Document Object Model
&
Form

By LAKSHMIKANT DESHPANDE

DOM

- DOM stands for Document Object Model.
- A programming interface for web documents
- The DOM allows programs to interact with the structure, content, and style of web pages dynamically.
- DOM manipulation refers to accessing, modifying, or manipulating elements and properties of the DOM tree.
- It is performed using programming languages like JavaScript.
- DOM manipulation enables dynamic changes to the content, style, and behavior of web pages.
- DOM manipulation methods are provided by JavaScript to interact with the DOM.
- These methods allow for operations such as selecting elements, creating new elements, modifying attributes or content, and adding or removing elements from the DOM.

DOM Manipulation Methods

- getElementById(): Retrieves an element from the DOM based on its unique ID.
- getElementsByClassName(): Retrieves a collection of elements based on the class name.
- getElementsByTagName(): Retrieves a collection of elements based on the tag name.
- querySelector(): Retrieves the first element that matches a specified CSS selector.
- querySelectorAll(): Retrieves a list of elements that match a specified CSS selector.
- createElement(): Creates a new DOM element.
- appendChild(): Appends a child element to a parent element.
- removeChild(): Removes a child element from its parent element.
- innerHTML: Gets or sets the HTML content of an element.
- style: Allows you to access or modify the CSS styles of an element.
- addEventListener(): Attaches an event handler function to an element.
- removeEventListener(): Removes an event handler function from an element.

Events in DOM

- Attaching Event
  - addEventListener()
- Detaching Event
  - removeEventListener()
- Manipulating Elements
  - createElement()
  - appendChild()
  - removeChild()
  - replaceChild()
- Manipulating Attributes
  - setAttribute()
  - removeAttribute()

- Mouse events:
  - click: Triggered when the mouse button is clicked.
  - dblclick: Triggered when the mouse button is double-clicked.
  - mouseover: Triggered when the mouse pointer enters an element.
  - mouseout: Triggered when the mouse pointer leaves an element.
  - mousedown: Triggered when a mouse button is pressed down.
  - mouseup: Triggered when a mouse button is released.
  - mousemove: Triggered when the mouse pointer moves.
  - contextmenu: Triggered when the right mouse button is clicked, opening the context menu.
  - wheel: Triggered when the mouse wheel is scrolled.
- Keyboard events:
  - keydown: Triggered when a key is pressed down.
  - keyup: Triggered when a key is released.
  - keypress: Triggered when a key is pressed down and released.

- Form events:
  - submit: Triggered when a form is submitted.
  - input: Triggered when the value of an input field changes.
  - change: Triggered when the value of a form element changes and loses focus.
  - focus: Triggered when an element receives focus.
  - blur: Triggered when an element loses focus.
  - select: Triggered when the text within an input or textarea is selected.
- Touch events:
  - touchstart: Triggered when a touch point is placed on the touch surface.
  - touchend: Triggered when a touch point is removed from the touch surface.
  - touchmove: Triggered when a touch point moves along the touch surface.
  - touchcancel: Triggered when a touch event is canceled.

- Focus events:
  - focus: Triggered when an element receives focus.
  - blur: Triggered when an element loses focus.
- Window(Page) events:
  - load: Triggered when the web page finishes loading.
  - resize: Triggered when the browser window is resized.
  - scroll: Triggered when scrolling occurs in an element.
  - unload: Triggered when the web page is unloaded or closed.
  - orientationchange: Triggered when the device orientation changes (for mobile devices).
  - DOMContentLoaded: Triggered when the initial HTML document has been completely loaded and parsed.
  - readystatechange: Triggered when the ready state of the document changes (e.g., loading, interactive, complete).

Manipulating CSS using Javascript

- document.style.property = 'value';
- classlist
- addClass
- removeClass
- InnerHTML
- Document.write() vs innerHTML

Event bubbling and Event Capturing

- Event bubbling and event capturing are two mechanisms that describe the order in which event handlers are executed when an event occurs on a nested DOM structure.

