DOM and Events

Document Object Model (DOM)
Events Handling in JavaScript

Table of Contents

DOCUMENT OBJECT MODEL (DOM)

Document Object Model

- What is Document Object Model (DOM)?
  - A concept of representing a HTML document as a "DOM tree"
  - Consists of elements that have child elements
  - Elements have properties (attribute + value) and events
- DOM provides an API for traversing / modifying the DOM tree
  - Enables developers to modify the HTML content and the visual presentation of the currently loaded HTML document
  - E.g. load a table data (JSON) and show it as a HTML table

THE DOM API

The DOM API

- Web browsers provide a DOM API
  - Consists of objects and methods to interact with the HTML page
  - Can add / modify / remove HTML elements
  - Can add / modify / remove HTML attributes
  - Can apply CSS styles dynamically
- HTML elements and their properties are mapped to JS objects
  - **`document.documentElement`** is the **`<html>`** element
  - **`document.body`** is the **`<body>`** element of the page

HTML Elements – Common Properties

- All HTML elements have common properties
  - Corresponding to the their HTML attributes
  - `id`, `className`, `style`, `onclick`, etc.
  - `innerHTML`
    - Holds a string – the content of the element, without the element
  - `outerHTML`
    - Holds a string – the content of the element, with the element
  - `innerText` / `textContent`
    - Holds a string – the text content of the element, without the tags

DOM Objects and Types

- Each HTML element has a corresponding DOM object type
  - **`HTMLLIElement`** represents **`<li>`**
  - **`HTMLAudioElement`** represents **`<audio>`**
- Each of these objects have its specific properties
  - **`HTMLAnchorElement`** has **`href`** property
  - **`HTMLImageElement`** has **`src`** property
  - **`HTMLInputElement`** has **`value`** property
- The **`document`** object is a special object
  - It represents the entry point for the DOM API (the DOM tree root)

SELECTING DOM ELEMENTS

Selecting HTML Elements from DOM

- Select a single element → returns `HTMLElement`

```
var header = document.getElementById('header');
var nav = document.querySelector('#main-nav');
```

- Select a collection of elements → returns a collection

```
var inputs = document.getElementsByTagName('li');
var radiosGroup = document.getElementsByName('genders[]');
var header = document.querySelectorAll('#main-nav li');
```

– Access the predefined collections of elements

```
var links = document.links;
var forms = document.forms;
```

Selecing with document.getElementsBy…

- Select element by ID → returns `HTMLElement`

```
var header = document.getElementById('header');
```

- Select elements by CSS class → returns a collection

```
var posts = document.getElementsByClassName('post-item');
```

- Select elements tag name → returns a collection

```
var sidebars = document.getElementsByTagName('sidebar');
```

- Select element by name (in forms) → returns a collection

```
var gendersGroup = document.getElementsByName('genders[]');
```

Query Selectors

- CSS-like selectors for accessing the DOM tree
  - `querySelector(…)`
    - Returns the first element that matches the selector
  - `querySelectorAll(…)`
    - Returns a collection of all elements that match the selector

```
var header = document.querySelector('#header');
```

```
var tableCells = document.querySelectorAll('table tr td');
```

```
var selectedLi = document.querySelector('menu > li.selected');
```

```
var specialLinks = document.querySelectorAll('a.special');
```

Selecting Inner Elements

- HTML elements support select for their inner elements
  - Select all DIVs that are inside an element with id "**`wrapper`**"

```
var wrapper = document.getElementById('wrapper');
var divsInWrapper = wrapper.getElementsByTagName('div');
```

- All methods can be used on HTML elements
  - Except **`getElementById()`**

TRAVERSING THE DOM

Create, Remove, Alter and Append HTML Elements

Traversing the DOM

- DOM elements know their position in the DOM tree
  - Parent: **`element.parentNode`**
    - Returns the direct parent of the element (null for the document)
  - Children: **`element.childNodes`**
    - Returns a **`NodeList`** of all the child nodes (including the text nodes)
    - First / last child – **`element.firstChild`** / **`element.lastChild`**
  - Siblings (elements around the element):
    - **`element.nextSibling`** / **`element.nextElementSibling`**
    - **`element.previousSibling`** / **`element.previousElementSibling`**

Traversing the DOM – Example

```
var trainersList =
  document.getElementsByClassName("trainers-list")[0];

var parent = trainersList.parentNode;
log("parent of trainers-list: " + parent.nodeName +
  " with id: " + parent.id);

var children = trainersList.childNodes;
log("elements in trainers-list: " + children.length);

log("element in trainers-list");
for (var i = 0, len = children.length; i < len; i++) {
  var subItem = children[i]
  log(subItem.nodeName + " content: " + subItem.innerText);
}
```

Manipulating the DOM

- DOM can be manipulated dynamically with JS
  - – HTML elements can be created
  - – HTML elements can be removed
  - – HTML elements can be altered
    - Change their content
    - Change their styles
    - Change their attributes

Creating HTML Elements

- The document object can create new HTML elements
  - `document.createElement(elementName)`
- Newly created elements are not in the DOM (the web page)
  - Must be appended to DOM manually

```
var studentsList = document.createElement("ul");
studentsList.innerHTML = "Student: Alex";
var liElement = document.createElement("li"); studentsList.appendChild(studentLi);
document.body.appendChild(studentsList);
```

Inserting Elements Before / After Element

- The DOM API supports inserting a element before or after a specific element
  - `element.appendChild(child)`
    - Inserts the element always at the end of the DOM element
  - `parent.insertBefore(newNode, specificElement)`
    - Inserts the element before specific element
  - `parent.insertAfter(newNode, specificElement)`
    - Inserts the element after specific element

