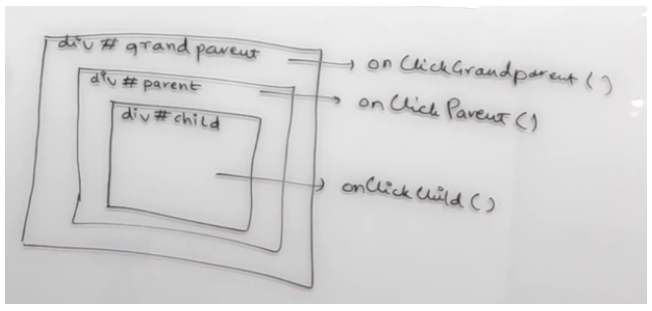
Event Bubbling, Capturing aka Trickling in JavaScript

Friday, 25 August 2023 12:56 PM

→ Event bubbling & capturing are 2 ways of events propagation in the DOM Tree.

→ Let's consider nested HTML elements :



→ So each of these f's is attached to its respective DOM element.

→ So, in event bubbling when we click on the child element, onClickChild() method will be called first, then it moves up the hierarchy & it goes directly till the end of the DOM. So, first onClickChild() is called, then onClickParent() is called & then onClickGrandParent() is called.

→ To remember this we can imagine how a bubble always moves up.

→ Opposite to this is event capturing. So, this goes up to down the DOM tree. So first onClickGrandparent() is called then onClickParent() is called & finally onClickChild() is called.

Note: If we click on Parent div, then in case of bubbling, onClickParent() & then onClickGrandParent() is clicked. But in case of capturing it is the opposite, first onClickGrandparent() is clicked & then onClickParent() is clicked.

→ Event capturing is also known as event tridding.

→ Syntax for event bubbling & capturing:

```
addEventListener('click', () => {...}, useCapture)
```

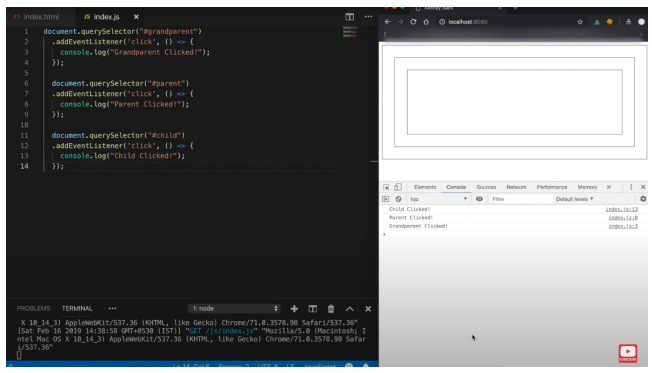
'click' → event; () => {...} → callback f^n; useCapture → boolean value

→ When useCapture is true, then event capturing mode is enabled.
→ If we don't pass any value or pass a false value then event bubbling mode is enabled.

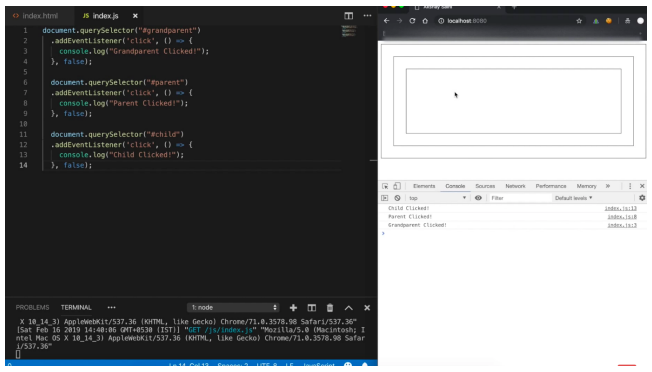
Code demonstration :

```html
<!DOCTYPE html>
<head>
  <title>Akshay Saini</title>

</head>
<body>

  <div id="grandparent">
    <div id="parent">
      <div id="child"></div>
    </div>
  </div>

  <script src="./js/index.js"></script>
</body>
</html>
```

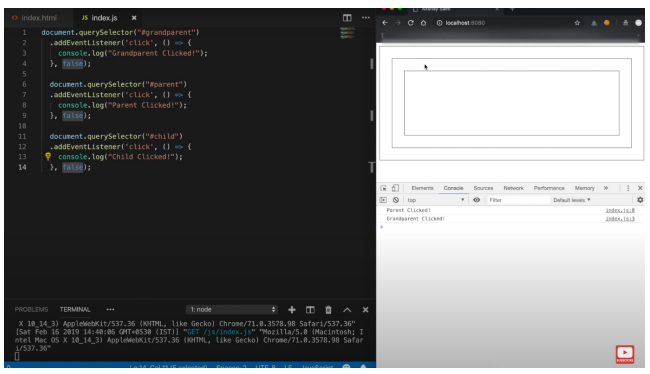
→ HTML file with 3 divs, one inside the other.



