Element: moveBefore() method

- Syntax
  - Parameters
  - Return value
  - Exceptions
- Description
  - moveBefore() constraints
  - Moving custom elements while preserving state
- Examples
  - Basic moveBefore() usage
    - HTML
    - CSS
    - JavaScript
    - Result
  - Demonstrating state preservation
    - HTML
    - CSS
    - JavaScript
    - Result
- Specifications
- Browser compatibility
- See also

The `moveBefore()` method of the Element interface moves a given Node inside the invoking node as a direct child, before a given reference node.

Syntax

```
moveBefore(movedNode, referenceNode)
```

Parameters

- `movedNode`
  - : A Node representing the node to be moved. Note that this must be an Element or a CharacterData node.
- `referenceNode`
  - : A Node that `movedNode` will be moved before, or `null`. If the value is `null`, `movedNode` is inserted at the end of the invoking node's child nodes.

Return value

None (undefined).

Exceptions

- `HierarchyRequestError` DOMException
  - : Thrown in any of the following situations:
    - The specified `movedNode` is not part of the DOM, and you are trying to move it inside a node that is part of the DOM, or vice versa.
    - The specified `movedNode` is an ancestor of the Element `moveBefore()` is being called on.
    - You are trying to move `movedNode` between two different documents.
    - The specified `movedNode` is not an Element or CharacterData node.
- `NotFoundError` DOMException
  - : The specified `referenceNode` is not a child of the node you are calling `moveBefore()` on, that is, the node you are trying to move `movedNode` inside.
- TypeError
  - : The second argument was not supplied.

Description

The `moveBefore()` method moves a given node to a new place in the DOM. It provides similar functionality to the Node.insertBefore() method, except that it doesn't remove and then reinsert the node. This means that the state of the node (which would be reset if moving it with `insertBefore()` and similar mechanisms) is preserved after the move. This includes:

- Animation and transition state.
- iframe loading state.
- Interactivity states (for example, :focus and :active).
- Fullscreen element state.
- Open/close state of popovers.
- Modal state of dialog elements (modal dialogs will not be closed).

The play state of video and audio elements is not included in the above list, as these elements retain their state when removed and reinserted, regardless of the mechanism used.

When observing changes to the DOM using a MutationObserver, nodes moved with `moveBefore()` will be recorded with a removed node and an added node.

`moveBefore()` constraints

There are some constraints to be aware of when using `moveBefore()`:

- It can only work when moving a node within the same document.
- It won't work if you try to move a node that is not connected to the DOM to an already connected parent, or vice versa.

In such cases, `moveBefore()` will fail with a `HierarchyRequestError` exception. If the above constraints are requirements for your particular use case, you should use Node.insertBefore() instead, or use `try...catch` to handle the errors that arise from such cases.

Moving custom elements while preserving state

Each time a custom element's position in the DOM is updated via `Element.moveBefore()`, or similar methods such as Node.insertBefore(), its `disconnectedCallback()` and `connectedCallback()` lifecycle callbacks are fired. Since these callbacks are typically used to implement any required initialization or cleanup code to run at the start or end of the element's lifecycle, running them when the element is moved (rather than removed or inserted) may cause problems with its state.

You can use the `connectedMoveCallback()` callback to preserve a custom element's state. When using `moveBefore()` to move a custom element, `connectedMoveCallback()` is run instead of `connectedCallback()` and `disconnectedCallback()`.

See Moving custom elements for further information.

Examples

Basic `moveBefore()` usage

In this demo we illustrate basic usage of `moveBefore()`.

HTML

The HTML features an article element containing a div element and two section elements. The `<div>` contains a button, which we later use to move it.

```
<article id="wrapper">
  <div id="mover">
    <button>Move me!</button>
  </div>
  <section id="section1">
    <h2>Section 1</h2>
  </section>
  <section id="section2">
    <h2>Section 2</h2>
  </section>
</article>
```

CSS

We provide some rudimentary styling for the look and feel and spacing of the boxes, and use flexbox to center their content.

```
#section1,
#section2,
#mover {
  width: 200px;
```

```
  height: 80px;
  border: 5px solid rgb(0 0 0 / 0.25);
  margin-bottom: 10px;
  display: flex;
  align-items: center;
  justify-content: center;
}

#section1,
#section2 {
  background-color: hotpink;
}

#mover {
  background-color: orange;
}
```

