DECT-KS© TOOLS

前端網絡開發人員課程
(二) 進階網絡程式設計

6. JS DOM VI: Events II

Presented by Krystal Institute

erb 僱員再培訓局 Employees Retraining Board

SCOPE School of Continuing and Professional Education 專業進修學院

KRYSTAL INSTITUTE DIGITAL ECONOMY CORE TECHNOLOGY

Learning Objective

- Understand more types of events
- Know how to use said events with Javascript

Content

6.1
Revise on the previous lesson

6.2
Events pt.2

6.1 Revise on the previous lesson

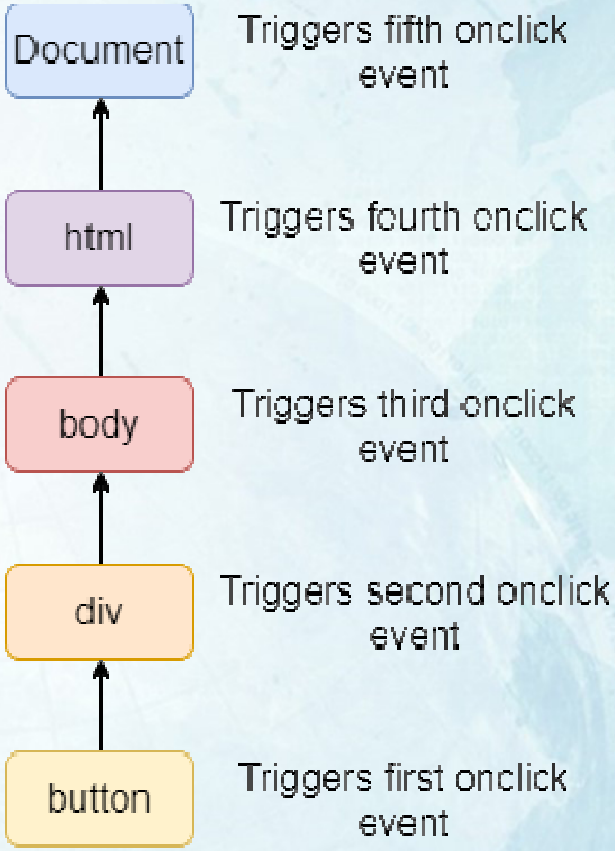
JavaScript Events

- Events are actions happening through interactions from the user in the webpage

- An event listener "listens" to a specific event that happens throughout operation, and can be used to trigger functions with it

Event Models

- There are 2 main models
- Event bubbling triggers events starting from the most specific element, working its way up the DOM Tree
- Event capturing is the complete opposite



Event Handling

- There are 3 ways of adding event listeners

- HTML event attribute is used inside the element attribute (e.g. onclick)
- DOM Level 0 Event Handlers are the most reliable way of adding a event listener, and it is used often by developers for its simplicity
- DOM Level 2 Event Handlers are a longer way to adding a event listener, but it also allows for the same event to be added multiple times

Page Load Events

- There are 4 events that triggers when the user enters / leaves a page

- DOMContentLoaded is triggered after the DOM Tree has been created but the external CSS or scripts has not been loaded
- load is triggered after everything has loaded

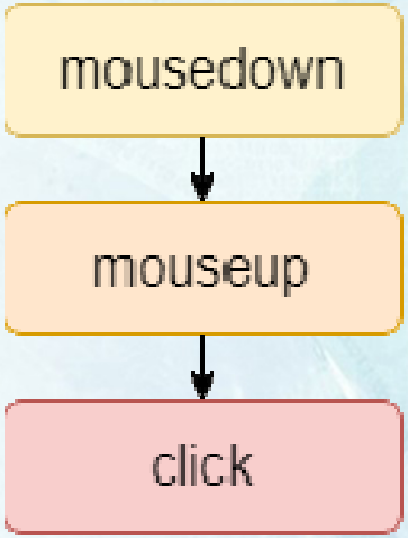
- Load events allows for referencing variables that are assigned or created later

Page Load Events

- beforeunload triggers before the page and resources are unloaded
- unload triggers after everything has been unloaded
- Unload Events allows for collecting and saving analytic data before the user closes the browser