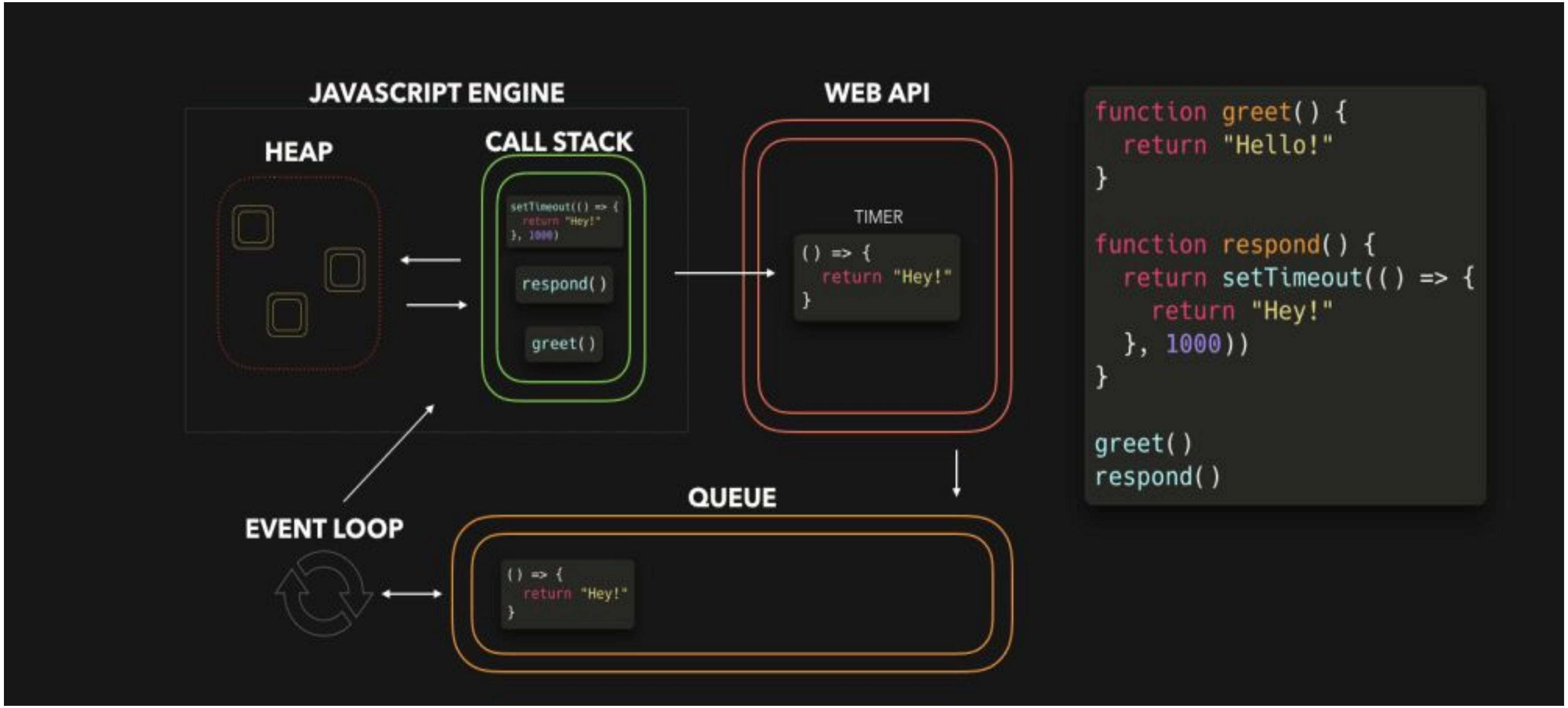
- ***Event Bubbling:***
  - In event bubbling, when an event is triggered on a nested element, the event is first handled by the innermost element and then propagates (bubbles up) to its parent elements in the DOM hierarchy.
  - Event handlers attached to the parent elements are also triggered in sequence until the event reaches the outermost element or the document level.
  - This is the default behavior in most browsers.
  - Event bubbling allows for a convenient way to handle events on parent elements that contain multiple child elements without having to attach individual event handlers to each child element.

- ***Event Capturing:***
  - In event capturing, when an event is triggered on a nested element, the event is first captured (flows down) from the outermost element to the innermost element.
  - Event handlers attached to the parent elements are triggered in sequence until the event reaches the target element (the element on which the event originally occurred).
  - After the event reaches the target element, the event then bubbles up to the parent elements following the regular event bubbling behavior.
  - Event capturing is less commonly used than event bubbling, but it can be useful in certain scenarios where you want to handle events at the early stages of the event flow.

Event Lifecycle

- ***Capturing Phase:***
  - The capturing phase is the first phase of the event lifecycle.
  - During this phase, the event starts at the topmost ancestor element and travels down the DOM hierarchy towards the target element.
  - Event handlers attached with event capturing (useCapture set to true) are triggered during this phase.
  - The capturing phase allows ancestor elements to intercept and handle the event before it reaches the target.
- ***Target Phase:***
  - The target phase is the second phase of the event lifecycle.
  - During this phase, the event reaches the target element, which is the element on which the event originally occurred.
  - Event handlers attached directly to the target element are triggered.
  - This phase provides an opportunity to handle the event specifically for the target element.
- ***Bubbling Phase:***
  - The bubbling phase is the final phase of the event lifecycle.
  - During this phase, the event starts from the target element and propagates up the DOM hierarchy towards the topmost ancestor.
  - Event handlers attached with event bubbling (useCapture set to false) are triggered during this phase.
  - The bubbling phase allows ancestor elements to handle the event after it has been handled by the target element.