Removing Elements

- Elements can be removed from the DOM
  - Using **`element.removeChild(elToRemove)`**
  - Pass the element-to-remove to their parent

```
var trainers = document.getElementsByTagName("ul")[0];
var trainer = trainers.firstChild;
trainers.removeChild(trainer);

// Remove a selected element
var selectedElement = //select the element
selectedElement.parentNode.removeChild(selectedElement);
```

Altering the Elements

- DOM elements can be changed and removed
- With the DOM API each DOM element node can be altered
  - – Change its properties or appearance

```
<div id="f"><p id="the-p">text</p></div>
<div id="s"></div>
…
var second = document.getElementById("s");
var theP = document.getElementById("the-p");
second.appendChild(theP);
…
// The DOM is:
<div id="f"></div>
<div id="s"><p id="the-p">text</p></div>
```

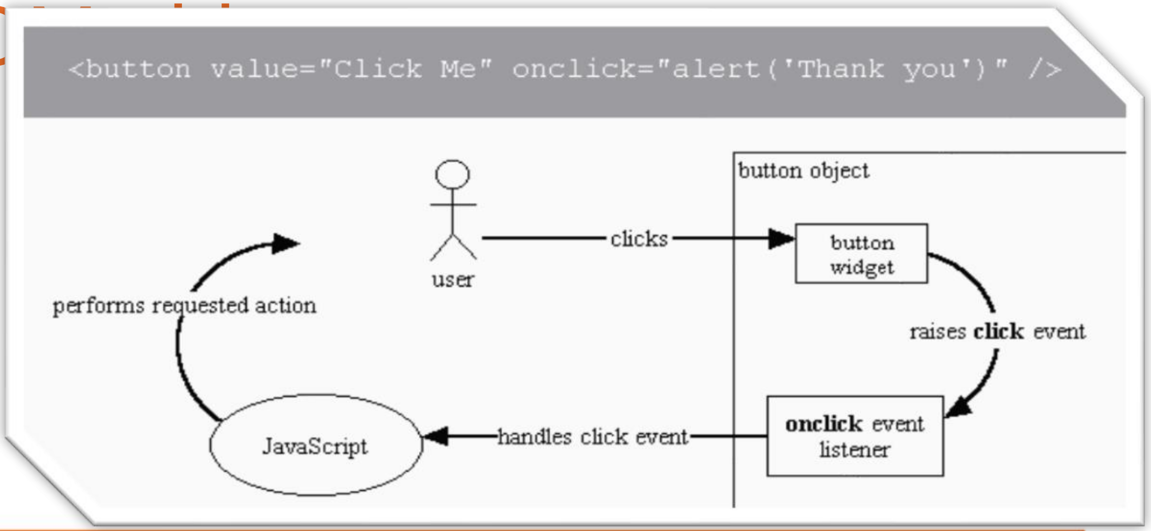
Appending DOM Elements

- The DOM API supports appending DOM elements to a element
- **`parentNode.appendChild(node)`**
  - Appends the DOM element node to the DOM element **`parentNode`**
  - If **`parentNode`** is appended to the DOM, the **`childNode`** is also appended

JAVASCRIPT EVENT MODEL

Event Model

- The DOM event model provides notifications for certain events
  - E.g. execute a JS function when a button is clicked
- The DOM event model consists of events and event listeners attached to the DC
- Events Demo



```
<button value="Click Me" onclick="alert('Thank you')" />
```

Event Types

- DOM provides access to many events
  - – Mouse events – mouse clicks, mouse moves, mouse over, …
  - – Touch events – finger touch, touch start, end, move, …
  - – Form events – field focus, value change, form submit, …
  - – Keyboard events – key down, key up, key press, …
  - – DOM / UI events – node insert, node remove, load, resize, …
- Full list of all DOM event types:
  - – http://www.w3.org/TR/DOM-Level-3-Events/#event-types-list
- You may also define custom event types

Common Event Types

- Mouse

```
click
hover
mouseup
mousedown
mouseover
mouseout
```

- DOM / UI

```
load
abort
select
resize
change
```

- Touch

```
touchstart
touchend
touchcancel
touchleave
touchmove
```

- Keyboard

```
keydown
keypress
keyup
```

- Focus

```
focus
blur
focusin
focusout
```

EVENT HANDLER REGISTRATION

Define Event Handler in the HTML Code

- Event handling JavaScript code can be specified in the HTML attributes **onclick**, **onload**, **onmouseover**, **onresize**, …

```
<button onclick="buttonClickFunction()">Click Me!</button>
```

⇩

```
function buttonClickFunction() {
  console.log("You clicked the [Click Me!] button");
}
```

```
<button onclick="alert('OK clicked')">OK</button>
```

Define Event Handler in the JS Code

- Event handling JavaScript code can be specified in the JS code through the properties **onclick**, **onresize**, …

```
<button id="click-button">Click Me!</button>
```

⇩

```
<button id="click-button">Click me</button>

var button = document.getElementById("click-button");

button.onclick = function onButtonClick() {
  console.log("You clicked the button");
}
```

Using addEventListener(…)

- A more powerful way for attaching event handlers:

```
domElement.addEventListener(
  eventType, eventHandler, isCaptureEvent)
```

  - `isCaptureEvent`: catch the "capture" or "bubbling" phase
  - Can attach multiple events in a chain

```
var button = document.getElementById("buttonOK");

button.addEventListener("click", function() {
  console.log("You clicked me");
}, false);
```

Summary

1. Document Object Model (DOM)
   – The DOM API
   – Selecting DOM elements
   – Creating / modifying / deleting elements
2. JavaScript Event Model
   – Event handler registration