Mouse Events

- Mouse events triggers when you use the mouse to interact with elements in the page
- mousedown, mouseup and click all represents different parts of a mouse click
- click only triggers after a mouseup and a mousedown event and does not recognize buttons other than the left mouse button



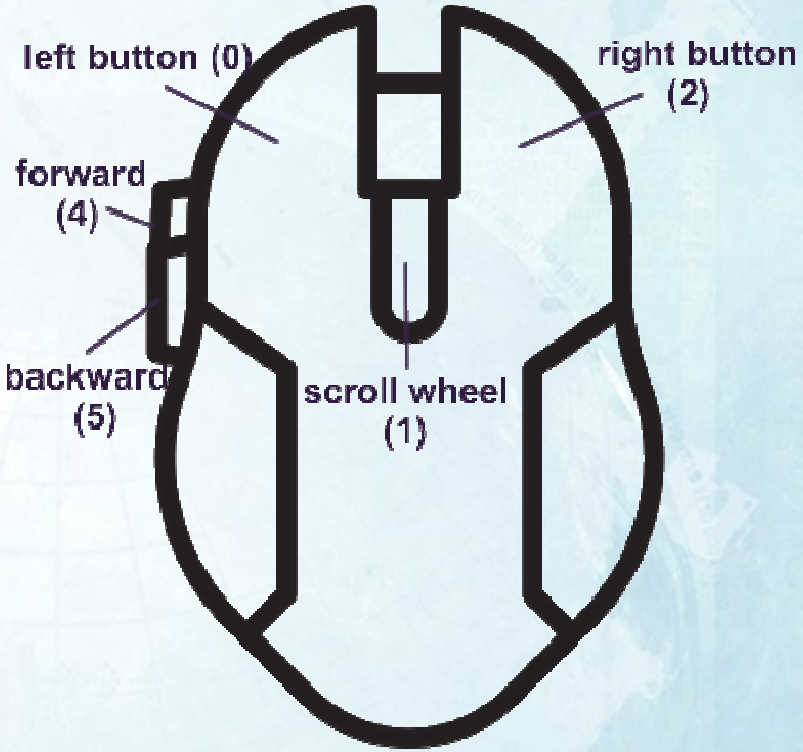
Mouse Events

- dblclick triggers after 2 loops of click events
- mousemove triggers every time your mouse moves
- mouseover / mouseout triggers when the mouse enters / leaves the element as well as all of its descendants
- mouseenter / mouseleave triggers when the mouse enters / leaves the element

Mouse Event Handling

- Event.button has 5 values, and returns the value representing which button the user has clicked

- screenX / screenY returns the coordinate of the mouse cursor in relation to the monitor screen
- clientX / clientY returns the coordinate of the mouse cursor in relation to the web broswer



6.2 Events pt.2

Keyboard Events

- Interacting with the keyboard triggers keyboard events
- Keyboard events typically triggers on text boxes, ,but can work on all kinds of elements
- There are 3 types of keyboard events

Keyboard Events

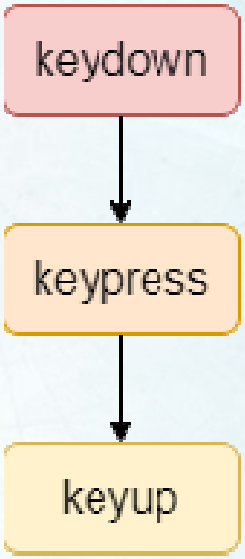
- keydown triggers when you press the key – it works similar to mousedown

- keyup triggers when you release the key – it works similar to mouseup

```
<body>
    <input type="text" id="text">
    <div id="display"></div>
<script>
    let text = document.querySelector("#text")
    let display = document.querySelector("#display")
    text.addEventListener("keydown", function(event) {
        display.textContent = text.value
    });
</script>
</body>
```

Keyboard Events

- keypress triggers when you press character keys like a,b or c
- Arrow keys, home key, end key…those will not trigger keypress event
- It triggers constantly as you're holding down the key



