Programação Avançada para a Internet

Mestrado em Engenharia de Software

Modern JavaScript & HTML5 APIs

Today's Agenda

1. **The Modern JS Foundation:** A quick review of the essential ES6+ syntax.
2. **The Asynchronous Challenge:** Understanding the Event Loop.
3. **Solving Asynchronicity:** From Promises to `async/await`.
4. **Powerful HTML5 APIs:** Unlocking native browser capabilities.
5. **The Ultimate Goal:** Building a Progressive Web App (PWA).

Part 1: The Modern JavaScript Foundation

(ES6+ Syntax)

`let` / `const` & Block Scope

- `let` : A variable that can be reassigned.
- `const` : A constant reference; cannot be reassigned.
- **Both are block-scoped (`{...}`)**, which prevents bugs. Avoid `var` .

```
// let is block-scoped
if (true) {
  let y = 10;
}
console.log(y); // ReferenceError: y is not defined

// const cannot be reassigned
const z = 20;
z = 25; // TypeError
```

Rule: Default to `const` , use `let` only when you need to.

Arrow Functions `=>`

A more concise syntax for writing functions, now the standard in modern JS.

```
// Classic function
const add = function(a, b) {
  return a + b;
};

// Arrow function with implicit return
const multiply = (a, b) => a * b;

// Arrow function with one parameter
const square = num => num * num;
```

Destructuring Assignment

A convenient way to extract values from arrays or properties from objects into distinct variables.

```
// Object Destructuring
const user = {
  firstName: 'John',
  lastName: 'Doe'
};
const { firstName, lastName } = user;
console.log(firstName); // 'John'

// Array Destructuring
const [first, second] = [10, 20];
console.log(first); // 10
```

Spread (`...`) and Rest (`...`) Operators

The same `...` syntax serves two opposite purposes:

- **Spread:** Expands an iterable (like an array) or object properties into another array or object.
- **Rest:** Bundles the rest of function arguments into an array.

```
// Spread
const arr1 = [1, 2, 3];
const arr2 = [...arr1, 4, 5]; // [1, 2, 3, 4, 5]

// Rest
function sum(...numbers) {
  return numbers.reduce((acc, current) => acc + current, 0);
}
sum(1, 2, 3); // 6
```

ES Modules (`import` / `export`)

The official, standardized module system for JavaScript, allowing you to split your code into reusable, organized files.

```
// utils.js - Exporting functions
export const PI = 3.14;
export function add(a, b) {
  return a + b;
}
```

```
// main.js - Importing and using them
import { PI, add } from './utils.js';

console.log(PI); // 3.14
console.log(add(5, 10)); // 15
``````html
<!-- In HTML, you must load the entry script as a module -->
<script type="module" src="main.js"></script>
```
```

The `class` Syntax

A clear, modern syntax for creating objects using Object-Oriented patterns.

```
class Person {
  constructor(name) {
    this.name = name;
  }
  greet() {
    return `Hello, my name is ${this.name}.`;
  }
}

// Inheritance with `extends` and `super`
class Student extends Person {
  constructor(name, studentId) {
    super(name); // Call the parent constructor
    this.studentId = studentId;
  }
}

const jane = new Student('Jane Doe', 'S123');
console.log(jane.greet()); // "Hello, my name is Jane Doe."
```

Part 2: The Asynchronous Challenge

(Why we need modern patterns)

The Single-Threaded Problem

JavaScript has a **single call stack**, meaning it can only do one thing at a time.

A long-running task (like a complex calculation or a network request) would **freeze the entire user interface**.

How does the browser stay responsive?
Answer: A **non-blocking event loop**.

