**What is the DOM?**

DOM stands for the Document Object Model. It’s the API for HTML and XML documents. We’re concerned with HTML when looking at what browsers are doing to render the screen.

The HTML elements in a page are put in one big object that can be used by browsers to display elements on the page.

We manipulate the DOM API to manipulate the DOM structure by changing specific elements or nodes in the object. The changes are reflected on the browser screen.

The DOM tree looks something like the following:

The JavaScript document object represents the DOM in the browser.

We can manipulate the DOM tree by finding elements in the DOM tree and then manipulate them by calling the methods or set the properties on the object that’s found.

We can do the by using various methods like document.querySelector, document.getElementById to find one element by CSS selector or ID respectively to return an element or node that we can call methods or set properties on.

Also, we can manipulate a group of objects by document.querySelectorAll to select a group of elements by CSS selector. It returns an array-like object which we can iterate through change things.

**What is Event Propagation?**

Event propagation occurs when an event is fired on a DOM element. The event is also triggered by the parent elements.

There’s the bubbling phase, where the event bubbles up from the originating node to the parent element, grandparent, and all the way up to window.

Then there’s the capturing phase, where the event starts from window all the way down to the originating element that triggers the event.

Altogether, there’re 3 phases:

1. capturing phase — event starts from window then goes down all the way to child elements until it reaches the target element.
2. target phase — event has reached the target element.
3. bubbling — event bubbles up from the target element and goes up all the way to the window.

**What is Event Propagation?**

Event propagation occurs when an event is fired on a DOM element. The event is also triggered by the parent elements.

There’s the bubbling phase, where the event bubbles up from the originating node to the parent element, grandparent, and all the way up to window.

Then there’s the capturing phase, where the event starts from window all the way down to the originating element that triggers the event.

Altogether, there’re 3 phases:

1. capturing phase — event starts from window then goes down all the way to child elements until it reaches the target element.
2. target phase — event has reached the target element.
3. bubbling — event bubbles up from the target element and goes up all the way to the window.

**What is Event Bubbling?**

This is the situation where events start from the element where the event originates from, and then event bubbles up to the parent, grandparent, etc., until it reaches the window object.

For example, if we have the following HTML code:

<div id='grandparent'>  
 <div id='parent'>  
 <div id='start'>  
 Start  
 </div>  
 </div>  
</div>

and the following JavaScript code:

const start = document.querySelector('#start');  
const parent = document.querySelector('#parent');  
const grandparent = document.querySelector('#grandparent');start.addEventListener('click', () => {  
 alert('start clicked');  
 })parent.addEventListener('click', () => {  
 alert('parent clicked');  
 })grandparent.addEventListener('click', () => {  
 alert('grandparent clicked');  
 })document.addEventListener('click', () => {  
 alert('document clicked');  
 })window.addEventListener('click', () => {  
 alert('window clicked');  
 })

When we click Start, we should see all the alerts show in the following order:

1. start clicked
2. parent clicked
3. grandparent clicked
4. document clicked
5. window clicked

This is because the click event from the div with ID start originates from that div, and then it gets bubbled up to the div with the ID parent, then to the div with ID grandparent.

Then the event goes to the document and the window.

addEventListener takes a third argument, which is to set the useCapture option. Is that’s set to true, then events will occur in the capturing phase instead of the bubbling phase, which is the default.

For example, if we have the following code:

const start = document.querySelector('#start');  
const parent = document.querySelector('#parent');  
const grandparent = document.querySelector('#grandparent');start.addEventListener('click', () => {  
 alert('start clicked');  
 }, true)parent.addEventListener('click', () => {  
 alert('parent clicked');  
 }, true)grandparent.addEventListener('click', () => {  
 alert('grandparent clicked');  
 }, true)document.addEventListener('click', () => {  
 alert('document clicked');  
 }, true)window.addEventListener('click', () => {  
 alert('window clicked');  
 }, true)

The alerts will be displayed in the reverse order from what we have before. We can see that capturing is the reverse of bubbling.

**What is Event Capturing?**

Event capturing is the reverse of event bubbling. This means that events starts from the window all the way to the element that triggered the event.

For example, if we have the following HTML code:

<div id='grandparent'>  
 <div id='parent'>  
 <div id='start'>  
 Start  
 </div>  
 </div>  
</div>

and the following JavaScript code:

const start = document.querySelector('#start');  
const parent = document.querySelector('#parent');  
const grandparent = document.querySelector('#grandparent');start.addEventListener('click', () => {  
 alert('start clicked');  
 }, true)parent.addEventListener('click', () => {  
 alert('parent clicked');  
 }, true)grandparent.addEventListener('click', () => {  
 alert('grandparent clicked');  
 }, true)document.addEventListener('click', () => {  
 alert('document clicked');  
 }, true)window.addEventListener('click', () => {  
 alert('window clicked');  
 }, true)

Then when we click Start, we get alerts displayed in the following order:

1. window clicked
2. document clicked
3. grandparent clicked
4. parent clicked
5. start clicked

This is because we passed in true to the 3rd argument of addEventListener, which is the argument to set whether useCapture is enabled or not.

The default of the 3rd argument is false, which lets events bubble up instead of being captured.

Setting it to true changes the event to occur in the capturing phase instead of letting it bubble up.