前端網絡開發人員課程
(二) 進階網絡程式設計

7. JS DOM VII: Web Forms

Presented by Krystal Institute

Learning Objective

- Understand how to use JS on HTML forms, and handling different input events
- Know how to create a functioning dynamic website

Content

7.1 Revise on the previous lesson

Keyboard Events

- Interacting with the keyboard triggers keyboard events
- There are 3 main types of events:
- keydown triggers when a key was pressed
- keyup triggers when a key was released
- Keypress triggers when characters keys are pressed, and is constant as long as you're holding the key

Keyboard Events

- There are 2 important properties of the keyboard event:
- event.key returns the character that has been pressed
- event.code returns the keycode of the key

Scroll Events

- The scroll event triggers when you use your mouse wheel or the scroll bar
- event.scrollTop / scrollLeft are used to check the offset of the scrolling

- scrollIntoView scolls elements into view, accepts a alignToTop Boolean argument by default that aligns the top of the element to the top of the scrollable area

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 and Dispatching

- It is possible to assign an event listener on the parent elements of multiple child elements
- It has better performance handling one event handler than multiples of the same event handler

- dispatch triggers chosen events from the code

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

- It is useful for automatic testing of the website
- event.isTrusted ius used to determine if an event is triggered by user actions

Mutation Observers

- MutationObserver observes changes in the DOM Tree and fires a callback when it detects any changes
- It can be used to observe a specific Element, along with some options

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

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

7.2 JavaScript Forms

Forms Recap

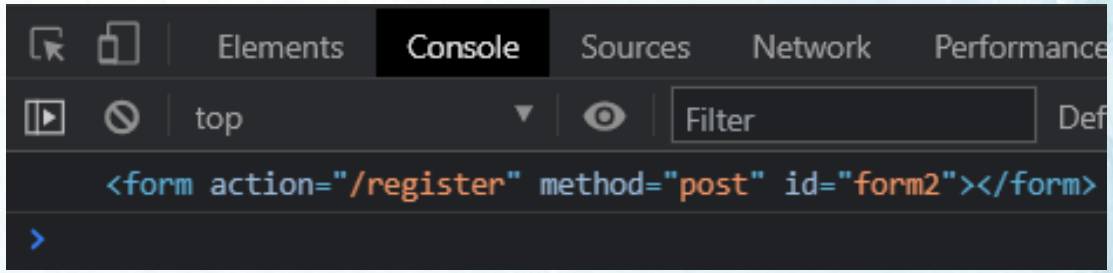
- To create a form in HTML, you use the <form> element
- The action attribute is the url that will process the form submission
- the method attribute is the HTTP method of sending the form data
  - post sends data to the server as a body
  - get sends data through the URL

```
<form action="/login" method="post" id="form"></form>
```

Forms

- An HTML document can have multiple forms
- document.forms returns a collection of forms on the document
- Use index to locate a specific form in the collection of forms

```
<body>
    <form action="/login" method="post"></form>
    <form action="/register" method="post"></form>
<script>
    let formlist = document.forms
    console.log(formlist[1])
</script>
</body>
```



Forms

- Typically, we use buttons to handle submitting form data to the server

- When we click on the submit button or press enter, the submit event triggers and the form data is then sent to the server

```
<form action="/login" method="post" id="form1">
    <input type="text" name="username">
    <button type="submit">Submit</button>
</form>
```

Forms

- We can utilize the submit event and validate the form data before sending it to the server
- You can use addEventListener to apply the submit event to the form

```
<body>
    <form action="/login" method="post" id="form">
        <input type="text" name="username">
        <button type="submit">Submit</button>
    </form>
<script>
    let form = document.querySelector("#form");
    form.addEventListener("submit", function() {
        // Validate data here
    });
</script>
</body>
```

Forms

- preventDefault is a method that cancels an event
- If the form is invalid, use preventDefault to cancel the submit event and stop the form data to be sent

```
<body>
    <form action="/login" method="post" id="form">
        <input type="text" name="username">
        <button type="submit">Submit</button>
    </form>
<script>
    let form = document.querySelector("#form");
    form.addEventListener("submit", function() {
        // Validate data here
        // form data is invalid
        event.preventDefault()
    });
</script>
</body>
```

