Lecture 6.2

Handling Events

Topics

- **Intro to Events**

DOM Levels

What are DOM Levels?

DOM Levels are the versions of the specification for defining how the Document Object Model should work, similarly to how we have HTML4, HTML5, and CSS2.1 specifications.

As of 2020, the most recent spec is DOM Level 4, published in November 2015.

DOM Level 1 defines the core elements of the Document Object Model.
DOM Level 2 extends those elements and adds events.
DOM Level 3 extends DOM lvl 2 and adds more elements and events.

Handling Events

Event Handlers

When an event occurs, you can create an event handler which is a piece of code that will execute to respond to that event. An event handler is also known as an event listener. It listens to the event and responds accordingly to the event fires.

An event listener is a function with an explicit name if it is reusable or an anonymous function in case it is used one time.

There are three ways to assign event handlers.

- HTML event handler attributes
- DOM Level 0 event handlers
- DOM Level 2 event handlers

HTML event handler attributes

Event handlers typically have names that begin with `on`, for example, the event handler for the `click` event is `onclick`.

To assign an event handler to an event associated with an HTML element, you can use an HTML attribute with the name of the event handler.

```
<input type="button" value="Save" onclick="alert('Clicked!')">
```

Disadvantages of HTML event handler attributes

Assigning event handlers using HTML event handler attributes are considered as bad practices and should be avoided as much as possible because of the following reasons:

First, the event handler code is mixed with the HTML code, which will make the code more difficult to maintain and extend.

Second, it is a timing issue. If the element is loaded fully before the JavaScript code, users can start interacting with the element on the webpage which will cause an error.

Dom Level 0 event handlers

Each element has event handler properties such as `onclick`. To assign an event handler, you set the property to a function

```
let btn = document.querySelector('#btn');

btn.onclick = function() {
    alert('Clicked!');
};
```

Dom Level 0 remove event handlers

To remove the event handler, you set the value of the event handler property to `null`:

```
btn.onclick = null;
```

The DOM Level 0 event handlers are still being used widely because of its simplicity and cross-browser support.

Dom Level 2 event handlers

DOM Level 2 Event Handlers provide two main methods for dealing with the registering/deregistering event listeners:

- addEventListener() – register an event handler
- removeEventListener() – remove an event handler

```
btn.addEventListener('click', showAlert);


// remove the event listener
btn.removeEventListener('click', showAlert);
```

Page Load Events

Page Life Cycle

The lifecycle of an HTML page has three important events:

- DOMContentLoaded – the browser fully loaded HTML, and the DOM tree is built, but external resources like pictures <img> and stylesheets may not yet have loaded.
- load – not only HTML is loaded, but also all the external resources: images, styles etc.
- beforeunload/unload – the user is leaving the page.

Page Life Cycle cont..

Each event may be useful:

- DOMContentLoaded event – DOM is ready, so the handler can lookup DOM nodes, initialize the interface.
- load event – external resources are loaded, so styles are applied, image sizes are known etc.
- beforeunload event – the user is leaving: we can check if the user saved the changes and ask them whether they really want to leave.
- unload – the user almost left, but we still can initiate some operations, such as sending out statistics.

```
document.addEventListener('DOMContentLoaded',() => {
    // handle DOMContentLoaded event
});

document.addEventListener('load',() => {
    // handle load event
});

document.addEventListener('beforeunload',() => {
    // handle beforeunload event
});

document.addEventListener('unload',() => {
    // handle unload event
});
```

To handle the page events, you can call the addEventListener() method on the document object:

Mouse Events

Video - Window onLoad Event

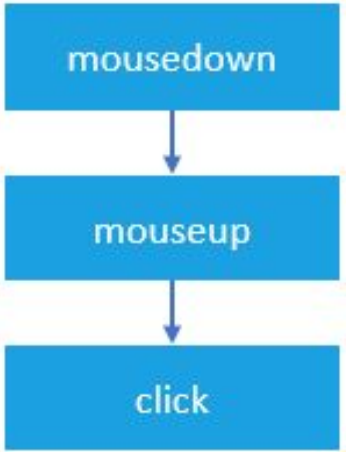
Mouse Events

Mouse events fire when you use the mouse to interact with the elements on the page. DOM Level 3 events define nine mouse events.

mousedown, mouseup, and click

When you click an element, there are no less than three mouse events fire in the following sequence:

1. The mousedown fires when you depress the mouse button on the element.
2. The mouseup fires when you release the mouse button on the element.
3. The click fires when one mousedown and one mouseup detected on the element.



Mouse Events cont..

If you depress the mouse button on an element and move your mouse off the element, and then release the mouse button. The only mousedown event fires on the element.