Keyboard Events

- There are 2 important properties of the keyboard event:
- event.key returns the character that has been pressed (z)
- event.code reutrns the physical key code (KeyZ)

```
<body>
    <input type="text" id="text">
    <div id="display"></div>
<script>
    let text = document.querySelector("#text")
    let display = document.querySelector("#display")
    text.addEventListener("keydown", function(event) {
        display.textContent = event.key + "    " + event.code
    });
</script>
</body>
```

Scroll Events

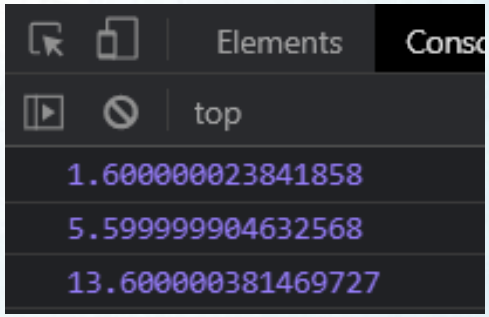
- Scrolling happens when you…
- Use the scroll bar
- Use the mouse wheel
- Click an ID link
- Call functions in JS
- scroll event handler can be added using addEventListener or onscroll property

```
<body>
    <div style="height: 2000px"></div>
<script>
    window.addEventListener("scroll", function(event) {
        console.log("scrolling...");
    });
    window.onscroll = function() {
        console.log("scrolling again...");
    };
</script>
</body>
```

Scroll Events

- scrollTop and scrollLeft are used to check the offset of the scrolling
- If no scrolling is performed, the value of scrollTop and scrollLeft is 0
- The returned value is a decimal, use math.round() to round them off

```
<body>
    <div style="height: 2000px"></div>
<script>
    window.addEventListener("scroll", function(event) {
        let div = document.querySelector("body")
        console.log(div.scrollTop)
    });
</script>
</body>
```



Scroll into View

- scrollIntoView allows for elements that are out of view to scroll into view

- It accepts a alignToTop Boolean (true by default), determining if the element will be aligned at the top of the scrollable area

```
element.scrollIntoView(alignToTop);
```

Scroll into View Activity

- Activity: create a webpage that contains a <ul> with multiple <li>, enough to make the webpage scrollable (add some
 to make it easier)

- Assign id to an <li> element that is outside of view
- add a button on top

```
    <style>
        li {
            font-size: 144px;
        }
    </style>
</head>
<body>
    <ul>
        <button type="button" onclick="toview()">
        Scroll into view</button>
        <li>Python</li>
        <li>JS</li>
        <li>Java</li>
        <li>HTML</li>
        <li>C++</li>
        <li id="special">C#</li>
        <li>Vue</li>
        <li>Go</li>
        <li>VBA</li>
        <li>Assembly</li>
        <li>Julius</li>
    </ul>
```

Scroll into View Activity

- In the function, use scrollIntoView on button click

```
<script>
    function toview() {
        let special = document.querySelector("#special")
        special.scrollIntoView()
    }
</script>
```

Focus Events

- The focus event is triggered when an element receives or loses focus

- focus happens when…
- You use alt-tab to get to a window
- Clicking on windows or links
- Clicking on form fields

Focus Events

- There are 2 types of focus events
- focus triggers when an element receives focus
- blur triggers when an element has lost focus

```
<body>
    <form>
        <label>Username</label>
        <input type="text"><br>
        <label>Password</label>
        <input type="password">
    </form>
<script>
    let input = document.querySelector("input[type='text']")
    input.addEventListener("focus", function() {
        input.style.backgroundColor = "yellow";
    });
    input.addEventListener("blur", function() {
        input.style.backgroundColor = "initial";
    });
</script>
</body>
```

Event Delegation

- In a menu list, you may add a click event on every menu item

```
<body>
    <ul id=“menu">
        <li><a id="home">Home</a></li>
        <li><a id="about">About</a></li>
        <li><a id="services">Services</a></li>
    </ul>
<script>
    let home = document.getElementById("home");
    home.onclick = function() {console.log("Home was clicked!")};
    let about = document.getElementById("about");
    about.onclick = function() {console.log("About was clicked!")};
    let services = document.getElementById("services");
    services.onclick = function() {console.log("Services was clicked!")};
</script>
</body>
```

Event Delegation

- However, this may cause some issues:
- Each event handler is a function and an object that takes up memory
- It takes time to assign event handlers to each and every element