→ Output when we click the child div. So, it is following event bubbling

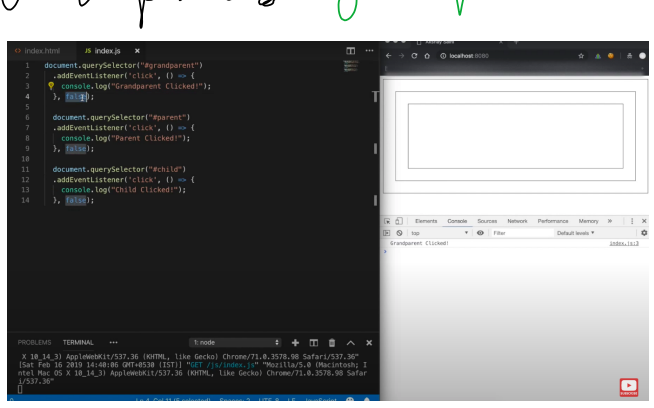


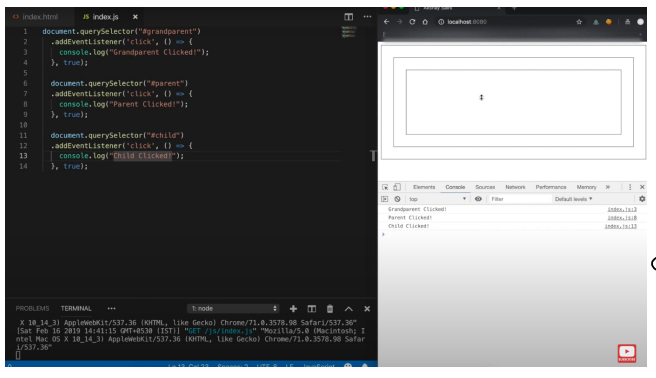
→ This is the same as above.



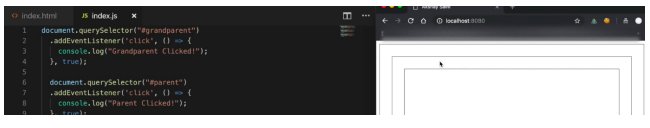
→ output when we click the parent element. Then, the event is bubbled out to the grandparent element.

→ If we had clicked on the grandparent element then it would have just printed grandparent clicked.

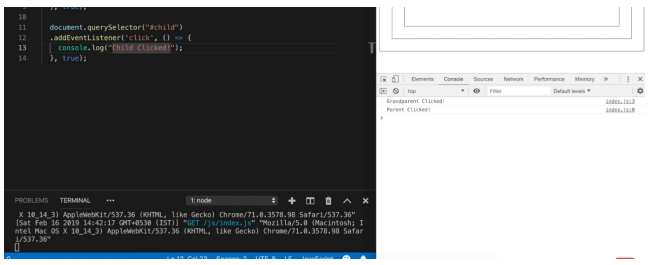




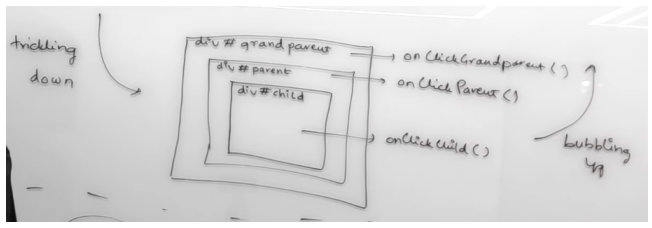
→ Now, we've made the useCapture flag as true. So when we click on child element, event capturing is enabled & event bubbles down the DOM tree. Hence, the following output.



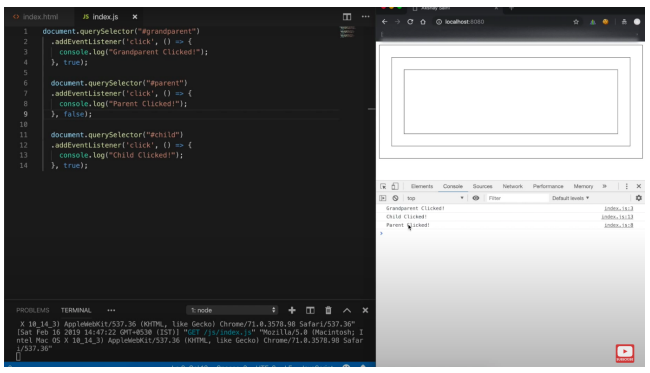
→ Similarly if we click on the parent div,



then event trickles down the hierarchy from grandparent to parent & the attached callback fn's are run. Hence, the following output.