Forms

- To submit the form after validating the data, use form.submit()
- form.submit do not trigger the submit event, so validate the data before using this method

```
<body>
    <form action="/login" method="post" id="form">
        <input type="text" name="username">
        <button type="submit">Submit</button>
    </form>
<script>
    let form = document.querySelector("#form");
    form.addEventListener("submit", function() {
        // Validate data here
        // form data valid
        form.submit()
    });
</script>
</body>
```

Forms

- You can access elements inside the form using form.elements
- It returns a list of all the elements in order, use index or name to get it

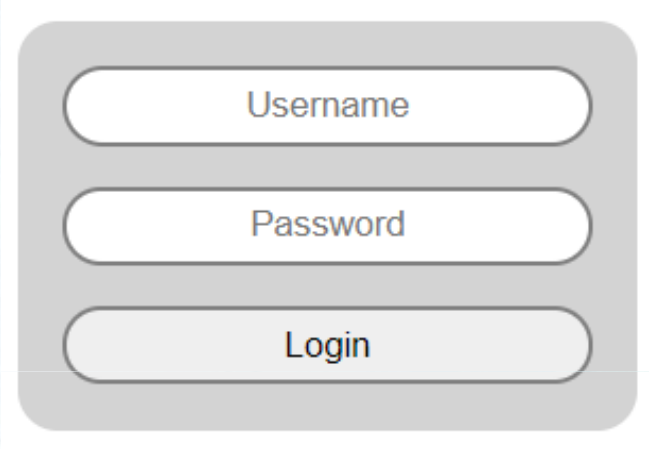
```
<script>
    let form = document.querySelector("#form");
    let textinput = form.elements["username"];
    let button = form.elements[1];
</script>
```

- Use the value property of the element to access an elements' property

```
<script>
    let form = document.querySelector("#form");
    form.addEventListener("submit", function() {
        let textinput = form.elements["username"];
        let button = form.elements[1];
        let textans = textinput.value;
        console.log(textans);
        event.preventDefault();
    });
</script>
```

Forms Activity

- Activity: make a login page with validation
- Create text inputs like username and password
- Create the submit button
- Decorate it with CSS styles




```
<form action="/login" method="post" id="form">
    <input type="text" name="username" placeholder="Username">
    <input type="password" name="password" placeholder="Password">
    <button type="submit">Login</button>
</form>
```

Forms Activity

- Lets set admin and 1234 as the correct username and password
- Create a validate function that is called when the user presses submit button
- The function checks if the username and password input has the correct value
- Set border color to initial if inputs are right

```
let form = document.querySelector("#form");
let username = form.elements[0]
let password = form.elements[1]
form.addEventListener("submit", function(event) {
    if (username.value == "admin") {
        username.style.borderColor = "initial";
        if (password.value == "1234") {
            password.style.borderColor = "initial";
            form.submit()
        } else {
            Invalidinput("password");
            event.preventDefault();
        }
    } else {
        Invalidinput("username");
        event.preventDefault();
    };
});
```

Forms Activity

- Create a function for invalid inputs
- Set the border color to red and its placeholder to an error message
- Add arguments on the function to specify which part of the form is incorrect
- Use preventDefault to stop the form from submitting

```
function Invalidinput(input) {
    if (input == "username") {
        username.style.borderColor = "red";
        username.value = "";
        username.placeholder = "Invalid username";
    } else {
        password.style.borderColor = "red";
        password.value = "";
        password.placeholder = "Invalid password";
    };
};
```

7.3 Special Inputs

Radio Button

- Recap: Radio buttons are checkboxes that only allow one selection over multiple buttons with the same name

- Use input type radio to use a radio button, and use the name attribute to group it together

```
<input type="radio" name="genders" value="M">Male
<input type="radio" name="genders" value="F">Female
<button onclick="checksel()">Submit</button>
```

Handling Radio Buttons

- To check which button is selected, we can use a for loop over every radio button in the same name group

```
let genders = document.querySelectorAll("input[name='genders']");
function checksel() {
    let selectedvalue;
    for (let i=0; i < genders.length; i++) {
        if (genders[i].checked) {
            selectedvalue = genders[i].value;
        };
    };
    console.log(selectedvalue)
};
```

