// The HTML for divA is now:

// <div id="divA">This text is different!</div>



**Classlist add(), remove(), toggle();**

The Element.classList is a read-only property that returns a live DOMTokenList collection of the class attributes of the element. This can then be used to manipulate the class list.

Using classList is a convenient alternative to accessing an element's list of classes as a space-delimited string via element.className.

This proerty use class names

**Value**

1. A DOMTokenList representing the contents of the element's class attribute. If the class attribute is not set or empty, it returns an empty DOMTokenList, i.e. a DOMTokenList with the length property equal to 0.
2. Although the classList property itself is read-only, you can modify its associated DOMTokenList using the add(), remove(), replace(), and toggle() methods.

You can test whether the element contains a given class using the classList.contains() method.

Examples

JS

const div = document.createElement("div");

div.className = "foo";

// our starting state: <div class="foo"></div>

console.log(div.outerHTML);

// use the classList API to remove and add classes

div.classList.remove("foo");

div.classList.add("anotherclass");

// <div class="anotherclass"></div>

console.log(div.outerHTML);

// if visible is set remove it, otherwise add it

div.classList.toggle("visible");

// add/remove visible, depending on test conditional, i less than 10

div.classList.toggle("visible", i < 10);

// false

console.log(div.classList.contains("foo"));

// add or remove multiple classes

div.classList.add("foo", "bar", "baz");

div.classList.remove("foo", "bar", "baz");

// add or remove multiple classes using spread syntax

const cls = ["foo", "bar"];

div.classList.add(...cls);

div.classList.remove(...cls);

// replace class "foo" with class "bar"

div.classList.replace("foo", "bar");

