前端網絡開發人員課程
(二) 進階網絡程式設計

3. JS DOM III: Elements III

Presented by Krystal Institute

Learning Objective

- Understand what DocumentFragment is, and how to use it
- Learn how to manipulate HTML elements

Content

3.1
Revise on the previous lesson

3.2
Document Object Model (DOM)

3.1 Revise on the previous lesson

Traversing Elements

- parentNode is used to get the read-only parent node of a specified node
- firstChild returns the first child node of the specified Element
- White space counts as text nodes
- To get the first element child, use firstElementChild instead
- lastChild returns the last child node of the specified Element
- To get the last element child, use lastElementChild instead

Traversing Elements

- childNodes return a live Nodelist of all child nodes from the specified element
- To get element nodes only, use children instead
- nextElementSibling returns the next sibling in a list of element
- previousElementSibling returns the previous sibling in a list of element

Manipulating Elements pt.1

- createElement creates and returns a separate element node
- Elelment node's properties can be manipulated
- appendChild moves a node onto the end of a list of nodes from a specified parent node
- appendChild are moved and not copied

Manipulating Elements pt.1

- textContent can be used to get the text of the specified element, along with the text of all its child nodes
- Comments and styles are ignored in textContent
- innerText returns the text of the specified element and the text of all its child nodes
- innerText only returns human-readable text
- Text with visbility:hidden and display:none will not be returned

Manipulating Elements pt.1

- TextContent/innerText can also be used to change the text of an element
- innerHTML is used to get/set the HTML markup of a specified element
- Untrusted inputs with innerHTML is dangerous
- Although HTML5 provides a safeguard that disables JS scripts from executing, there are other means of executing dangerous JS functions

innerHTML vs. createElement

- CreateElement is more efficient than innerHTML as innerHTML will need to recreate all the nodes inside the affected element, which will decrease the performance of the server
- CreateElement is more secure as it will not cause breaches, innerHTML should only be used with trusted sources like a database

3.2 Manipulating Elements pt.2

DocumentFragment

- DocumentFragment is a lightweight version of the Document that stores pieces of the document
- Document Fragment is not part of the DOM Tree
- Thus, making changes to a Document Fragment will not affect the actual DOM Tree or causes any performance issues
- To understand the use of DocumentFragment, you'll need to understand browser reflow first

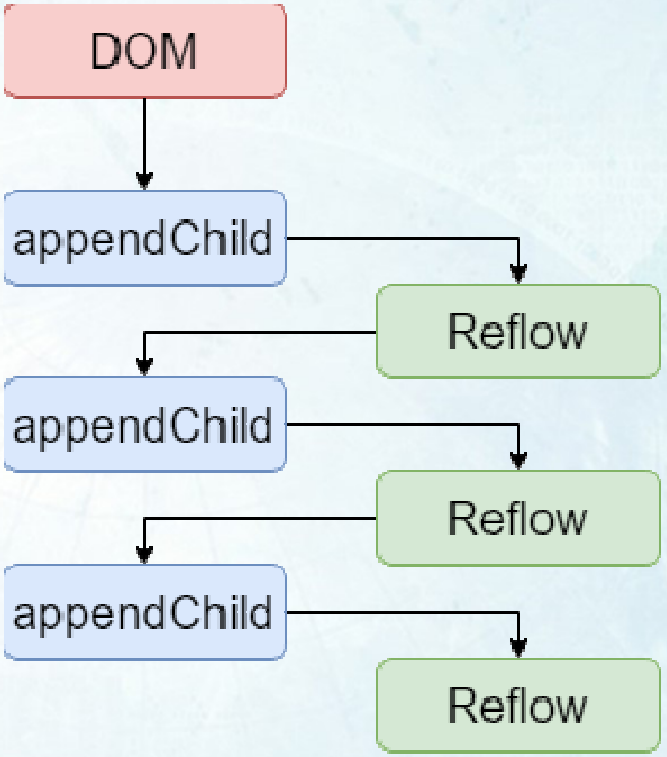
Browser Reflow

- Reflow is the term where the browser recreates and recalculates all the positions and styles of the element

