JavaScript Day - 15

Events Handling, Event Object

What is an event ?

JavaScript events are like actions that happen based on user input, such as mouse clicks or key presses, and the system responds accordingly. These events can trigger specific functions in JavaScript code. For example, when a user clicks a button, JavaScript can execute a function to display a message.

Here’s a list of the most useful DOM events, just to take a look at:

Common Events

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.

Common Events

Keyboard events:

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

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.

NOTE: There are many other events. We'll get into more details of particular events in upcoming chapters.

Event Handlers

To react on events we can assign a handler – a function that runs in case of an event.

Handlers are a way to run JavaScript code in case of user actions.

There are several ways to assign a handler. Let’s see them, starting from the simplest one.

Event Handlers (HTML Attribute)

A handler can be set in HTML with an attribute named **on<event>**

For instance, to assign a click handler for an input, we can use onclick, like here:

```
1  <input value="Click me" onclick="alert('Click!')" type="button">
```

On mouse click, the code inside onclick runs.

Event Handlers (HTML Attribute)

An HTML-attribute is not a convenient place to write a lot of code, so we'd better create a JavaScript function and call it there.

Here a click runs the function countRabbits():

```
<script>
  function countRabbits() {
    for(let i=1; i<=3; i++) {
      alert("Rabbit number " + i);
    }
  }
</script>

<input type="button" onclick="countRabbits()" value="Count rabbits!">
```

Event Handlers (DOM Attribute)

We can assign a handler using a DOM property **on<event>.**

For instance, **elem.onclick**:

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

If the handler is assigned using an HTML-attribute then the browser reads it, creates a new function from the attribute content and writes it to the DOM property.

So this way is actually the same as the previous one.

Event Handlers (DOM Attribute)

Limitation of the above two methods ?

As there's **only one onclick property**, we can't assign more than one event handler.

Adding another a handler with JavaScript overwrites the existing handler.

Event Handlers (DOM Attribute)

Q1. What will be the value of **this keyword** when using HTML attribute or DOM attribute method for event handling ?

Q2. What will be the value of **this** if we used arrow function instead of function expression ?

CAUTION: Don't use **setAttribute** for handlers. Such a call won't work:

Event Handlers (addEventListener)

The fundamental problem of the aforementioned ways to assign handlers is that we can't assign multiple handlers to one event.

Let's say, one part of our code wants to highlight a button on click, and another one wants to show a message on the same click.

We have two special methods which overcomes the previous limitation with attribute event handlers. **(addEventListener, removeEventListener)**

Event Handlers (addEventListener)

The syntax to add a handler:

```
element.addEventListener(event, handler, [options]);
```

event: Event name, e.g. "click"

handler: The handler function.

Using this method we can add multiple event handlers.

```
elem.addEventListener("click", handler1);
elem.addEventListener("click", handler2);
```

Event Handlers (addEventListener)

To remove the handler, use removeEventListener:

```
element.removeEventListener(event, handler, [options]);
```

Removal requires the same function

To remove a handler we should pass exactly the same function as was assigned.

This doesn't work:

```
elem.addEventListener( "click" , () => alert('Thanks!'));
// ....
elem.removeEventListener( "click", () => alert('Thanks!'));
```

Event Handlers (addEventListener)

The handler won't be removed, because removeEventListener gets another function – with the same code, but that doesn't matter, as it's a different function object.

Here's the right way:

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

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

Please note – if we don't store the function in a variable, then we can't remove it. There's no way to "read back" handlers assigned by addEventListener.

Event Object

To properly handle an event we'd want to know more about what's happened. Not just a "click" or a "keydown", but what were the pointer coordinates? Which key was pressed? And so on.

When an event happens, the browser creates an event object, puts details into it and passes it as an argument to the handler.