The Event Loop (Simplified)

JavaScript offloads long tasks to the browser. When the stack is empty, the Event Loop pushes completed tasks from a queue to the stack to be executed.

```
console.log('1. Start');

// This is offloaded to the browser, doesn't block
setTimeout(() => {
  console.log('3. Timeout Complete');
}, 0);

console.log('2. End');
```

Output:

1. Start
2. End
3. Timeout Complete

Part 3: Solving Asynchronicity

(From Promises to `async/await`)

Promises: The First Solution

A **Promise** is an object representing a future result. It allows us to "chain" asynchronous actions instead of nesting them in "callback hell."

```
fetch('https://jsonplaceholder.typicode.com/todos/1')
  .then(response => response.json()) // First action
  .then(data => {                    // Second action
    console.log(data); // { userId: 1, id: 1, title: '...', ... }
  })
  .catch(error => {                  // Handles any error in the chain
    console.error('Request failed:', error);
  });
```

`async/await`: The Modern Standard

`async/await` is syntactic sugar over Promises, letting us write asynchronous code that looks synchronous and is much easier to read.

- `async`: Makes a function return a Promise.
- `await`: Pauses the function execution until a Promise settles.

```
async function getTodo() {
  try {
    const response = await fetch('https://jsonplaceholder.typicode.com/todos/1');
    const data = await response.json();
    console.log(data);
  } catch (error) {
    console.error('Request failed:', error);
  }
}
```

Part 4: Powerful HTML5 APIs

(Building Modern Apps)

Fetch API

The modern, Promise-based standard for making network requests, replacing the older `XMLHttpRequest`. It simplifies fetching resources across the network.

```
async function getUser(id) {
  try {
    const response = await fetch(`https://jsonplaceholder.typicode.com/users/${id}`);

    if (!response.ok) { // Check for HTTP errors like 404
      throw new Error(`HTTP error! Status: ${response.status}`);
    }

    const userData = await response.json();
    console.log(userData);
  } catch (error) {
    console.error('Could not fetch user:', error);
  }
}

getUser(1); // Fetch and log the user with ID 1
```

Web Storage

Simple key-value storage in the browser.

- `localStorage`: Persists after the browser is closed.
- `sessionStorage`: Cleared when the browser tab is closed.

```
// Save a string
localStorage.setItem('theme', 'dark');

// Save an object (must be converted to a string)
const user = { name: 'Alice' };
localStorage.setItem('user', JSON.stringify(user));

// Get the data back
const theme = localStorage.getItem('theme'); // 'dark'
const userData = JSON.parse(localStorage.getItem('user')); // { name: 'Alice' }
```

Web Workers

Run a script on a **background thread** to prevent the UI from freezing during heavy computations.

```
// main.js
const worker = new Worker('worker.js');
worker.postMessage(10000); // Send data to worker
worker.onmessage = e => {   // Listen for result
  console.log('Result from worker:', e.data);
};
```

```
// worker.js
// This runs in the background
onmessage = e => {
  const result = e.data * 2;
  postMessage(result); // Send result back
};
```

Drag and Drop API

Native browser support for rich drag-and-drop interactions.

```
<div id="item" draggable="true">Drag Me</div>
<div id="zone">Drop Here</div>
```

```
const zone = document.getElementById('zone');
// Must prevent default behavior on dragover to allow dropping
zone.addEventListener('dragover', e => e.preventDefault());
zone.addEventListener('drop', e => {
  e.preventDefault();
  dropzone.textContent = 'Item Dropped!';
});
```

Canvas API

A blank slate for drawing 2D graphics, charts, and animations with JavaScript.

```
<canvas id="myCanvas" width="200" height="100"></canvas>
```

```
const canvas = document.getElementById('myCanvas');
const ctx = canvas.getContext('2d');

// Draw a blue rectangle
ctx.fillStyle = 'blue';
ctx.fillRect(10, 10, 150, 80);
```

Intersection Observer API

A highly performant way to detect when an element enters the screen. Perfect for lazy loading images or infinite scrolling.

```
<img data-src="image.jpg">
```

```
const image = document.querySelector('img');

const observer = new IntersectionObserver(entries => {
  if (entries[0].isIntersecting) {
    image.src = image.dataset.src; // Load the image
    observer.unobserve(image);   // Clean up
  }
});

observer.observe(image);
```