- The more event handlers there is, the lower the performance of the website is

Event Delegation

- Instead of having to assign event handlers on every <li>, you can just assign a single event handler on the <ul> and handle all click events at once

- Handling all events of one type give better performance, and takes less time to load up the page

Event Delegation Activity

- Activity: using the <ul> from before, add a click that handles all
- Add the click event listener on
- Using event.target and switch, allow the click event to handle all <li> at once

```
let menu = document.getElementById("menu");
menu.addEventListener("click", function(event) {
    switch (event.target.id) {
        case "home":
            console.log("Home was clicked");
            break
        case "abbout":
            console.log("About was clicked");
            break
        case "services":
            console.log("Services was clicked");
            break
    };
});
```

Dispatch Events

- Events are generally triggered by user actions like mouse clicks
- However, they can be triggered from code as well

```
element.dispatchEvent(event);
```

- They are generally useful for automatic testing of the website

Dispatch Events

- To dispatch an event, you need to create a new event first
- Use an Event constructor
- Type defines the event type, click is one of the examples
- Options have 2 properties: bubbling, if true, the event bubbles
- cancelable: if the event can be cancelled

```
let event = new Event(type, [,options]);
```

```
let div = document.querySelector("div");
div.addEventListener("click", function(event) {
    alert(event.target.textContent)
});
let event = new Event("click")
```

Dispatch Events

- Use dispatchEvent to dispatch the click event

- The event will be dispatched when it is loaded, meaning it assumes the user have already clicked on the once

```
<body>
    <div>This is a wrapper div
        <p>Clicking on this will return p instead
        of div if using event.target</p>
    </div>
<script>
    let div = document.querySelector("div");
    div.addEventListener("click", function(event) {
        alert(event.target.textContent)
    });
    let event = new Event("click");
    div.dispatchEvent(event);
</script>
</body>
```

Dispatch Events

- event.isTrusted property can be used to determine if an event is triggered by user action of JS code

- It returns true if it is triggered from user actions

```
<script>
    let div = document.querySelector("div");
    div.addEventListener("click", function(event) {
        alert(event.isTrusted)
    });
    let event = new Event("click");
    div.dispatchEvent(event);
</script>
```

Mutation Observers

- MutationObserver is an object that observes a DOM element and fires a callback when it detect changes
- Useful for detecting and removing extra unwanted elements that were added externally, by third party APIs or systems

Mutation Observers

- For instance, you implemented a third party API and when running, it creates a display advertisement on your website!

- Mutation Observers can be used to detect such new elements and remove it

```
<div id="main"></div>
<div id="API">
    <script>//API code</script>
    <div id="ad"><!-- Advertisement created by API --></div>
</div>
```

Mutation Observers

- To create a mutation observer, you need to create a mutationObserver object

```
function callbackfunc(mutation) {
    //
};
let observer = new MutationObserver(callbackfunc);
```

- It accepts one argument – the function that will be called when mutationObserver triggers

Mutation Observers

- Lastly, call the observer() method on the mutation observer to observer an element
- The method accepts 2 arguments: the TargetNode that the observer observes, and the observer options
- The disconnect() method stops the observer from observing the node

```
<script>
    function callbackfunc(mutation) {
        //
    };
    let observer = new MutationObserver(callbackfunc);
    observer.observe(document.getElementById("API"), ObserverOptions)
</script>
```

Mutation Observers

- In the callback function, all the changes will be passed as the first argument of that function, MutationRecord

- There is various properties of the mutation record

Mutation Observers

- The type property has 3 types:
- “attributes” – attributes are changed
- “CharacterData” – data (often text) is modified
- “ChildList” – child elements are added / removed

```
function callbackfunc(mutation) {
    console.log(mutation.type);
    let target = mutation.target;
};
```

- The target property returns the text node if the type is “CharacterData”, otherwise an element will be returned

Mutation Observers

- addedNodes / removedNodes returns nodes that are added / removed
- attributeName returns the name of the changed attribute
- oldValue returns the previous value of a text input or a attribute

