Innominatum CONSULTORIA

Programação em JavaScript

ServiceNow - Deloitte

Rodrigo Costa

IPCA POLITÉCNICO DO CÁVADO E DO AVE
INSTITUTO DO EMPREGO E FORMAÇÃO PROFISSIONAL
PRR Plano de Recuperação e Resiliência
REPÚBLICA PORTUGUESA
Financiado pela União Europeia NextGenerationEU
IAPMEI Parcerias para o Crescimento
PORTUGAL DIGITAL

Innominatum CONSULTORIA

JavaScript

Módulo 2

IPCA POLITÉCNICO DO CÁVADO E DO AVE
INSTITUTO DO EMPREGO E FORMAÇÃO PROFISSIONAL
PRR Plano de Recuperação e Resiliência
REPÚBLICA PORTUGUESA
Financiado pela União Europeia NextGenerationEU
IAPMEI Parcerias para o Crescimento
PORTUGAL DIGITAL

Reatividade

Sessão 10

Objetivo geral

Capacitar os formandos na implementação de interfaces reativas, dominando os mecanismos de deteção e resposta a interações do utilizador no ecossistema JavaScript. A sessão foca na transição de scripts lineares para uma arquitetura orientada a eventos, ensinando a utilizar o addEventListener como padrão de excelência para garantir a separação de responsabilidades (Decoupling). Os formandos aprenderão a explorar o Objeto Evento para capturar dados contextuais, a compreender a mecânica de propagação (Bubbling e Capturing) e a aplicar a técnica avançada de Event Delegation. O objetivo final é a escrita de código escalável e de alta performance, capaz de gerir interações complexas em estruturas de dados volumosas com o mínimo consumo de recursos.

Objetivos específicos

Ao final da sessão, os formandos deverão ser capazes de:

- Aplicar **addEventListener** para reagir a eventos do utilizador, evitando o uso de handlers inline em HTML;
- Utilizar o objeto **event** para aceder a informações sobre o evento, como target, teclas pressionadas e posição do rato;
- Explicar o fluxo de eventos no DOM, distinguindo **bubbling** de **capturing**;
- Aplicar a técnica de **event delegation** para gerir eventos de forma eficiente em listas e elementos dinâmicos.

Introdução a Eventos e Event Listeners

Innominatum CONSULTORIA
IPCA POLITÉCNICO DO CÁVADO E DO AVE
INSTITUTO DO EMPREGO E FORMAÇÃO PROFISSIONAL
PRR Plano de Recuperação e Resiliência
REPÚBLICA PORTUGUESA
Financiado pela União Europeia NextGenerationEU
IAPMEI Parcerias para o Crescimento
PORTUGAL DIGITAL

O que é um evento?

Um evento é uma ocorrência ou acontecimento no navegador, no documento ou em objetos associados, sobre o qual o navegador notifica o programa JavaScript. Eles são a base do modelo de programação assíncrona e dirigida por eventos, permitindo que o código responda a interações como o término do carregamento de uma página, o movimento do mouse sobre um link ou a ação de uma tecla no teclado.

Evento

Um evento então é uma ação que acontece na página e pode ser:

- Um clique;
- Uma tecla pressionada;
- Um movimento do rato;
- O envio de formulário.

EventListener

Os event listeners (ouvintes de eventos) no JavaScript são mecanismos que permitem capturar e responder a interações do usuário ou eventos do navegador. Eles são fundamentais para criar páginas web interativas.

addEventListener()

O addeventListener() é um método da interface EventTarget que permite registrar uma função (listener) para ser executada sempre que um evento específico ocorre em um determinado alvo do elemento do DOM. Ele associa um **evento** a uma **função** sem executar a função imediatamente.

addEventListener()- sintaxe oficial

```
target.addEventListener(type, listener, options);
```

Seu primeiro parâmetro type, está a espera do nome do evento que o método irá escutar (“click”, “submit”, “keydown”, “input”), o segundo parâmetro listener, será a função que precisamos criar para realizar uma ação em resposta ao evento. O options é o terceiro parâmetro do método, que é opcional e costuma ser utilizado como um objeto de configuração do método, onde podemos configurar por exemplo, uma única execução do evento.