WebSockets API

Enables **real-time, two-way communication** between the client and server. Essential for chat apps, live notifications, and collaborative tools.

```
// Connect to a public echo server
const socket = new WebSocket('wss://echo.websocket.org');

// Listen for messages from the server
socket.onmessage = event => {
  // The server will echo back whatever we send
  console.log('Server says:', event.data);
};

// When the connection opens, send a message
socket.onopen = () => {
  socket.send('Hello from the client!');
};
```

Geolocation API

Allows a web app to request the user's geographical location.

```
navigator.geolocation.getCurrentPosition(
  position => {
    const { latitude, longitude } = position.coords;
    console.log(`You are at: ${latitude}, ${longitude}`);
  },
  () => {
    console.error('Could not get your location.');
  }
);
```

More APIs to Explore

- **IndexedDB:** A low-level API for client-side storage of significant amounts of structured data, including files/blobs. Far more powerful than `localStorage`.
- **History API:** Enables manipulation of the browser session history, which is crucial for building smooth client-side routing in Single-Page Applications (SPAs). (`pushState`, `replaceState`)
- **File API:** Allows web applications to asynchronously read the contents of files (or raw data buffers) on the user's computer.

Part 5: The Ultimate Goal

(Progressive Web Apps)

Web Components

The browser's native way to create **reusable, encapsulated UI components** with custom HTML tags.

```
class MyComponent extends HTMLElement {
  connectedCallback() {
    const shadow = this.attachShadow({ mode: 'open' });
    const style = document.createElement('style');
    style.textContent = 'p { color: purple; }';
    shadow.appendChild(style);
    const p = document.createElement('p');
    p.textContent = 'This is my custom web component!';
    shadow.appendChild(p);
  }
}
// Define the custom tag
customElements.define('my-component', MyComponent);
```

```
<!-- Now use it in your HTML -->
<my-component></my-component>
```

What is a Progressive Web App (PWA)?

A web app that uses modern APIs to deliver a native app-like experience. It is:

- **Reliable:** Works offline or on slow networks.
- **Fast:** Loads near-instantly.
- **Engaging:** Is "installable" on the user's home screen and can send push notifications.

PWA Core #1: The Web App Manifest

A `manifest.json` file that describes your app to the browser, enabling the "Add to Home Screen" feature.

```
// manifest.json
{
  "name": "My Awesome App",
  "short_name": "AwesomeApp",
  "start_url": "/",
  "display": "standalone",
  "background_color": "#ffffff",
  "theme_color": "#333333",
  "icons": [ { "src": "icon-512x512.png", "sizes": "512x512", "type": "image/png" } ]
}
```

```
<!-- Link it in your HTML -->
<link rel="manifest" href="/manifest.json">
```

PWA Core #2: The Service Worker

A script that runs in the background and acts as a network proxy. This is the key to **offline functionality**.

```
// main.js - Register the worker
if ('serviceWorker' in navigator) {
  navigator.serviceWorker.register('/sw.js');
}
```

```
// sw.js - Intercept network requests
self.addEventListener('fetch', event => {
  event.respondWith(
    // Try to find a match in the cache first
    caches.match(event.request)
      .then(response => response || fetch(event.request))
  );
});
```

PWA Core #3: The Notifications API

Allows your app to send system-level notifications to re-engage users. Permission must be granted by the user.

```
// Request permission first
Notification.requestPermission().then(permission => {
  if (permission === 'granted') {
    // Then show the notification (can be done from a Service Worker)
    new Notification('Hello from PWA!', {
      body: 'This is a push notification.',
      icon: '/icon-192x192.png'
    });
  }
});
```

Conclusion

- We use **modern JS syntax** (`ES6+`) to write clean, modular, and maintainable code.
- We use **`async/await`** to handle asynchronous tasks gracefully.
- We leverage powerful **HTML5 APIs** to build rich, interactive, and performant user experiences.
- We combine these technologies to build **Progressive Web Apps** that provide a first-class, reliable, and engaging experience for our users.