JavaScript Event Delegation

Summary: in this tutorial, you will learn how to use the JavaScript event delegation that adds a single event handler to the parent element instead of having to register multiple event handlers to the child elements.

Introduction to JavaScript Event Delegation

Suppose that you have the following menu:

```
<ul id="menu">
    <li><a id="home">home</a></li>
    <li><a id="dashboard">Dashboard</a></li>
    <li><a id="report">report</a></li>
</ul>
```

To handle the `click` event of each menu item, you may add the corresponding `click` event handlers:

```
let home = document.querySelector('#home');
home.addEventListener('click',(event) => {
    console.log('Home menu item was clicked');
});

let dashboard = document.querySelector('#dashboard');
dashboard.addEventListener('click',(event) => {
    console.log('Dashboard menu item was clicked');
});

let report = document.querySelector('#report');
report.addEventListener('click',(event) => {
```

```
    console.log('Report menu item was clicked');
});
```

In JavaScript, if you have a large number of event handlers on a page, these event handlers will directly impact the performance because of the following reasons:

- First, each event handler is a function which is also an object that takes up memory. The more objects in the memory, the slower the performance.
- Second, it takes time to assign all the event handlers, which causes a delay in the interactivity of the page.

To solve this issue, you can leverage the event bubbling.

Instead of having multiple event handlers, you can assign a single event handler to handle all the `click` events:

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

menu.addEventListener('click', (event) => {
    let target = event.target;

    switch(target.id) {
        case 'home':
            console.log('Home menu item was clicked');
            break;
        case 'dashboard':
            console.log('Dashboard menu item was clicked');
            break;
        case 'report':
            console.log('Report menu item was clicked');
            break;
    }
});
```

How it works.

- When you click any `<a>` element inside the `<ul>` element with the id `menu`, the `click` event bubbles to the parent element which is the `<ul>` element. So instead of handling the `click` event of the individual `<a>` element, you can capture the `click` event at the parent element.
- In the `click` event listener, you can access the `target` property which references the element that dispatches the event. To get the `id` of the element that the event fires, you use the `target.id` property.
- Once having the `id` of the element that fires the `click` event, you can have the code that handles the event correspondingly.

The way that we handle the too-many-event-handlers problem is called the **event delegation**.

The event delegation refers to the technique of using event bubbling to handle events at a higher level in the DOM than the element on which the event originated

JavaScript event delegation benefits

When it is possible, you can have a single event handler on the `document` that will handle all the events of a particular type. By doing this, you gain the following benefits:

- Less memory usage, better performance.
- Less time is required to set up event handlers on the page.
- The `document` object is available immediately. As long as the element is rendered, it can start functioning correctly without delay. You don't need to wait for the `DOMContentLoaded` or `load` events.

Summary

- Having a large number of event handlers will take up memory and degrade the performance of a page.
- The event delegation technique utilizes the event bubbling to handle the event at a higher level in the DOM than the element on which the event originated.