Mutation Observers

- Observeroptions determines the condition of the detection, it is an object with Boolean options:
- childList – if observer reacts to changes in the direct children of the node

```
observer.observe(document.getElementById("API"), {
    childList: true,
    subtree: true
});
```

Mutation Observers

- attributes – if attributes will be observed
- attributeFilter – an array of attribute names, only those in the array will be observed
- characterData – if the node.data (text content) will be observed

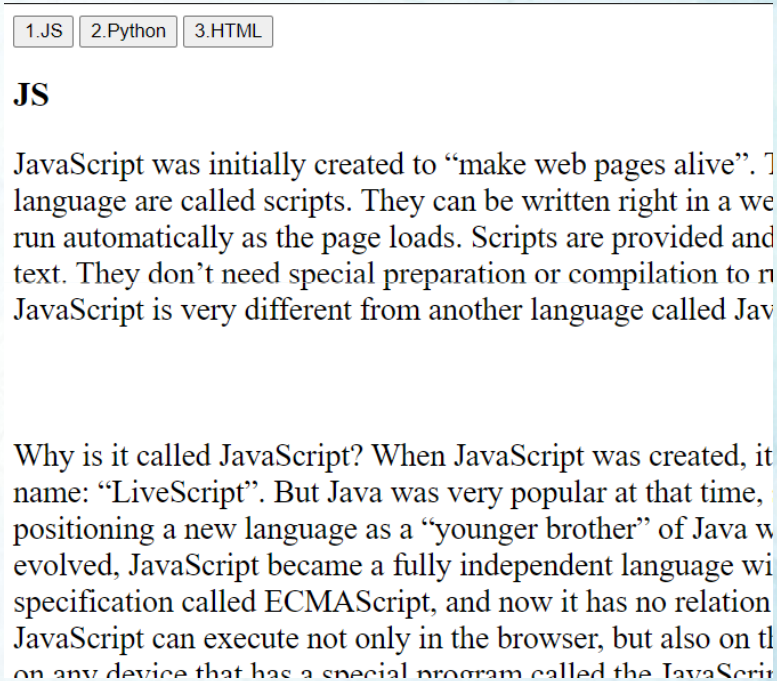
```
observer.observe(document.getElementById("API"), {
    attributes: true,
    attributeFilter: true,
    characterData: true
});
```

Exercise

- Create a website with multiple sections containing headings and text
- Add various buttons on top representing different sections that when clicked, will scroll into it's respective section
- Add Event Handlers that when the user types a number (e.g. 3), it will scroll into that section (section 3)
- Hint: use
 to expand the length of the text, and event delegation to shorten the code
- Finish the Exercise by the end of the lesson

Exercise Example

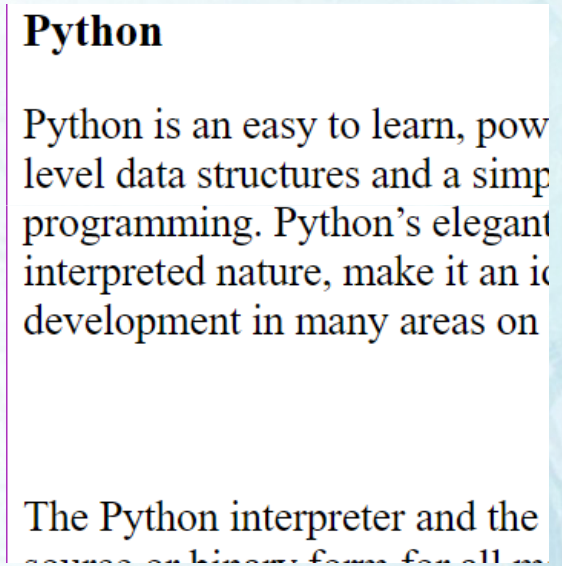
- There are 3 buttons showing each section's heading, along with the heading and the text spanning over the whole page



Exercise Example

- Upon clicking on of the buttons (e.g. the second one), the second section will be aligned at the top of the page

- Upon typing 1, 2 or 3 on your keyboard, it will align the respective sections to the top of the page as well



References

- Use these if you need more explanations!
- https://www.javascripttutorial.net/es6/
- https://javascript.info/

- Use this if you need more specific answers!
- https://stackoverflow.com/