- If the webpage is complex or large, it might take longer to load everything in

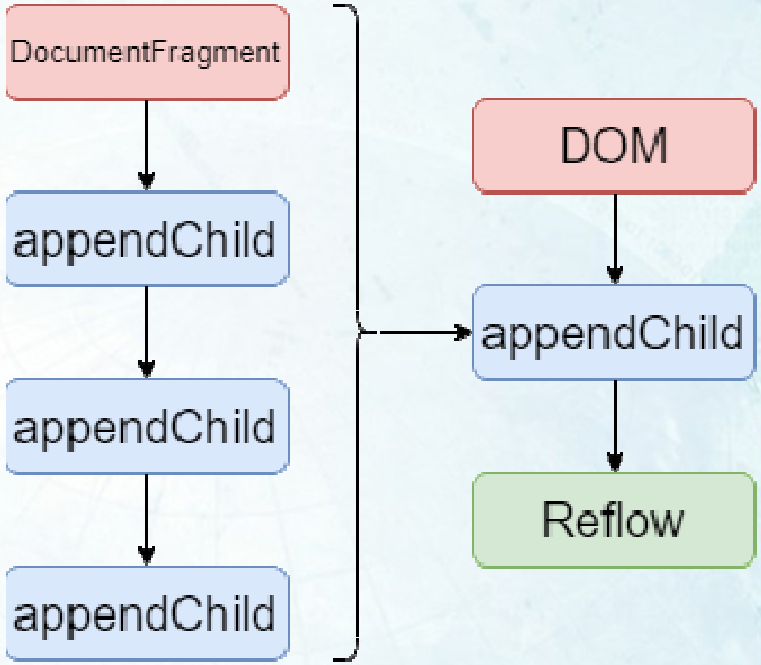
Reflow

- Manipulating elements in the DOM Tree will cause browser reflow
- This will happen with almost every element manipulation
- If 3 elements are appended, reflow will happen 3 times



Reflow

- To fix this issue, DocumentFragment creates a separate DOM
- It does not affect the performance or cause any reflow
- It can be used to be appended back to the DOM Tree, totaling with 1 reflow



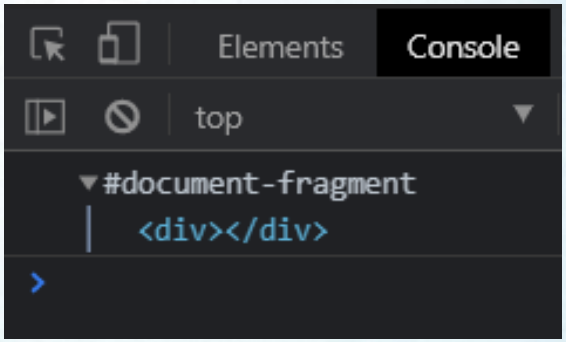
DocumentFragment

- To create a document fragment, use the DocumentFragment constructor, or use the createDocumentFragment method

```
<script>
    let fragment1 = new DocumentFragment()
    let fragment2 = document.createDocumentFragment()
</script>
```

```
let div = document.createElement("div")
fragment1.appendChild(div)
console.log(fragment1)
```

- Use appendChild to add elements into the DocumentFragment



DocumentFragment Activity

- Activity: create a documentfragment and add 100 elements into it, before adding it into the tree
- Create document fragment
- Use for loop and append 100 elements of your choice
- Append documentfragment to <body>

```
<body>
<script>
    let fragment = new DocumentFragment()
    for (let i = 0; i < 100; i++) {
        let span = document.createElement("span")
        fragment.appendChild(span)
    }
    let body = document.querySelector("body")
    body.appendChild(fragment)
</script>
</body>
```

Inserting Nodes

- insertBefore inserts a new node before the specified child node of a parent node

```
parentNode.insertBefore(newNode, existingNode);
```

```
<body>
    <ul id="list">
        <!-- insertBefore places home here -->
        <li>Services</li>
        <li>About</li>
        <li>Contacts</li>
    </ul>
<script>
    let home = document.createElement("li");
    home.textContent = "Home";
    let list = document.querySelector("#list")
    list.insertBefore(home, list.firstElementChild)
</script>
</body>
```