Checkbox

- Recap: Checkbox is a selectable box input
- To check the state of range of checkboxes, use a for loop similar to handling radio buttons, but use an array to store the selection as it can have multiple boxes checked

```
<input type="checkbox" name="color" value="red">Red
<input type="checkbox" name="color" value="blue">Blue
<input type="checkbox" name="color" value="green">Green
<input type="checkbox" name="color" value="yellow">Yellow
<button onclick="checksel()">Submit</button>
```

```
let colors = document.querySelectorAll("input");
function checksel() {
    let selectedvalue = [];
    for (let i=0; i < colors.length; i++) {
        if (colors[i].checked) {
            selectedvalue.push(colors[i].value);
        };
    };
    console.log(selectedvalue)
};
```

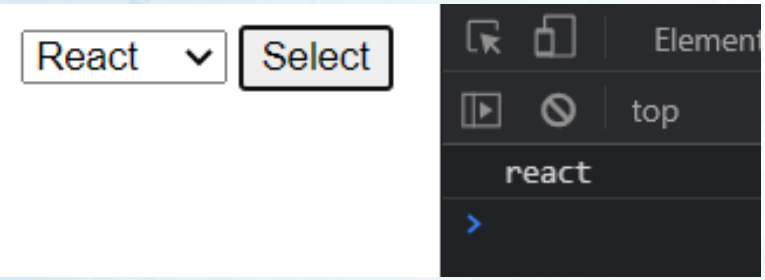
Select Box

- Recap: a <select> element contains a list of options for users to choose from
- To create a select box, create a <select> element and put <option> element inside with values

```
<select id="selection">
    <option value="js">JS</option>
    <option value="react">React</option>
    <option value="angular">Angular</option>
    <option value="node">Node</option>
</select>
```

Select Box

- Using element.value on the select element will return the selected box's value




```
<body>
    <select id="selection">
        <option value="js">JS</option>
        <option value="react">React</option>
        <option value="angular">Angular</option>
        <option value="node">Node</option>
    </select>
    <button onclick="checksel()">Select</button>
<script>
    let selection = document.querySelector("#selection");
    function checksel() {
        console.log(selection.value)
    };
</script>
</body>
```

7.4 Event Handling

Change Events

- The change event triggers when an element has completed changing
- Attach the change event listener like before, or use the onchange attribute, on an input element
- The change event of the input element only triggers when it loses focus, but not when you're typing

```
<body>
    <input type="text" id="input" name="input">
<script>
    let input = document.querySelector("#input")
    addEventListener("change", function() {
        //
    })
</script>
</body>
```

Change Events Activity

- Activity: create a input text that displays it's value on another when you finished typing
- Create the input element
- Add the change event onto the element
- Display the value of the input on a

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

Change Events: Special inputs

- For radio buttons, the change event fires after you select one of the boxes
- For checkboxes, the change event triggers after selection of the checkbox, selecting or unselecting it counts
- For select boxes, the change event triggers after selection

Input Events

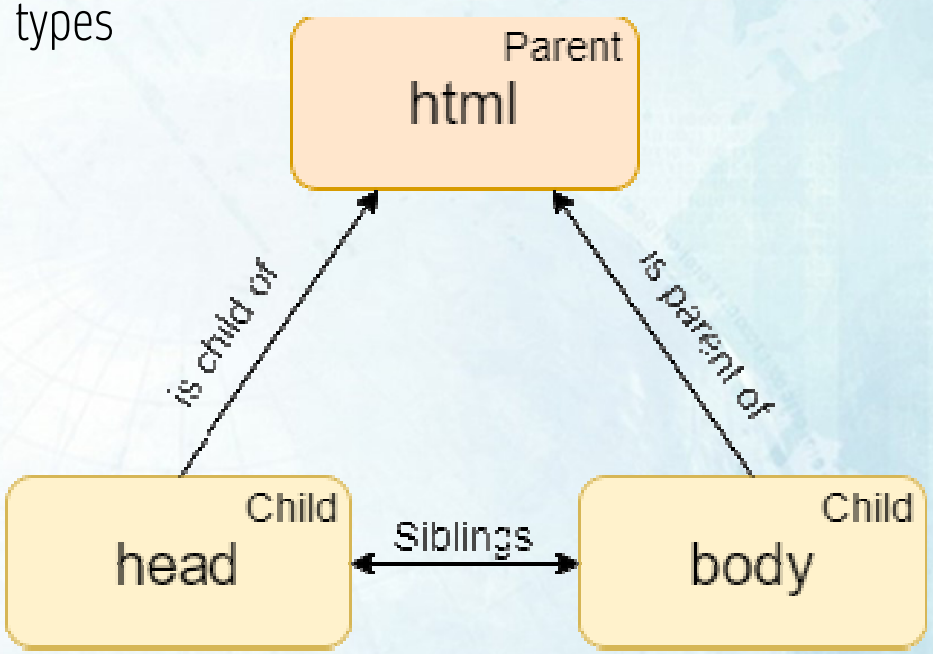
- Input event triggers every time when the below elements' value changes:
- <input>
- <select>
- <textarea>
- Unlike change, input changes every time the value changes

```
<body>
    <input type="text" id="input" name="input">
    <select id="select">
        <option value="1">1</option>
        <option value="2">2</option>
        <option value="3">3</option>
    </select>
    <textarea id="textarea"></textarea>
<script>
    let input = document.querySelector("#input");
    addEventListener("input", function() {
        //
    });
</script>
</body>
```