Event Loop



```
function greet() {
  return "Hello!"
}

function respond() {
  return setTimeout(() => {
    return "Hey!"
  }, 1000))
}

greet()
respond()
```

Event loop In Theory

- The event loop is a fundamental concept in JavaScript that manages the execution of asynchronous operations and event-driven programming.
- It is responsible for handling and dispatching events, executing callbacks, and ensuring the non-blocking nature of JavaScript.

The event loop works based on an event-driven architecture and consists of the following components:

- ***Call Stack:*** The call stack is a data structure that keeps track of function calls in the JavaScript runtime. Whenever a function is called, it is added to the top of the call stack, and when a function finishes executing, it is removed from the stack.
- ***Event Queue:*** The event queue is a queue-like data structure that holds events and their associated callback functions. When an event occurs or an asynchronous operation completes, the corresponding callback is added to the event queue.
- ***Event Loop:*** The event loop is responsible for continuously monitoring the call stack and the event queue. It checks if the call stack is empty, and if so, it takes the first callback from the event queue and pushes it onto the call stack for execution.

The event loop follows a specific cycle

- The event loop checks if the call stack is empty.
- If the call stack is empty, it takes the first callback from the event queue and pushes it onto the call stack.
- The callback on top of the call stack is executed.
- As the callback executes, it may perform synchronous operations or initiate asynchronous operations using timers, AJAX requests, or other APIs.
- If an asynchronous operation completes or an event occurs, the corresponding callback is added to the event queue.
- The event loop repeats the cycle, checking the call stack and processing events from the event queue.

- The event loop ensures that JavaScript remains responsive and non-blocking by handling asynchronous operations and events efficiently.
- It allows for the execution of time-consuming tasks without blocking the main execution thread and enables the handling of user interactions, network requests, timers, and other events in an organized manner.
- Understanding the event loop is important for writing efficient, responsive, and asynchronous JavaScript code.
- By leveraging the event loop, developers can create smooth user experiences and handle complex operations without blocking the execution of other tasks.

Forms

- Forms are an essential part of web development that allow users to input and submit data to a web page or application.
- A form typically consists of various form elements such as text fields, checkboxes, radio buttons, dropdown menus, and buttons.
- Users can fill in the form fields with data and submit it to be processed by the server or utilized in client-side JavaScript.
- Form validations refer to the process of validating the user-submitted data to ensure it meets certain requirements or constraints.
- Form validations can be implemented using JavaScript, HTML attributes, or server-side scripting languages.
- Validations can be performed using JavaScript functions, regular expressions, or libraries/frameworks that offer built-in validation mechanisms.

Common types of form validations

- Form validation refers to the process of validating the entire form as a whole, considering the interdependencies and interactions between multiple form fields. Form validation is typically performed before the form is submitted to the server. It ensures that the collected data is complete, valid, and consistent.
- Required field validation: Ensures that specific fields are not left empty.
- Data format validation: Verifies that data entered matches a specific format (e.g., email address, phone number, date).
- Length validation: Checks the length of input against predefined minimum and maximum lengths.
- Numeric value validation: Validates that numeric input falls within a certain range or is a valid number.
- Password validation: Applies rules to ensure secure and strong passwords (e.g., minimum length, inclusion of special characters).
- Confirmation/validation of specific fields: Compares the values of two or more fields to ensure they match (e.g., password and confirm password fields).

Regular Expressions

Form Validation Using Regular Expressions (Regex):

- Regex can be used to define and validate specific patterns of user input.
- Examples of regex-based form validations include:
  - Validating email addresses: /^[A-Za-z0-9._%+-]+@[A-Za-z0-9.-]+\.[A-Za-z]{2,}$/
  - Validating phone numbers: /^\d{3}-\d{3}-\d{4}$/
  - Validating ZIP codes: /^\d{5}(-\d{4})?$/

Form Validation Without Regular Expressions:

- Form validation can also be performed without using regex, relying on JavaScript methods and techniques.
- Examples of form validation without regex include:
  - Required field validation: Checking if a required field is empty or not.
  - Numeric value validation: Using isNaN() or typeof to check if a value is a valid number.
  - Length validation: Comparing the length of an input value using length property.
  - Custom validation functions: Creating custom validation functions to validate specific patterns or conditions.