JavaScript - Events

Index

Events

- An event is a signal that something has happened.

Mouse events:

- `click` – when the mouse clicks on an element (touchscreen devices generate it on a tap).
- `contextmenu` – when the mouse right-clicks on an element.
- `mouseover` / `mouseout` – when the mouse cursor comes over / leaves an element.
- `mousedown` / `mouseup` – when the mouse button is pressed / released over an element.
- `mousemove` – when the mouse is moved.

Keyboard events:

- `keydown` and `keyup` – when a keyboard key is pressed and released.

Events

Form element events:

- `submit` – when the visitor submits a `<form>`.
- `focus` – when the visitor focuses on an element, e.g. on an `<input>`.

Document events:

- `DOMContentLoaded` – when the HTML is loaded and processed, DOM is fully built.

CSS events:

- `transitionend` – when a CSS-animation finishes.

- There are many other events.

Event handlers

- Event handlers make it possible to detect and react to events happening in our web page
- Each event has a type ("keydown", "focus", and so on) that identifies it.
- Most events are called on a specific DOM element and then propagate to that element’s ancestors, allowing handlers associated with those elements to handle them.
- When an event handler is called, it is passed an event object with additional information about the event.
- This object also has methods that allow us to stop further propagation (stopPropagation) and prevent the browser’s default handling of the event (preventDefault).

Event Handler - HTML attribute

- Don’t use this!

```
<button onClick="alert('Hello,old-fashioned event handler!');" >
    Press me
</button>
```

Event Handler - DOM property

- A handler can be assigned using a DOM property on<event>.

```
<input id="elem" type="button" value="Click me">
<script>
   document.getElementById("elem").onclick = function () {
       alert('Thank you');
   };
</script>
```

Event Handler - addEventListener

- Use the addEventListener() method of the element object

```
<input id="elem" type="button" value="Click me"/>
<script>
     function handler1() {
       alert('Thanks!');
     }

    document.getElementById("elem").addEventListener("click", handler1);
                                                                 // Thanks!
</script>
```

Event Handler - addEventListener

- Multiple calls to addEventListener allow it to add multiple handlers, like this:

```
<input id="elem" type="button" value="Click me"/>
<script>
   function handler1() {
       alert('Thanks!');
   }

   function handler2() {
       alert('Thanks again!');
   }

   document.getElementById("elem").addEventListener("click", handler1);
                                                                     // Thanks!
   document.getElementById("elem").addEventListener("click", handler2);
                                                                 // Thanks again!
</script>
```

removeEventListener

- To remove the handler:

```
function handler() {
    alert( 'Thanks!' );
}

input.addEventListener("click", handler);
// ....
input.removeEventListener("click", handler);
```

Event object - Properties

- Some properties of event object:
  - `event.type`
    Event type, here it's `"click"`.
  - `event.currentTarget (=this)`
    The current element that handles the event (the one that has the handler on it)
    `this`, unless the handler is an arrow function, or its `this` is bound to something else.
  - `event.clientX` / `event.clientY`
    Window-relative coordinates of the cursor, for pointer events.
  - `event.target`
    The most nested element where the event happens
  - `event.eventPhase`
    The current phase (capturing=1, target=2, bubbling=3).
- There are more properties.
- Many of them depend on the event type: keyboard events, pointer events…

preventDefault

- There are many default browser actions. For instance:
  - A click on a link – initiates navigation to its URL.
  - A click on a form submit button – initiates its submission to the server.
  - Pressing a mouse button over a text and moving it – selects the text.

- All the default actions can be prevented if we want to handle the event exclusively by JavaScript.
- To prevent a default action – use either event.preventDefault() or return false.
  The second method works only for handlers assigned with on<event>.

preventDefault

- Some properties of event object:

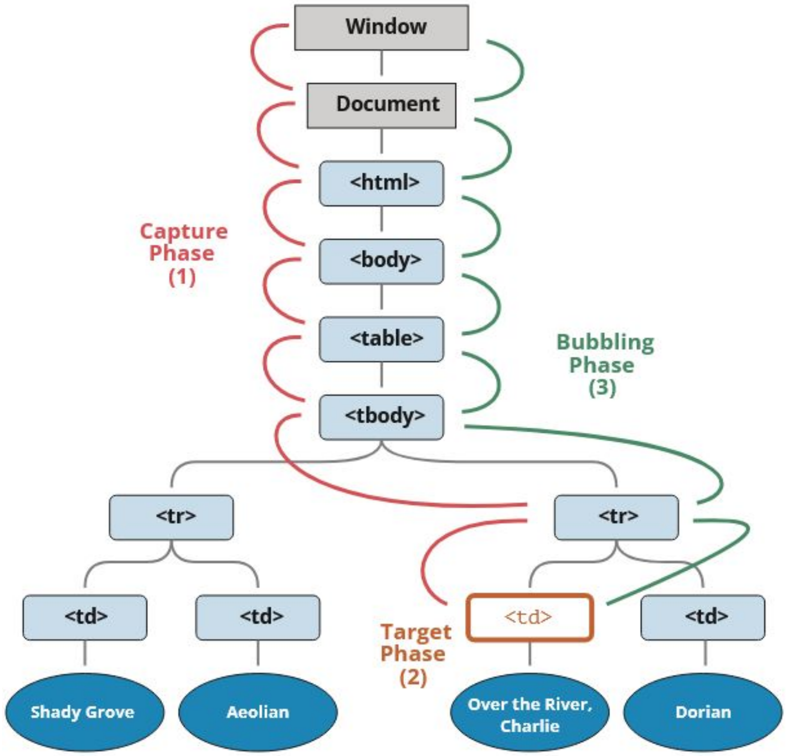
```
<input type="button" value="Click me" id="elem">

<script>
     elem.onclick = function(event) {
      // show event type, element and coordinates of the click
     alert(`${event.type} at ${event.currentTarget}`);
     alert(`Coordinates: ${event.clientX}:${event.clientY}`);
};
</script>
```