7.5 Recap

DOM: Nodes and Elements

- Nodes are an abstract concept that are split into 3 main types
- Text nodes: includes texts, whitespace and newline
- Element nodes: includes elements
- Comment nodes: includes comments
- The relationship between each node are the same as a traditional family tree



Selecting Elements

- getElementbyId finds element by their id
- getElementsbyName returns a collection of elements by their name
- getElementsByTagname returns a collection of elements by their tag
- getElementsByClassName returns a collection of elements by their class
- querySelector / querySelectorAll returns an element / a collection of element by their CSS selector

Traversing Elements

- parentNode returns the parent of a specified node
- firstChild / firstElementChild returns the first child / element child of the parent node
- lastChild / lastElementChild returns the last child / element child of the parent node
- childNodes / children returns a collection of all child nodes / child element nodes of the parent node
- nextElementSibling returns the next sibling in the list of elements
- previousElementSibling returns the previous sibling in the list of elements

Manipulating Elements

- createElement(Tag) returns a new element without the specified element type
- appendChild(parentElement) moves a node onto the end of the list of nodes in the parent node
- element.textContent gets / sets the text node of an element
- innerText gets / sets human-readable text only
- innerHTML gets / sets the HTML markup of a specified element

Manipulating Elements

- DocumentFragment creates a lightweight version of the document for easy modification and appending
- insertBefore inserts a new node before the specified child node
- append() inserts a set of nodes after the last child of the specified parent node
- prepend() inserts a set of nodes before the first child of the specified parent node
- insertAdjacentHTML inserts texts adjacent to the specified element

Manipulating Elements

- replaceChild replaces the old node with a new node
- cloneNode clones an element and returns it
- removeChild removes a child node from a parent node

Attributes and Properties

- Attributes and properties define a element
- element.attributes returns a collection of attributes
- Data-* attributes are reserved for developer use
- setAttribute / getAttribute sets / gets the value of an attribute
- removeAttribute removes an attribute from a specified element
- hasAttribute returns a Boolean that determines if an element has a specified attribute

Styling

- element.style changes the style property of an element
- cssText and setAttribute can be used to set multiple styles at once
- getComputedStyle retuns the computed style of an element
- className returns a space-separated string of CSS classes
- classList retuns a collection of CSS classes

Events

- An event listener "listens" to user actions and triggers
- onclick attribute, onclick property, and addEventListener applies a event to an element
- removeEventListener removes an existing event listener
- event.target returns the element that triggered the event
- DOMContentLoaded / load triggers after parts / all of the browser has been loaded
- beforeunload / unload triggers before / after everything is unloaded

Events

- Mouse events triggers when you use your mouse
- mousedown / mouseup triggers when you press / release the mouse button
- click triggers after one mouseup and one mousedown
- Keyboard events triggers when you use your keyboard
- keydown / keyup triggers when you press / release the key
- Keypress triggers constantly when you hold character keys

Events

- event.key / event.code returns the character / character key that's been pressed
- Scroll events triggers when you use the scroll wheel or scroll bar
- scrollTop / scrollLeft returns the offset of the scrolling
- scrollIntoView scrolls elements into view
- focus / blur triggers when an element receives / loses focus

Events

- Multiples of the same event can be delegated by adding it onto the parent node
- dispatchEvent triggers the event from the code
- event.isTrusted returns if the event was triggered by user actions
- MutationObservers observes changes to an element

Assignment

- Follow the assignment paper and finish the assignment in class

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/