Node: cloneNode() method

- Syntax
  - Parameters
  - Return value
- Example
  - Using cloneNode()
  - Using cloneNode() with templates
- Specifications
- Browser compatibility

The `cloneNode()` method of the Node interface returns a duplicate of the node on which this method was called. Its parameter controls if the subtree contained in the node is also cloned or not.

By default, cloning a node copies all of its attributes and their values, including event listeners specified via attributes. By setting the `deep` parameter, you can also copy the subtree contained in the node. It does *not* copy any other internal data, such as event listeners added using `addEventListener()` or `onevent` properties (e.g., `node.onclick = someFunction`), or the painted image for a canvas element.

The Document.importNode() method also creates a copy of a node. The difference is that `importNode()` clones the node in the context of the calling document, whereas `cloneNode()` uses the document of the node being cloned. The document context determines the CustomElementRegistry for constructing any custom elements. For this reason, to clone nodes to be used in another document, use `importNode()` on the target document. The HTMLTemplateElement.content is owned by a separate document, so it should also be cloned using `document.importNode()` so that custom element descendants are constructed using the definitions in the current document.

> **WARNING:** `cloneNode()` may lead to duplicate element IDs in a document! If the original node has an `id` attribute, and the clone will be placed in the same

> document, then you should modify the clone's ID to be unique.
>
> Also, `name` attributes may need to be modified, depending on whether duplicate names are expected.

Syntax

```
cloneNode()
cloneNode(deep)
```

Parameters

- `deep`
  - : If `true`, then the node and its whole subtree, including text that may be in child Text nodes, is also copied.
    If `false` or omitted, only the node will be cloned. The subtree, including any text that the node contains, is not cloned.

    Note that `deep` has no effect on void elements, such as the img and input elements. It defaults to `false`.

Return value

The new Node cloned. The cloned node has no parent and is not part of the document, *until* it is added to another node that is part of the document, using Node.appendChild() or a similar method.

Example

Using cloneNode()

```
const p = document.getElementById("para1");
const p2 = p.cloneNode(true);
```

Using cloneNode() with templates

Avoid using `cloneNode()` on the content of a template element, because if the template contains custom elements, they will not be upgraded until they are inserted into the document.

```
class MyElement extends HTMLElement {
  constructor() {
    super();
    console.log("MyElement created");
  }
}
customElements.define("my-element", MyElement);

const template = document.createElement("template");
template.innerHTML = `<my-element></my-element>`;

const clone = template.content.cloneNode(true);
// No log here; my-element is undefined in the template's document
customElements.upgrade(clone);
// Still no log; my-element is still undefined in the template's document
document.body.appendChild(clone);
// Logs "MyElement created"; my-element is now upgraded
```

Instead, use `document.importNode()` to clone the template content, so that any custom elements are upgraded using the definitions in the current document:

```
const clone = document.importNode(template.content, true);
// Logs "MyElement created"; my-element is upgraded using document's definitions
document.body.appendChild(clone);
```

Specifications

Browser compatibility

□□□ #3
□ □□□ □□□ 2026-10-31 03:27:53 UTC
□ cocobook-sync □□□ 2026-11-04 17:34:47 UTC