

{javascript}

Explorando o Funcionamento Completo do DOM em JavaScript

Willian de Mattos Silva



Introdução ao DOM

- O que é o DOM?
 - Abreviação de Document Object Model
 - Representa a estrutura do documento HTML/XML
 - Permite interação dinâmica e manipulação da estrutura do documento através do JavaScript

Estrutura do DOM

- Árvore de nós (nodes)
 - Cada elemento, atributo e texto do documento é representado como um nó.
 - Os nós estão interconectados formando uma estrutura hierárquica.



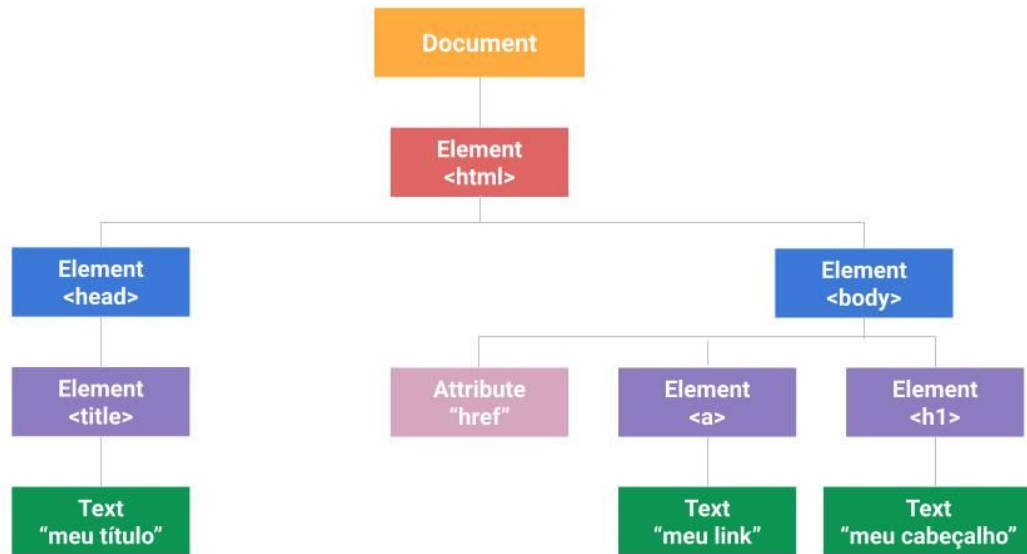
```
<html>
  <head>
    <title>Exemplo</title>
  </head>
  <body>
    <div id="container">
      <p>Olá, mundo!</p>
    </div>
  </body>
</html>
```

```
<!-- Nessa estrutura, temos nós para <html>,
<head>, <title>, <body>, <div>, <p>, e o
texto "Olá, mundo!". -->
```

Tipos de Nós no DOM

- Nós de Elemento: Representam elementos HTML como `<div>`, `<p>`, ``, etc.
- Nós de Atributo: Representam os atributos dos elementos.
- Nós de Texto: Representam o texto dentro dos elementos.
- Nós de Comentário: Representam comentários dentro do código HTML.
- Nós de Documento: Representam o documento inteiro.

Estrutura do DOM



Manipulação do DOM

- Como acessar elementos no DOM
- Métodos como **getElementById**, **getElementsByClassName**, **getElementsByTagName**, **querySelector**, **querySelectorAll**



```
// Acessando um elemento pelo ID
const container =
document.getElementById('container');


// Acessando elementos por classe
const elements =
document.getElementsByClassName('element');

// Acessando elementos por tag
const paragraphs =
document.getElementsByTagName('p');

// Acessando elementos com seletor CSS
const firstElement =
document.querySelector('.main .element');
```

Manipulação do DOM

- Atributos e propriedades para acessar e modificar elementos



```
// Acessando e modificando atributos
container.id = 'newId';
container.className = 'newClass';

// Acessando e modificando conteúdo
container.textContent = 'Novo conteúdo';

// Adicionando um novo elemento filho
const newParagraph =
document.createElement('p');
newParagraph.textContent = 'Outro
parágrafo';
container.appendChild(newParagraph);
```

Eventos no DOM

- O que são eventos?
- Exemplos de eventos: **click**, **mouseover**, **submit**, **etc.**
- Adicionando manipuladores de eventos usando **addEventListener**



```
// Adicionando um manipulador de eventos
container.addEventListener('click',
function() {
  console.log('Elemento clicado!');
});
```


Classes e Padrões no ES6

- Introdução às classes no ES6
 - Sintaxe simplificada para criação de objetos e herança
 - Uso de class, constructor, super
 - Exemplo de definição de uma classe e criação de instâncias



```
class Animal {  
  constructor(name) {  
    this.name = name;  
  }  
  
  speak() {  
    console.log(this.name + ' faz algum som.');  }  
}  
  
const dog = new Animal('Cachorro');  
dog.speak(); // Saída: Cachorro faz algum som.
```

Padrões no ES6

- Conceito de padrões de design:
- Padrões de design são soluções reutilizáveis para problemas comuns no desenvolvimento de software.
- Eles fornecem abordagens testadas e comprovadas para resolver problemas específicos de forma eficiente e elegante.

Padrão de Módulo

- O Padrão de Módulo permite organizar o código em módulos reutilizáveis, evitando poluição do namespace global e fornecendo encapsulamento.
- Exemplo de implementação em ES6:

```
// module.js
const myModule = (() => {
  const privateVariable = 'I am private';

  const privateFunction = () => {
    console.log('This is a private function');
  }

  const publicFunction = () => {
    console.log('This is a public function');
  }

  return {
    publicFunction
  };
})();

myModule.publicFunction(); // Saída: This is a public function
```

Padrão de Fábrica

- O Padrão de Fábrica é usado para encapsular a criação de objetos, permitindo a criação de objetos de diferentes tipos com uma interface uniforme.
- Exemplo de implementação em ES6:

```
class Car {  
  constructor(make, model) {  
    this.make = make;  
    this.model = model;  
  }  
}  
  
class CarFactory {  
  createCar(make, model) {  
    return new Car(make, model);  
  }  
}  
  
const carFactory = new CarFactory();  
const myCar = carFactory.createCar('Toyota',  
  'Corolla');
```

Padrão de Observador

- O Padrão de Observador é usado para definir um mecanismo de assinatura/observação para notificar objetos sobre mudanças no estado de outros objetos.
- Exemplo de implementação em ES6:

```
class Subject {
  constructor() {
    this.observers = [];
  }

  addObserver(observer) {
    this.observers.push(observer);
  }

  notifyObservers(data) {
    this.observers.forEach(observer =>
      observer.update(data));
  }
}

class Observer {
  update(data) {
    console.log('Received data:', data);
  }
}

const subject = new Subject();
const observer1 = new Observer();
const observer2 = new Observer();

subject.addObserver(observer1);
subject.addObserver(observer2);

subject.notifyObservers('Hello,
  observers!');
```

Conclusão

- Os padrões de design oferecem soluções elegantes e reutilizáveis para problemas comuns no desenvolvimento de software.
- No ES6, podemos implementar esses padrões de forma mais clara e concisa usando classes e outras características da linguagem.

NaN

Perguntas ?

{Obrigado}

