JavaScript DOMContentLoaded Event

Summary: in this tutorial, you will learn about the JavaScript `DOMContentLoaded` event.

Introduction to JavaScript DOMContentLoaded Event

The `DOMContentLoaded` event fires when the initial HTML document has been completely loaded and parsed, without waiting for stylesheets, images, frames, and async script `<script async src="...">` to complete loading.

Here are the key points of the `DOMContentLoaded` event:

- The `DOMContentLoaded` event is fired when the HTML document has been completely parsed.
- It does not wait for external resources such as stylesheets, images, frames, and async script `<script async src="...">` to finish loading.
- All deferred scripts `<script defer src="...">` and modules `<script type="module">` have been downloaded and executed.
- The `DOMContentLoaded` event cannot be cancellable.

To add an event handler to the `DOMContentLoaded` event, you use the `addEventListener()` method like this:

```
document.addEventListener("DOMContentLoaded", (event) => {
  console.log("DOM fully loaded and parsed.");
});
```

In this example, the code will show the message “DOM fully loaded and parsed.” in the console once the DOM is fully loaded

JavaScript DOMContentLoaded Event examples

In practice, you can listen to the `DOMContentLoaded` event when you place the JavaScript in the `head` of the page but referencing elements in the `body`, for example:

```
<!DOCTYPE html>
<html>

<head>
    <title>JS DOMContentLoaded Event</title>
    <script>
        let btn = document.getElementById('btn');
        btn.addEventListener('click', (e) => {
            // handle the click event
            console.log('clicked');
        });
    </script>
</head>

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

</html>
```

The example will cause an error because the button has not been loaded when the script tag runs.

To fix that, you can place the above code within a `DOMContentLoaded` event handler, like this:

```
<!DOCTYPE html>
<html>

<head>
    <title>JS DOMContentLoaded Event</title>
    <script>
        document.addEventListener('DOMContentLoaded', () => {
            let btn = document.getElementById('btn');
            btn.addEventListener('click', () => {
                // handle the click event
                console.log('clicked');
            });
        });
    </script>
</head>
```

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

</html>
```

When you place JavaScript in the header, it will cause bottlenecks and rendering delays. So it's better to move the script before the `</body>` tag. In this case, you don't need to place the code in the `DOMContentLoaded` event:

```
<!DOCTYPE html>
<html>

    <head>
        <title>JS DOMContentLoaded Event</title>
    </head>

    <body>
        <button id="btn">Click Me!</button>
        <script>
            let btn = document.getElementById('btn');

            // add an event listener
            btn.addEventListener('click', (e) => {
                console.log('clicked');
            });
        </script>
    </body>

</html>
```

readyState

An HTML document has three loading states:

- `"loading"` – the document is loading.
- `"interactive"` – the document was fully read.
- `"complete"` – the document was fully read and all resources such as images were loaded.

The `document.readState` property stores the current loading state.

If the document is loaded and you register a `DOMContentLoaded` event handler, the event handler will never execute.

To properly set up a handler or execute the event handler immediately if the document is ready, you can check the `readyState` property like this:

```
<!DOCTYPE html>
<html>
    <head>
        <title>JS DOMContentLoaded Event</title>
        <script>
            function handleReady() {
                console.log('DOM ready');
            }

            if (document.readyState === 'loading') {
                console.log('The document is loading...');
                document.addEventListener('DOMContentLoaded', handleReady);
            } else {
                console.log('The document has been loaded.');
                handleReady();
            }
        </script>
    </head>
    <body>
    </body>
</html>
```

Summary

- The `DOMContentLoaded` event fires when the DOM content is loaded, without waiting for external resources such as images stylesheets, and frames to finish loading.
- Only handle `DOMContentLoaded` event if you place the JavaScript code in the `head` and reference elements in the `body`.
- The readyState stores the current document loading state including `loading`, `interactive`, and `complete`.