WEB230: JavaScript 1

Module 6: Handling Events

Events

- events are interactions with our page
- often initiated by the user
- we can't predict when they will happen

Event Handlers

- JavaScript code that runs when an event occurs
- written as a function
- this function is passed to a method

Events and DOM Nodes

- every DOM Element node can have events associated with it
- use `.addEventListener()`
- first argument is the event name such as `'click'`
- second argument is a function (the event handler)

```
const button = document.querySelector('button');
button.addEventListener('click', function() {
  alert('Button clicked.');
});
```

Depricated Ways to add Event Handlers

- There are older ways to add event handlers
- `.onclick` property on a selected element
- `onclick=""` attribute in HTML
- JS properties and HTML attributes exist for other events
  - eg `onload`, `onmouseover`, `onkeydown`, etc.
- DO NOT USE THESE EVER!
- always use `.addEventListener()`

Deleting an Event Handler

- use a named function
- this provides a function reference that we can pass to `.removeEventListener()`

```
const button = document.querySelector('button');
function once() {
  alert('Done.');
  button.removeEventListener('click', once);
}
button.addEventListener('click', once);
```

The `event` Object

- event handlers can accept a parameter called the event object
- this object has information about the event
    - for example, which element was clicked on
    - which button or key was pressed
- properties and methods vary depending on the type of event
- this parameter is usually called `event` or simply `e`

Key Events

- `keydown` and `keyup` events
- `keydown` will repeat if held
- `event.key` holds a string with the value that the key would type
- boolean properties for modifier keys:
    - `event.shiftKey`
    - `event.ctrlKey`
    - `event.altKey`
    - `event.metaKey` (Windows key or Mac Command key)
- event occurs on element that has focus (or document.body)
- if you want to capture all keystrokes, use `window.addEventListener()`
    - `window.` is optional since it is the global object
- Note: the `keypress` event is depricated

Key Event Properties

- `event.key` (String) The key value of the key represented by the event. If the value has a printed representation, this value is that character (Eg. "a"). Otherwise, it describes the key (Eg. "Escape").
- `event.code` (String) Holds a string that identifies the physical key being pressed. The value is not affected by the current keyboard layout or modifier state, so a particular key will always return the same value.
- there are other depricated properties that should be avoided

```
window.addEventListener('keydown', function(event) {
  console.log('Key pressed:', event.key);
});
```

- `event.repeat` (Boolean) `true` if the key is being held down such that it is automatically repeating
    - can be used to avoid repeatedly running the event handler

```
window.addEventListener('keydown', function(event) {
  if (event.repeat) { return; }
  console.log('Key pressed:', event.key);
});
```

Mouse Clicks

- `mousedown`, `mouseup`, `click`, and `dblclick` events
- `event.clientX` and `event.clientY` properties give exact location

Mouse Button Event Order

1. `mousedown`
2. `mouseup`
3. `click`
4. `dblclick` - if applicable
    - `dblclick` will repeat the previous three twice

- `event.button` takes into account user customization
    - 0: Main button pressed, usually the left button or the un-initialized state
    - 1: Auxiliary button pressed, usually the wheel button or the middle button (if present)
    - 2: Secondary button pressed, usually the right button
    - 3: Fourth button, typically the Browser Back button
    - 4: Fifth button, typically the Browser Forward button

Mouse Motion

- `mousemove` event every time the mouse moves
- `mouseover` or `mouseout` event equivalent to CSS `:hover`

Scroll Events

- `scroll` event when page scrolls
- fired every time the page is scrolled
- `window.scrollX` and `window.scrollY` for scroll position

Focus Events

- `focus` and `blur`
- when an element is selected it is a `focus` event
- when it loses focus a `blur` event is fired
- most often used with form fields
- does not propogate

Load Event

- `load` event fires on the `window` object when the window finishes loading the page
- often used to schedule initialization actions that require the DOM
- element that load external files, such as images, also have a load event
- Note: window load is no longer required since the `defer` attribute was added for the script tag

Timers

- `setTimeout` to run a function after an amount of time
- schedules a function to be called in a specified amount of time
- `clearTimeout` can be used to cancel it
- `setInterval` and `clearInterval` is similar but repeats every specified time interval

```
const button = document.querySelector('button');
const list = document.querySelector('ul');
let interval;
button.addEventListener('click', function(event){
  if(interval) {
    clearInterval(interval);
  } else {
    interval = setInterval(function(){
      let item = document.createElement('li');
      item.textContent = 'New item';
      list.appendChild(item);
    },1000);
  }
});
```

Script Execution Timeline

- no two scripts can run at the same time
- each peice of code (often functions) will wait for others to finish
- web workers (not covered in this course) provide a way to do something while other things run

Propagation

- if an event occurs on a child element it will trigger the event handler on the parent element
- if both have handlers the more specific one runs first
- `event.stopPropogation()` method on the event object can stop this

Delegation

- an event handler can be placed on the parent element to handle the events on child elements

`target` Property

- most events have an `event.target` property
- this is the element that the event occurred on
- often used to delegate event handling to parent element

Default Actions

- some element have default actions
  - such as a form being submitted to a server or a link being followed
- the event handler runs before the default action
- `event.preventDefault()` method can stop the default action

Summary

- event handlers make it possible to detect and react to external events
- each event has a type - eg. 'click'
- events *propagate* to their parent elements

    - `event.stopPropagation()`
- some elements have default actions
    - `event.preventDefault()`
- only one piece of JavaScript can run at once

Reference

- MDN Events (https://developer.mozilla.org/en-US/docs/Web/Events)