Inserting Nodes Exercise

- There is no insertAfter helper function, so we need to create one
- Try to use concepts from past lessons and create a insertAfter function and insert a <li> with text = Product after Home element inside
- insertAfter function should have 3 arguments:

  ParentNode, NewNode, ExistingNode

```
<ul id="list">
    <li>Home</li>
    <li>Services</li>
    <li>About</li>
    <li>Contacts</li>
</ul>
```

Inserting Nodes Solution

- In the function, use insertBefore on the next element sibling of the existing node

```
<script>
    function insertAfter(ParentNode, NewNode, ExistingNode) {
        ParentNode.insertBefore(NewNode, ExistingNode.nextElementSibling)
    };

    let product = document.createElement("li")
    product.textContent = "Product"
    let home = document.querySelector("li")
    let list = document.querySelector("#list")
    insertAfter(list, product, home)
</script>
```

Inserting Nodes

- append method inserts a set of nodes after the last child of the specified parent node

- All arguments of append() will be inserted

```
<body>
    <ul id="list">
        <li>Home</li>
    </ul>
<script>
    let namelist = ["Products", "Services", "About"];
    let list = document.querySelector("#list");
    for (let i = 0; i < namelist.length; i++) {
        let li = document.createElement("li");
        li.textContent = namelist[i];
        list.append(li);
    };
</script>
</body>
```

Inserting Nodes

- append() can also be used to insert DOMString
- Could be useful for listing out large amount of separate text

```
<body>
    <p id="para"></p>
<script>
    let p = document.querySelector("#para")
    p.append("Mary", "James", "Albert")
</script>
```

MaryJamesAlbert

append() vs appendChild

Aspects	Append()	appendChild()
Return value	Undefined	The appended Node object
Input amount	multiple	single
Parameter Types	Accepts Node and DOMString	Accepts Node only

- append() is more useful when appending multiple nodes simultaneously
- append() is not supported in Internet Explorer

Inserting Nodes

- prepend() inserts set of Nodes or DOMString before the first child of the specified parent node

```
<body>
    <p id="para"></p>
<script>
    let p = document.querySelector("#para")
    p.prepend("Mary", "James", "Albert")
</script>
</body>
```

- Works very similar to append()

Inserting Nodes

- Note that each prepend operation puts Nodes at the very top

```
let p = document.querySelector("#para")
p.prepend("Mary", "James", "Albert")
```

MaryJamesAlbert

- Multiple prepends will result in a different order than prepending them all in one function

```
let p = document.querySelector("#para")
p.prepend("Mary")
p.prepend("James")
p.prepend("Albert")
```

AlbertJamesMary

insertAdjacentHTML

- insertAdjacentHTML inserts text adjacent to the specified element
- Takes 2 argument: positionname and text

```
<body>
    <h2>JS Tutorial</h2>
    <ul id="list">
        <li>JS</li>
        <li>DOM</li>
        <li>ES6</li>
    </ul>
    <h2>JS Tutorial</h2>
<script>
    let list = document.querySelector("#list")
    list.insertAdjacentHTML("beforeend", "<li>Hello!</li>")
</script>
</body>
```

insertAdjacentHTML

- insertAdjacentHTML cannot take nodes as argument, as it only accepts text and converts it to HTML markup

- Beforebegin and afterend is only relevant if the element is in the DOM Tree (Not in DocumentFragment) and has a parent element

insertAdjacentHTML

- There are 4 positions available for inserting:
- Beforebegin inserts before the element
- Afterbegin inserts before the first child of the element
- Beforeend inserts after the last child of the element
- Afterend inserts after the element

```
    <h2>JS Tutorial</h2>
<!--  beforebegin  -->
    <ul id="list">
<!--    afterbegin    -->
        <li>JS</li>
        <li>DOM</li>
        <li>ES6</li>
<!--    beforeend    -->
    </ul>
<!--  afterend  -->
    <h2>JS Tutorial</h2>
```

insertAdjacentHTML

- The same security issue from innerHTML will also happen here
- either only use it on a trusted source or escape user input text in the function

