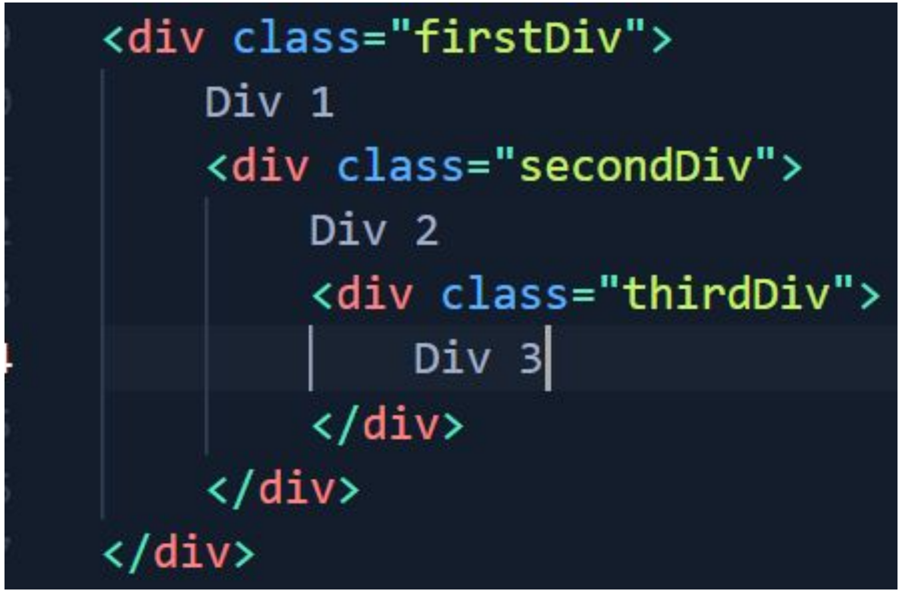
Javascript Day - 16

Event Bubbling and Event Capturing

Introduction

If you have an event handler assigned to an element which nested elements. Examples:



```
<div class="firstDiv">
    Div 1
    <div class="secondDiv">
        Div 2
        <div class="thirdDiv">
            Div 3
        </div>
    </div>
</div>
```

Introduction

We have an event handler assigned to firstDiv.

Even though the handler is added to firstDiv but if we click on any other inner div.

The firstDiv event handler will get triggered.

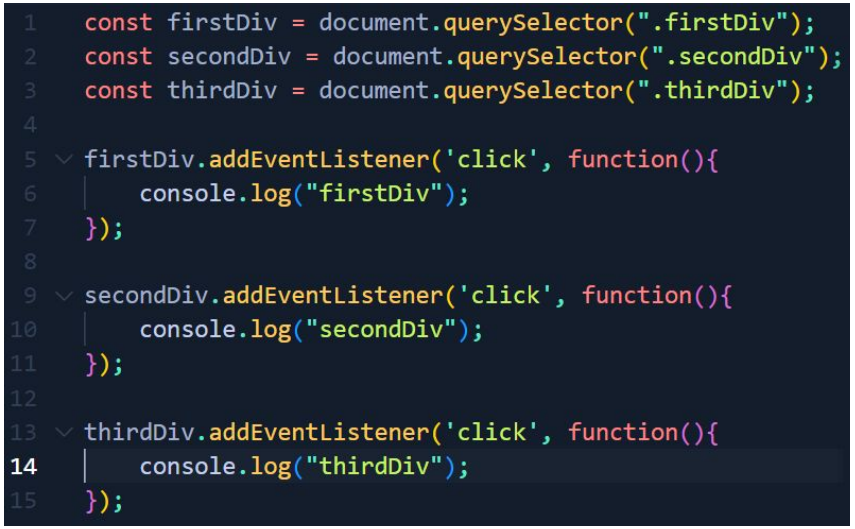
Isn’t it a bit strange? Why does the handler on firstDiv run if the actual click was on thirdDiv ?

Bubbling

The bubbling principle is simple.

When an event happens on an element, it first runs the handlers on it, then on its parent, then all the way up on other ancestors.

Example:



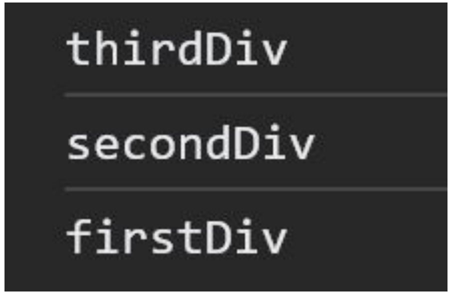
```javascript
const firstDiv = document.querySelector(".firstDiv");
const secondDiv = document.querySelector(".secondDiv");
const thirdDiv = document.querySelector(".thirdDiv");

firstDiv.addEventListener('click', function(){
    console.log("firstDiv");
});

secondDiv.addEventListener('click', function(){
    console.log("secondDiv");
});

thirdDiv.addEventListener('click', function(){
    console.log("thirdDiv");
});
```

Bubbling

A click on the inner div first runs onclick:

- On that div.
- Then on the outer <div>.
- Then on the outer <div>.
- And so on upwards till the document object.

So if we click on the inner most div (thirdDiv) we will see three console log



```
thirdDiv
secondDiv
firstDiv
```

event.target

A handler on a parent element can always get the details about where it actually happened.

The most deeply nested element that caused the event is called a target element, accessible as event.target.

event.target – is the “target” element that initiated the event, it doesn’t change through the bubbling process.

To get the element on which the handler is applied event.currentTarget can be used.

Stopping Bubbling

A bubbling event goes from the target element straight up. Normally it goes upwards till <html>, and then to document object, and some events even reach window, calling all handlers on the path.

But any handler may decide that the event has been fully processed and stop the bubbling.

The method for it is event.stopPropagation().

Stopping Bubbling

If an element has multiple event handlers on a single event, then even if one of them stops the bubbling, the other ones still execute.

In other words, event.stopPropagation() stops the move upwards, but on the current element all other handlers will run.

To stop the bubbling and prevent handlers on the current element from running, there's a method event.stopImmediatePropagation(). After it no other handlers execute.

Stopping Bubbling

WARNING: Bubbling is convenient. Don’t stop it without a real need: obvious and architecturally well thought out.

There’s usually no real need to prevent the bubbling

Capturing

There’s another phase of event processing called “capturing”. It is rarely used in real code, but sometimes can be useful.

In fact, the capturing phase was invisible for us, because handlers added using on<event>-property or using HTML attributes or using two-argument addEventListener(event, handler) don’t know anything about capturing, they only run on the 2nd and 3rd phases.