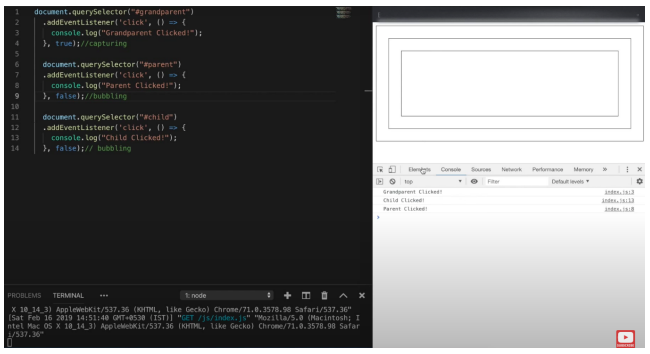


→ According to W3C, the whole chain is always followed, first trickling down happens & then bubbling up happens.
→ But this can be adjusted using useCapture parameter.



→ Now, we've passed false to parent element's useCapture param.

→ This happens because, first event captering happens & then event bubbling happens in a cycle.
→ So, during event capturing onClickGrandparent() is clicked first, then in parent div as useCapture flag is false, thus, it won't be included in event capturing cycle. then it moves to child div & since its useCapture value is true so its callback fn is included in event capturing cycle & 'child clicked' is printed.
→ After this, bubbling happens, & since parent div's useCapture value is false so its callback fn will be called now & 'parent clicked' will be printed.

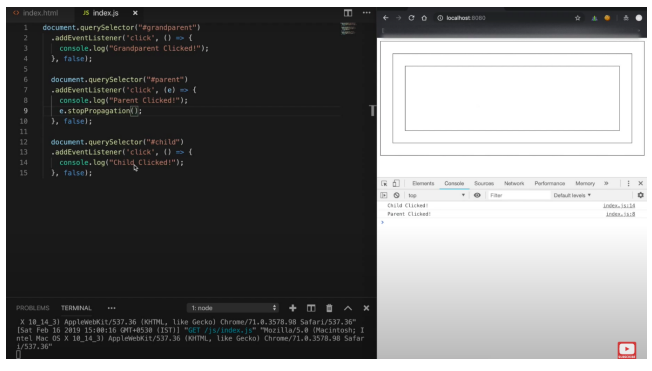


→ Similarly, we can explain this output.
→ onClickGrandparent() is called during event capturing phase & then event trickles down till the last element & so first onClickChild() & then onClickParent() is clicked during bubbling phase.

→ These operations are expensive & cause performance issues. So let's see how we can handle this:

```
document.querySelector("#parent")
.addEventListener('click', (e) => {
  console.log("Parent Clicked!");
  e.stopPropagation();
}, false);//bubbling
```

→ We have a method attached with the event called stopPropagation() & we can stop the event propagation using this.



→ So, now after Event capturing phase, when we reach the innermost element of the Dom tree, then event bubbling phase ~~starts~~ and onChildClick() is run & 'child clicked' is printed. Then onParentClick() is run & 'parent clicked' is printed & when JS encounters stopPropagation() then event propagation is stopped.

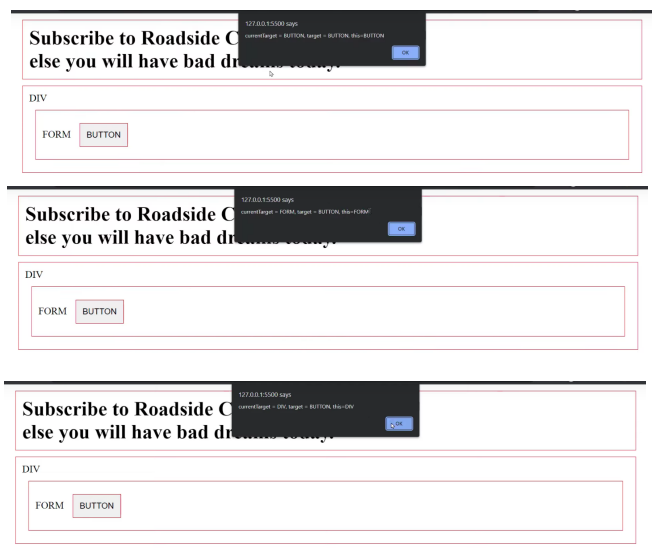
Note: The 3 stages of event propagation are:
Event Capturing > Target > Event Bubbling

→ the complete process of deciding when & in which direction an event is executed is event propagation.

Q: event.target vs this.target vs event.currentTarget:

```
// Ques 3 - event.target vs this.target vs event.currentTarget

const div = document.querySelector("div");
const form = document.querySelector("form");
const button = document.querySelector("button");

div.addEventListener("click", func);
button.addEventListener("click", func);
form.addEventListener("click", func);
          function func(event: any): void
function func(event) {
  alert(
    "currentTarget = " +
      event.currentTarget.tagName +
      ", target = " +
      event.target.tagName +
      ", this=" +
      this.tagName
  );
}
```

o/p:



→ So, when we click on the 'button' event bubbling happens, so, the event bubbles up the hierarchy.

→ Now 'event.currentTarget.tagName' & 'this.tagname point to the html tags they are associated to. But 'event.target.tagName' points to the clicked HTML element only. Because this is where the origin of our bubbling is.