Replace Child

- replaceChild uses a new Node to replace the old Node

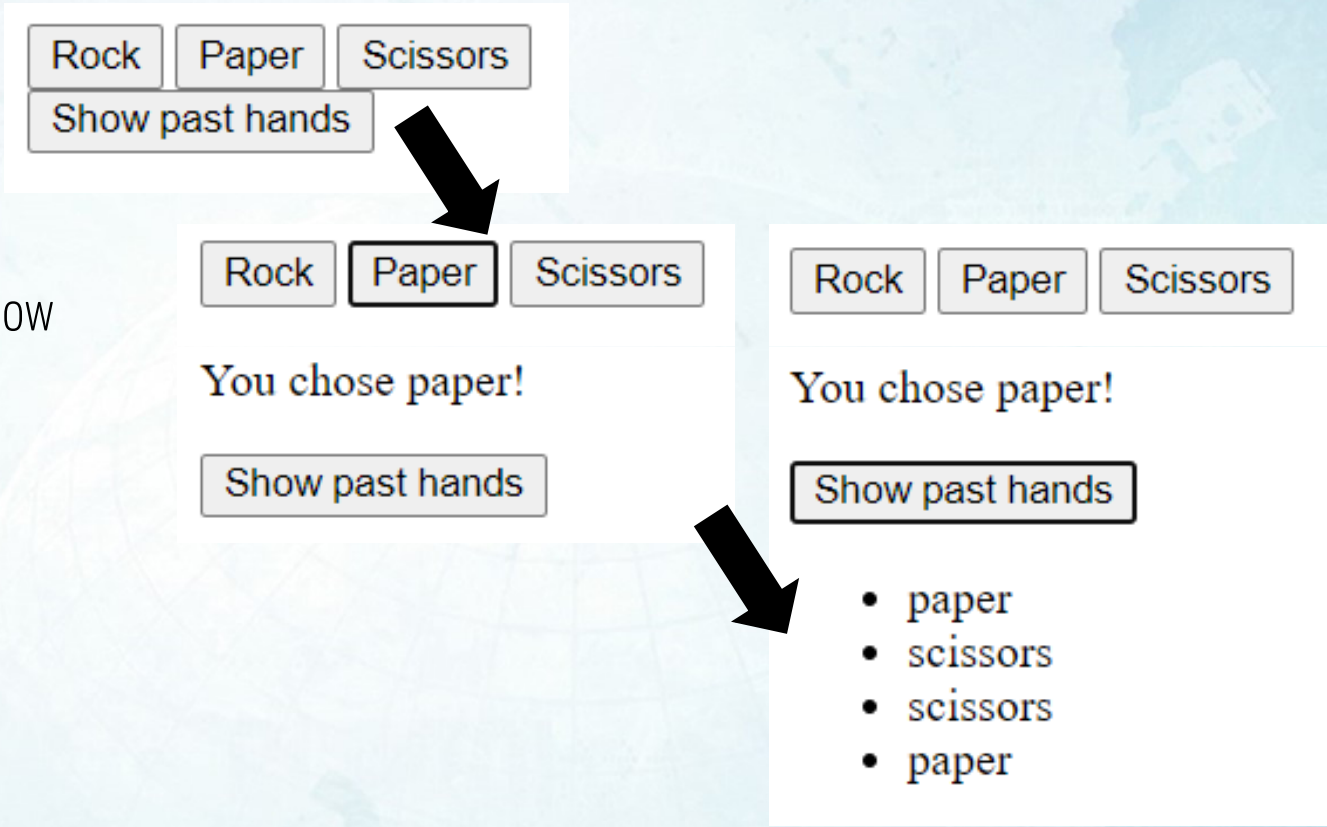
```
<body>
    <ul id="list">
        <li>JS</li>
        <li>DOM</li>
        <li>ES6</li>
    </ul>

<script>
    let list = document.querySelector("#list");
    let python = document.createElement("li");
    python.textContent = "Python";
    list.replaceChild(python, list.lastElementChild)
</script>
</body>
```