Bubbling and capturing

- Bubbling
  - 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.
- Capturing
  - **It is rarely used in real code**
  - It is invisible for us

For a click on <td> the event first goes through the ancestors chain down to the element (capturing phase), then it reaches the target and triggers there (target phase), and then it goes up (bubbling phase), calling handlers on its way.



Bubbling and capturing

- Capturing (***it is rarely used in real code)***

To catch an event on the capturing phase, we need to set the handler capture option to true:

```
elem.addEventListener(..., {capture: true})

// or, just "true" is an alias to {capture: true}
elem.addEventListener(..., true)
```

There are two possible values of the capture option:

If it’s **`false`** **(default)**, then the handler is set on the bubbling phase.

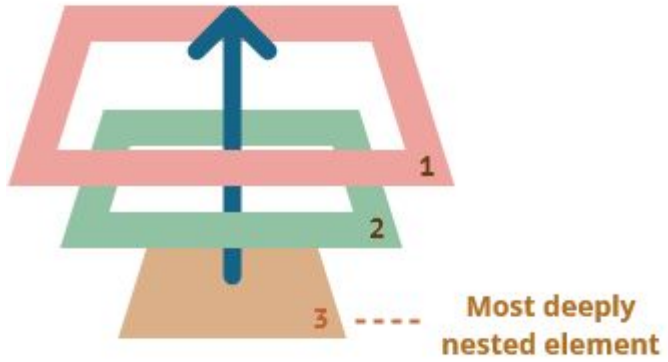
- If it’s true, then the handler is set on the capturing phase.

Bubbling and capturing

- Bubbling

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.

```
<form onclick="alert('form')">FORM
   <div onclick="alert('div')">DIV
       <p onclick="alert('p')">P</p>
   </div>
</form>
```



A click on the inner <p> first runs onclick:

1. On that <p>.
2. Then on the outer <div>.
3. Then on the outer <form>.
4. And so on upwards till the document object.

stopPropagation

- Any event handler can stop the event by calling event.stopPropagation()
- This method is used to prevent the parent element from accessing the event.

```
<div id="box">
   <button className="btn">Register</button>
</div>
<script>
   const btn = document.querySelector('.btn');
   const box = document.querySelector('#box');
   btn.addEventListener('click', function (e) {
       alert('The button was clicked!');
       e.stopPropagation(); // // event onclick in box won't be executed
   });
   box.addEventListener('click', function (e) {
       alert('The box was clicked!');
   });
</script>
```

Event delegation

- Event Delegation is a pattern to handle events efficiently.
- It allows to write cleaner code, and create fewer event listeners with similar logic
- Instead of adding an event listener to each and every similar element, an event listener is added to a parent element and call an event on a particular target using the .target property of the event object.
- Benefits
    - This makes it easier for you to add and remove elements without having to add new or remove existing event listeners.

Event delegation

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

```
 <form id="form">FORM
   <div>DIV
       <p>P</p>
   </div>
</form>

<script>
   form.onclick = function(event) {
       event.target.style.backgroundColor = 'yellow';

       // chrome needs some time to paint yellow
       setTimeout(() => {
           alert("target = " + event.target.tagName + ", this=" + this.tagName);
                                                     // this.tagName is always FORM
           event.target.style.backgroundColor = ''
       }, 0);
   };
</script>
```

Event delegation

❌ Three event listeners for the three buttons.

```
<div>
   <button>Button 1</button>
   <button>Button 2</button>
   <button>Button 3</button>
</div>
const buttons = document.querySelectorAll('button');
buttons.forEach(button => {
   button.addEventListener("click", (event) => {
       console.log(event.target.innerText)
   })
});
```

✅ Just one event listener

Even if an extra button is added

```
<div>
   <button>Button 1</button>
   <button>Button 2</button>
   <button>Button 3</button>
</div>

div.addEventListener("click", (event) => {
   if(event.target.tagName === 'BUTTON') {
       console.log(event.target.innerText);
   }
});
```

Event delegation

- The algorithm:

  1. Put a single handler on the container.

  2. In the handler – check the source element event.target.

  3. If the event happened inside an element that interests us, then handle the event.