JavaScript

In our script, we attach a click event listener to the `<button>` via addEventListener(). When the button is clicked, we check whether the nextElementSibling of our `mover` `<div>` is the first `<section>` element. If so, we invoke `moveBefore()` on the `wrapper` `<article>` and specify to move the `<div>` before the second `<section>`. If not, we use `moveBefore()` to move the `<div>` before the first `<section>`.

```
const wrapper = document.getElementById("wrapper");
const section1 = document.getElementById("section1");
const section2 = document.getElementById("section2");
const mover = document.getElementById("mover");
const moveBtn = document.querySelector("button");

moveBtn.addEventListener("click", () => {
  if (mover.nextElementSibling === section1) {
    wrapper.moveBefore(mover, section2);
  } else {
    wrapper.moveBefore(mover, section1);
  }
});
```

Result

The rendered example looks like this:

Try clicking the `<button>` a few times and note how it toggles between the two positions.

Demonstrating state preservation

In this demo we provide multiple mechanisms to move a `<div>` element containing a YouTube embed between two different containers, demonstrating how `moveBefore()` preserves the play state of the embed, but the other mechanisms do not.

HTML

The HTML features an article element containing two section elements. The first `<section>` element contains a div element containing the YouTube embed code. We also have a div element containing three button elements, to which we will add functionality to move the embed `<div>` between sections via JavaScript later on.

```
<article id="wrapper">
  <section id="section1">
    <div id="mover">
      <iframe
        width="300"
        height="200"
        src="https://www.youtube.com/embed/XvoENpR9cCQ?si=o2i6MvxugD-O5yyv"
        title="YouTube video player"
        allow="accelerometer; autoplay; clipboard-write; encrypted-media; gyroscope; picture-
in-picture; web-share"
        referrerpolicy="strict-origin-when-cross-origin"
        allowfullscreen></iframe>
    </div>
  </section>
  <section id="section2"></section>
</article>
<div id="controls">
  <button id="move-before">move with <code>moveBefore()</code></button>
  <button id="insertbefore">move with <code>insertBefore()</code></button>
  <button id="prepend">move with <code>prepend()</code></button>
</div>
```

CSS

We use flexbox for the layout to make the two `<section>` elements sit side-by-side, and space the buttons evenly inside the `controls` `<div>`.

```
#wrapper,
#controls {
  width: 100%;
  display: flex;
}

#wrapper {
  margin-bottom: 10px;
}

iframe {
  border: none;
}

section {
  flex: 1;
  padding: 10px;
}

#controls {
  display: flex;
  justify-content: space-around;
}

#section1 {
  background-color: hotpink;
}

#section2 {
  background-color: orange;
}

#mover {
  max-width: 100%;
  background-color: black;
}
```

JavaScript

In our script, we attach `click` event listeners to each `<button>` via addEventListener(). When the buttons are clicked, we check which `<section>` element is the parentElement of our embed `<div>`, and then use the relevant function (`moveBefore()`, insertBefore(), or prepend()) to move it inside the *other* `<section>` element.

```
const section1 = document.getElementById("section1");
const section2 = document.getElementById("section2");
const mover = document.getElementById("mover");
const moveBeforeBtn = document.getElementById("move-before");
const insertbeforeBtn = document.getElementById("insertbefore");
const prependBtn = document.getElementById("prepend");

moveBeforeBtn.addEventListener("click", () => {
  if (mover.parentElement === section1) {
    section2.moveBefore(mover, null);
  } else {
    section1.moveBefore(mover, null);
  }
});

insertbeforeBtn.addEventListener("click", () => {
  if (mover.parentElement === section1) {
    section2.insertBefore(mover, null);
  } else {
    section1.insertBefore(mover, null);
  }
});

prependBtn.addEventListener("click", () => {
  if (mover.parentElement === section1) {
    section2.prepend(mover);
  } else {
    section1.prepend(mover);
  }
});
```

Result

The rendered example looks like this:

Try playing the YouTube embed and then clicking each `<button>` a couple of times to toggle the `<div>` element screen position from left to right. Note how, in the case of `insertBefore()` and `prepend()`, the embed state is reset after each move so it needs to be restarted. However, in the case of `moveBefore()`, the state is preserved after each move.

Specifications

Browser compatibility

See also

- Document.moveBefore()
- DocumentFragment.moveBefore()
- Node.insertBefore()

□□□ #3
□ □□□ □□□ 2026-10-31 03:27:53 UTC
□ cocobook-sync □□□ 2026-11-04 17:34:29 UTC