JavaScrpit não executa sozinho

Até agora o JS executava linha a linha, sem nenhuma interrupção ou desvio de ciclo. Com os eventos o JS será capaz de ficar **à espera** de um acontecimento, em outras palavras, a espera de um evento acontecer, isto se chama **reatividade**.

Exemplo simples de evento

```
// Sintaxe básica
elemento.addEventListener("evento", funcao);
```

```
const botao = document.querySelector("button");

botao.addEventListener("click", () => {
  console.log("Botão clicado!");
});
```

O código só será executado depois que ocorre o clique.

Um evento não se limita ao clique

```
botao.addEventListener("mouseenter", () => {
  console.log("Entrou");
});

botao.addEventListener("mouseleave", () => {
  console.log("Saiu");
});
```

Aqui temos um evento relacionado a movimentação do rato dentro de uma caixa.

Eventos comuns

O JS possui uma infinidade eventos disponíveis, o que dificulta a apresentação de todos aqui, contudo, abaixo segue uma pequena lista dos eventos mais utilizados:

- click;
- input;
- submit;
- keydown;
- mouseover.

O objeto evento

Todo evento traz informação

Quando um evento acontece, o navegador:

- Cria um objeto;
- Envia esse objeto para a função;
- Esse objeto chama-se **event** (ou **e**).

```
botao.addEventListener("click", (event) => {
  console.log(event);
});
```

O event tem todos os detalhes desse evento de clique

event.target

```
document.addEventListener("click", (e) => {
  console.log(e.target);
});
```

Captura informações do elemento alvo como:

- A referência ao elemento que disparou o evento;
- Mostra exatamente qual elemento clicado.

Eventos de teclado

```
document.addEventListener("keydown", (e) => {
  console.log(e.key);
});
```

O keydown é o evento disparado no exato momento em que o utilizador pressiona uma tecla do teclado. Diferente do clique do rato, que acontece num ponto fixo do ecrã, o keydown é um evento focado na entrada de dados e no controlo de interface via teclado.

Eventos de rato

```
document.addEventListener("click", (e) => {
  console.log(e.clientX, e.clientY);
});
```

É a forma do JS detectar qualquer movimento, clique ou interação física que o utilizador faça com o ponteiro do rato. Destaque para métodos que capturam:

- Clique;
- Movimento;
- Estado do botão.

Fluxo de Eventos (Bubbling vs Capturing)

Eventos não acontecem isoladamente

Quando ocorre um evento ele não fica preso ao elemento, ele percorre a árvore **DOM**, esse fenômeno é conhecido como **Propagação de eventos**.

As três fases do evento

1. Capturing (captura);
2. Target (alvo);
3. Bubbling (borbulhamento);

O navegador percorre o DOM **duas vezes**.

Capturing (captura)

É a primeira fase de propagação, ela ocorre antes mesmo de o evento atingir seu alvo. O evento "desce" na hierarquia, começando no objeto Window, passando pelo Document e seguindo pelos ascendentes até chegar ao pai do alvo.

```
div.addEventListener("click", handler, true);
```

Target (alvo)

É a segunda fase, na qual as rotinas de tratamento registradas no próprio objeto onde o evento foi disparado originalmente são chamadas

