Client-side Scripting

Databases and Interfaces

Matthew Pike & Yuan Yao

University of Nottingham Ningbo China (UNNC)

Overview

This Lecture

- Identify when Client-side scripting is necessary
- Introduce JavaScript
- Write (simple) Client-side scripts using JavaScript
- Understand Event-driven Programming

Note: This Lecture

! DBI Assessment

In the DBI exam, you will not be expected to write JavaScript code. However, you will be expected to understand the concepts of Client-side Scripting and Event-driven Programming. If you are interested in learning more about JavaScript, you can find a number of resources online. We recommend the Mozilla Developer Network as a good starting point.

Recap: What makes a web page?

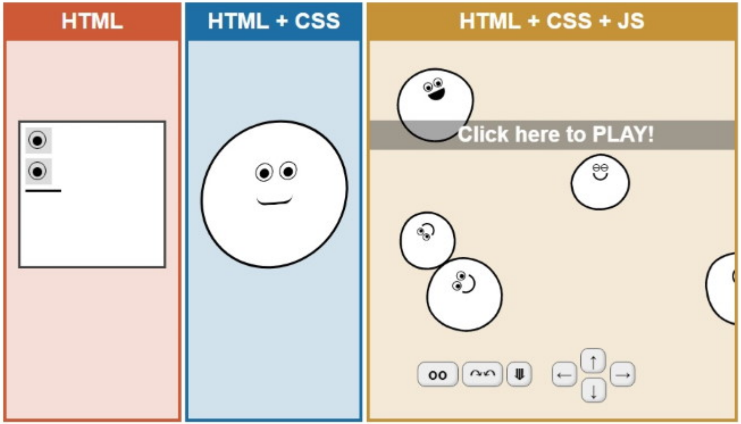

Figure 1: The role of HTML, CSS and JavaScript in a webpage.

What is Client-side Scripting?

- Client-side scripting is the use of a programming language to add interactivity to a web page
- Client-side scripting is executed (run) in by the user’s web browser
- JavaScript is the most common client-side scripting language used on the web

Client-side Scripting vs Server-side Scripting

- A client-side script runs in the user’s browser **when particular events occur**
    - An event may be:
        - A user clicking a button or moving the mouse
        - A user typing on the keyboard
        - The page completing loading
    - Client-side scripting is often used to validate user input before it is sent to the server
        - This improves the user experience as the user does not have to wait for the server to respond
        - It also reduces the load on the server
- Server-side scripts run on the web server **when the user requests information**
    - Server-side scripts have access to the server’s file system and databases
    - Server-side scripting will generate dynamic content, as we’ve seen in previous lectures with Flask
    - Server-side scripting is often used to validate user input before it is stored in the database

When to use Client-side Scripting?

- Yes
    - Providing quick feedback to users on their (form) input
    - Dynamic content loading/update
    - Introduce interactive components to pages
- No
    - Tasks which require privileged access to remote databases
    - If access to user’s hard drive is required (also, server-side scripting is not appropriate here)
    - Operations which require some guarantee of running (users may disable client-side scripting)

JavaScript

What is JavaScript?

- JavaScript is a lightweight programming language
- It is most well-known as the scripting language for web pages, but it is also used in many non-browser environments, such as node.js
- JavaScript != Java
- JavaScript was first released by Netscape in 1995
- JavaScript is often abbreviated to JS
- It is a web standard, and is supported by all major web browsers

What can JavaScript do?

- JavaScript can be used to add interactivity to a web page
    - React to user, page and state events
- Update the content of a web page (without reloading the page), using the Document Object Model (DOM)
    - Change the content of HTML elements
    - Change the style of HTML elements
    - Change the attributes of HTML elements
- Send and receive data from a web server (without reloading the page)
    - Send data to a web server
    - Receive data from a web server
- Create Cookies: small files stored on the user’s computer by the web browser to store information about the user
- Detect the user’s browser and operating system
- Validate user input before it is sent to the server

JavaScript: Hello World

- JavaScript is embedded in HTML pages using the `<script>` tag
- JavaScript code can be included in two ways:
    - In the HTML page using the `<script>` tag with the code inside
    - In an external file using the src attribute of the `<script>` tag to point to the file
        - e.g. `<script src="myScript.js">`

```html
<!DOCTYPE html>
<html>
    <head>
        <title>JavaScript: Hello World</title>
    </head>
    <body>
        <h1>JavaScript: Hello World</h1>
        <script>
            alert("Hello World!");
        </script>
    </body>
</html>
```

Event-driven Programming

What is Event-driven Programming?