Likewise, if you depress the mouse button, and move the mouse over the element, and release the mouse button, the only mouseup event fires on the element.

In both cases, the click event never fires.

Best Practice

```
btn.addEventListener('click',(event) => {
    console.log('clicked');
});
```

or you can assign a mouse event handler to the element's property:

```
btn.onclick = (event) => {
    console.log('clicked');
};
```

In legacy systems, you may find that the event handler is assigned in the HTML attribute of the element:

```
<button id="btn" onclick="console.log('clicked')">Click Me!</button>
```

Mouse Events

The mouseover fires when the mouse cursor is outside of the element and then move to inside the boundaries of the element.

The mouseout fires when the mouse cursor is over an element and then move another element.

The mouseenter fires when the mouse cursor is outside of an element and then moves to inside the boundaries of the element.

The mouseleave fires when the mouse cursor is over an element and then moves to the outside of the element's boundaries.

Detecting Mouse Buttons

The event object passed to the mouse event handler has a property called button that indicates which mouse button was pressed on the mouse to trigger the event.

The mouse button is represented by a number:

Keyboard Events

Keyboard Events

When you interact with the keyboard, the keyboard events are fired. There are three main keyboard events:

keydown – fires when you press a key on the keyboard and it fires repeatedly while you holding down the key.
keyup – fires when you release a key on the keyboard.
keypress – fires when you press a character keyboard like a,b, or c, not the left arrow key, home, or end keyboard, ... The keypress also fires repeatedly while you hold down the key on the keyboard.

The keyboard events typically fire on the text box, through all elements support them.

The following illustrates how to register keyboard event listeners:

```
msg.addEventListener("keydown", (event) => {
    // handle keydown
});

msg.addEventListener("keypress", (event) => {
    // handle keypress
});

msg.addEventListener("keyup", (event) => {
    // handle keyup
});
```

Handling keyboard events

Video - OnClick Event

Scoll Events

Scroll Events

When you scroll a document or an element, the scroll events fire. You can trigger the scroll events in the following ways, for example:

- Using the scrollbar manually
- Using the mouse wheel
- Clicking an ID link
- Calling functions in JavaScript

To register a scroll event handler, you call the addEventListener() method on the target element, like this:

```
targetElement.addEventListener('scroll', (event) => {
    // handle the scroll event
});
```

or assign an event handler to the onscroll property of the target element:

```
targetElement.onscroll = (event) => {
    // handle the scroll event
};
```

Typically, you handle the `scroll` events on the `window` object to handle the scroll of the whole webpage.

```
window.addEventListener('scroll',(event) => {
    console.log('Scrolling...');
});
```

Or you can use the `onscroll` property on the `window` object:

```
window.onscroll = function(event) {
    //
};
```

Scrolling the document

The onscroll property of the window object is the same as document.body.onscroll and you can use them interchangeably, for example:

```
document.body.onscroll = null;
console.log(window.onscroll); // null
```

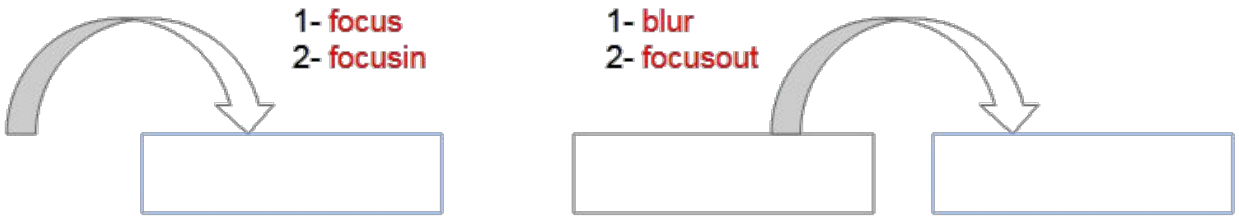
Focus Events

Focus Events

The focus events fire when an element receives or loses focus. These are the two main focus events:

focus fires when an element has received focus.
blur fires when an element has lost focus.

The focusin and focusout fire at the same time as focus and blur, however, they bubble while the focus and blur do not.



Focus Events

The following elements are focusable:

The **window** gains focus when you bring it forward by using Alt+Tab or clicking on it and loses focus when you send it back.

Links when you use a mouse or a keyboard.

Form fields like input text when you use a keyboard or a mouse.

Elements with **tabindex**, also when you use a keyboard or a mouse.

Handling Focus Events

To register a foucs event handler, you call the addEventListener() method on the target element, like this:

```
target.addEventListener('focus', (e) => {
    console.log('foucs');
});

target.addEventListener('blur', (e) => {
    console.log('blur');
}
```