Replace Child Exercise

- Try and create a website that has 3 buttons named: Rock, Paper and Scissors and 1 button named: Show past hands
- The 3 buttons will replace a display below it with a <p>, showing what the user have clicked and chosen.
- The show past hands button will display a list of all the user's past inputs on click, and replace it on every click
- Use replaceChild instead of textContent

Replace Child Exercise Example

- Clicking on rock, paper or scissors button will display a text showing what the user chose
- clicking on show past hands will show a list of all past hands



Replace Child Solution

- Setting up buttons and divs
- Note that the <body> has id for getting its node element

```
<body id="body">
    <button type="button" onclick="Sel('rock')">Rock</button>
    <button type="button" onclick="Sel('paper')">Paper</button>
    <button type="button" onclick="Sel('scissors')">Scissors</button><br>
    <div id="display"></div>
    <button type="button" onclick="showpast()">Show past hands</button>
    <div id="pastdisplay"></div>
```

Replace Child Solution

- Setup the body element and the past record array
- In the Sel() function, <p> are created along with the text in it, it is used to replace the original
- The hand is added into the records array

```
let body = document.querySelector("#body")
let record = []
function Sel(hand) {
    let display = document.querySelector("#display")
    let p = document.createElement("p");
    p.textContent = "You chose "+hand+"!";
    p.id = "display";
    body.replaceChild(p, display);
    record.push(hand)
}
```

Replace Child Solution

- In the showpast function, a new <ul element is created
- A for loop adds hands in record into the <ul> as <li>, and it is replacing the original display

```
function showpast() {
    let pastdisplay = document.querySelector("#pastdisplay")
    let ul = document.createElement("ul");
    for (let i = 0; i < record.length; i++) {
        let li = document.createElement("li")
        li.textContent = record[i]
        ul.appendChild(li)
    };
    ul.id = "pastdisplay"
    body.replaceChild(ul, pastdisplay)
}
```

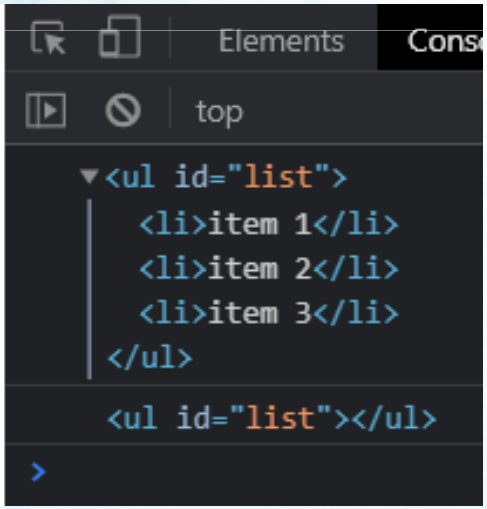
Cloning Nodes

- cloneNode is a method that allows you to clone an element
- It returns the cloned element, and is called from the target element

```
let clonedNode = originalNode.cloneNode(deep);
```

Cloning Nodes

- Deep is an Boolean argument that tells the function if the cloned node would keep all its descendants




```
<body>
    <ul id="list">
        <li>item 1</li>
        <li>item 2</li>
        <li>item 3</li>
    </ul>

<script>
    let list = document.querySelector("#list")
    let clonedlist1 = list.cloneNode(true)
    let clonedlist2 = list.cloneNode(false)
    console.log(clonedlist1)
    console.log(clonedlist2)
</script>
</body>
```

Cloning Nodes

- cloneNode copies ALL attributes and inline listeners of the original node
- It doesn't clone assigned properties in <script> and event listeners added via addEventListener (will be covered in future lessons)
- The id attribute will also be copied so it is advised to change the id of the cloned node