- Event-driven programming is a paradigm in which the flow of the program is determined by events such as user actions (mouse clicks, key presses), sensor outputs, or messages from other programs or threads
- Event-driven programming is different from the traditional procedural programming paradigm in which the flow of the program is determined by the sequence of statements in the code
- In JavaScript, the most common way to handle events is to assign a function to an event handler. Example:

```
button = document.getElementById("myButton");
button.onclick = function() {
    alert("Hello World!");
}
```

Event-driven Programming: Example

```html
<!DOCTYPE html>
<html>
    <head> <title>Event-driven Programming: Example</title> </head>
    <body>
        <h1>Event-driven Programming: Example</h1>
        <button id="myButton">Click Me!</button>
        <script>
            button = document.getElementById("myButton");
            button.onclick = function() {
                alert("Hello World!");
            }
        </script>
    </body>
</html>
```

Interacting with the DOM

Recap: The Document Object Model (DOM)

- The Document Object Model (DOM) is a programming interface for HTML and XML documents
- The DOM represents the document as nodes and objects
- That way, programming languages can connect to the page and change the document structure, style and content dynamically

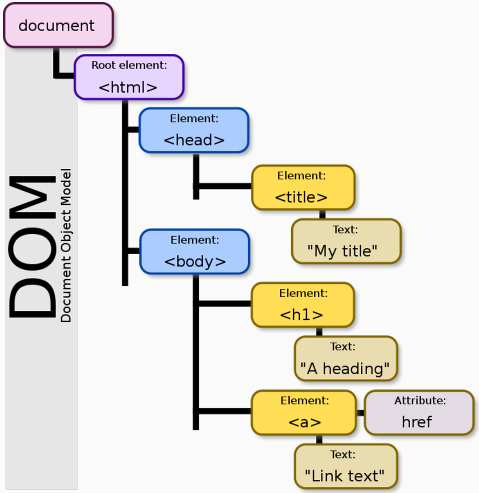

Figure 2: A visual representation of the DOM

DOM Interaction with JavaScript

- We can use JavaScript to interact with the DOM, this may include:
  - Changing the contents, style or attributes of existing HTML elements
  - Adding new HTML elements to the DOM
  - Introducing event handlers to existing HTML elements
  - Finding HTML elements by their id or other attributes
- We access different parts of the DOM using the `document` object
  - `document.getElementById(id)`: returns the element with the given id
  - `document.getElementsByClassName(className)`: returns a list of elements with the given class name
  - `document.getElementsByTagName(tagName)`: returns a list of elements with the given tag name

DOM Interaction with JavaScript: Example

```
<!-- For brevity, we've omitted the <html>, <head> and <body> tags -->
    <p id="myParagraph">This is a paragraph</p>
    <script>
        function changeParagraph() {
            // Get the paragraph element using its id
            paragraph = document.getElementById("myParagraph");
            // Change the html of the paragraph element
            paragraph.innerHTML = "This is a new paragraph";
        }
        // Wait 5 seconds after the page loads,
        // then change the paragraph's text
        setTimeout(changeParagraph, 5000);
    </script>
```

Form Validation with JavaScript

- We can use JavaScript to validate user input before it is sent to the server
    - This improves the user experience as the user does not have to wait for the server to respond
    - It also reduces the load on the server
- We must be careful to validate user input on the server as well, as the user can disable JavaScript in their browser
    - Client-side validation is not a substitute for server-side validation
- We can use JavaScript to validate user input in a form before it is sent to the server
    - We can use the `onsubmit` event handler to validate the form before it is submitted
    - Or we can use the `oninput` event handler to validate the form as the user is typing

Form Validation with JavaScript: Example (HTML Form)

```html
<!-- For brevity, we've omitted the <html>, <head> and <body> tags -->
    <h1>Form Validation: Example</h1>

    <form id="myForm" onsubmit="return validateForm()">
        <label for="name"> Name:</label>
        <input  type="text" id="name" name="name">

        <label for="email">Email:</label>
        <input  type="email" id="email" name="email">

        <input  type="submit" value="Submit">
    </form>
```

Form Validation with JavaScript: Example (JavaScript Validation)

```
function validateForm() {
    var form = document.getElementById("myForm");
    var name = form.elements["name"];
    var email = form.elements["email"];
    if (name.value.length < 3) {
        alert("Please enter your name");
        return false; // Prevent the form from being submitted
    }
    if (email.value.length < 3 || !email.value.includes("@")) {
        alert("Please enter your email");
        return false; // Prevent the form from being submitted
    }
    return true; // Allow the form to be submitted
}
```

References

Online Resources

- Mozilla Developer Network
    - https://developer.mozilla.org/en-US/docs/Web/JavaScript
- Learn JavaScript Online
    - https://learnjavascript.online
- JavaScript technologies overview
    - https://developer.mozilla.org/en-US/docs/Web/JavaScript/JavaScript_technologies_overview