Remove Child

- removeChild function removes a child node from a parent node

- If there is no child or no matching child in the parent node, the method will throw an exception

```
<body>
    <ul id="list">
        <li>item 1</li>
        <li>item 2</li>
        <li>item 3</li>
    </ul>

<script>
    let list = document.querySelector("#list")
    let li = list.removeChild(list.firstElementChild)
</script>
</body>
```

Remove Child

- Assigning the function to a variable will save the removed Child for later use

```
<body>
    <ul id="list">
        <li>item 1</li>
        <li>item 2</li>
        <li>item 3</li>
    </ul>

<script>
    let list = document.querySelector("#list")
    let li = list.removeChild(list.firstElementChild)
    // after some time
    list.appendChild(li)
</script>
</body>
```

Remove Child Exercise

- Create a website with a <ul> with at least 3 <li> items, a remove button and a add button

- Simulating a Queue, have the remove button remove the top child of the <ul>, and save it
- The add button will add the removed <li> back to the bottom of the

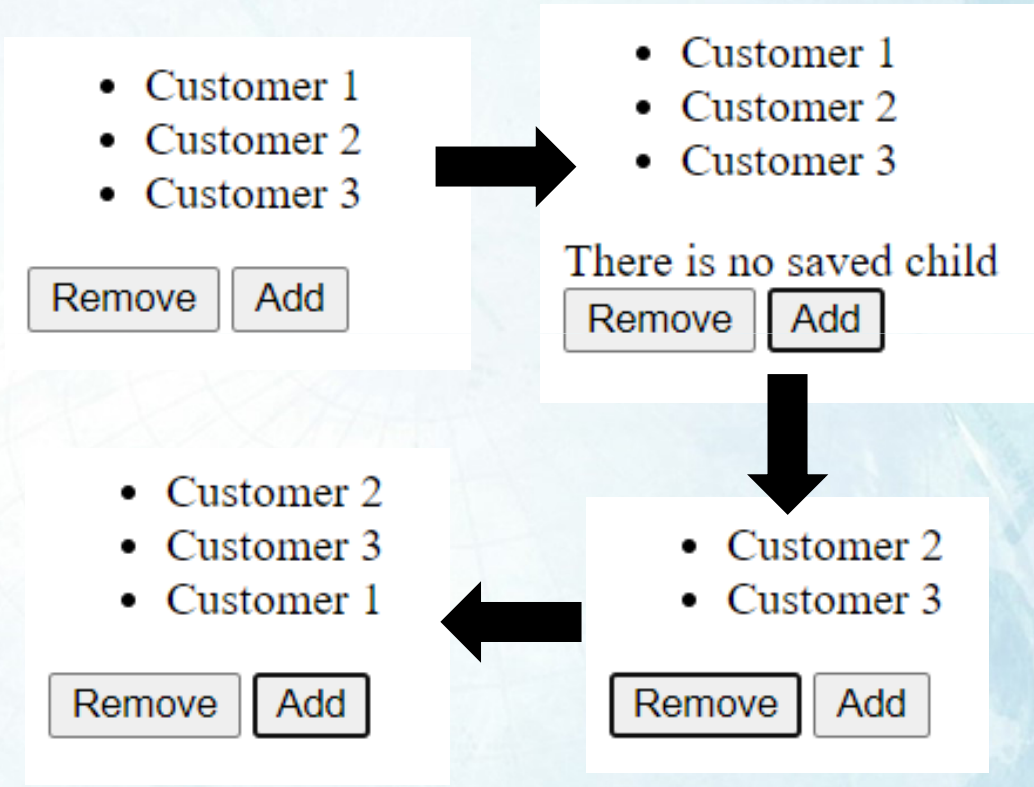
Remove Child Exercise

- Create a website with a <ul> with at least 3 <li> items, a remove button and a add button

- If there is no removed child, displays in a that there is no saved child when add button is clicked
- Button should work with multiple children, saving by the order they are removed, adding them back by order
- Finish the exercise by the end of the lesson

Remove Child Exercise Example

- Clicking the add button display that there is no saved child
- clicking remove removes the top item in the list
- Clicking add adds back the removed item



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/