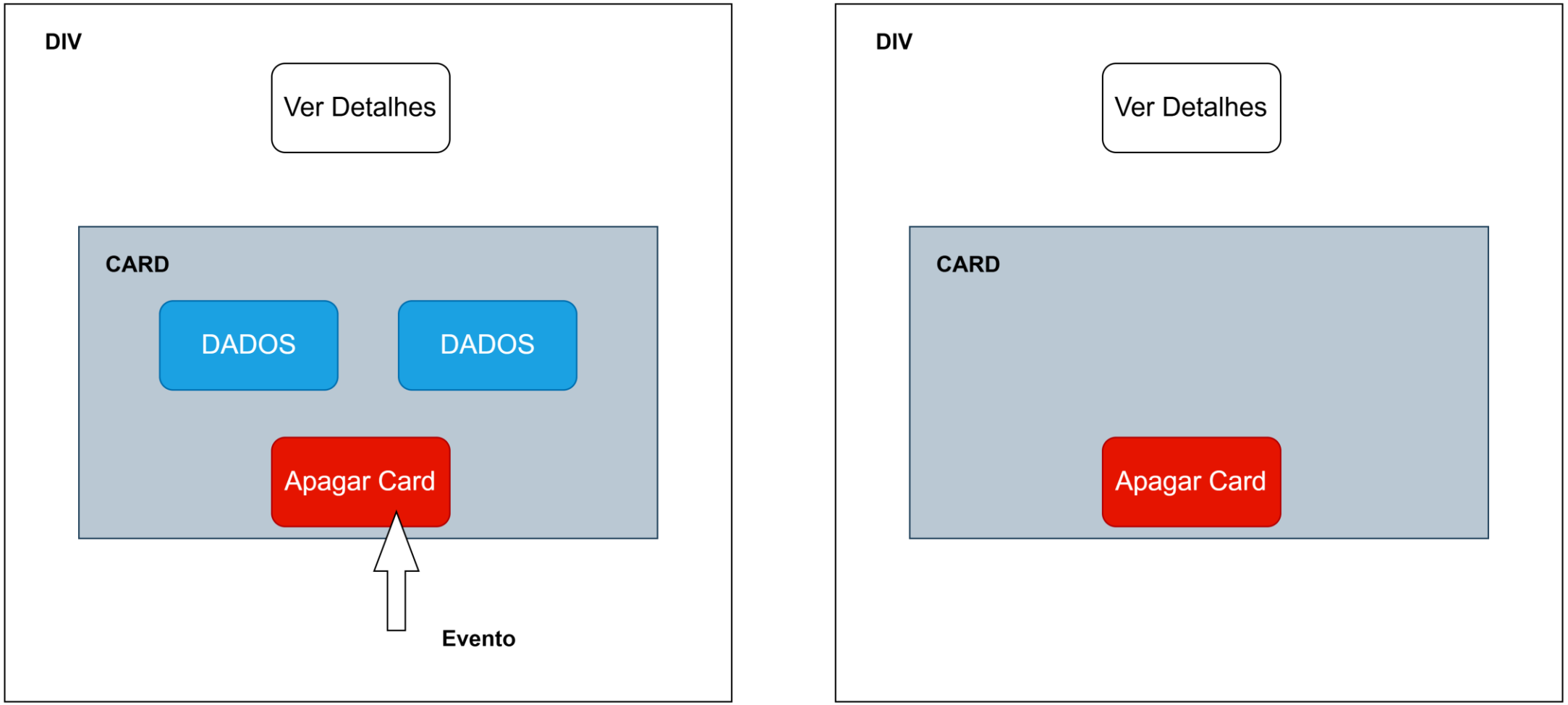
```
document.addEventListener("click", (e) => {
  console.log(e.target);
});
```

Bubbling (borbulhamento)

É a terceira e última fase, onde o evento "sobe" na árvore DOM, partindo do pai do alvo em direção ao topo da hierarquia, passando pelos blocos até atingir o objeto Window.

```
div.addEventListener("click", () => {
  console.log("DIV");
});
```

Bubbling (borbulhamento)



Parar a propagação

```
button.addEventListener("click", (e) => {
  e.stopPropagation();
});
```

Felizmente existe um método que usaremos para parar a propagação que é chamado de **stopPropagation**. O método é uma função do objeto **event** utilizada para impedir que um evento continue a se propagar pela árvore de objetos do documento (DOM)

Event Delegation

O problema dos muitos listeners

Imagina uma lista que possa conter 50,

100, ou 500 itens, e imagine ter de

adicionar um addEventListener em cada

item? Essa ação trás problemas como:

- Código poluído;
- Problemas de performance;
- Dificuldade de manutenção.

```
document.querySelectorAll("li")
.forEach(item => {
  item.addEventListener("click",
    () => {
    console.log("Item clicado");
  });
});
```

Event Delegation – Definição

```
const lista = document.querySelector("ul");

lista.addEventListener("click", (e) => {
  if (e.target.tagName === "LI") {
    console.log("Item clicado:", e.target.textContent);
  }
});
```

O **event delegation** é uma técnica de programação que utiliza o processo de **borbulhamento (propagação de baixo para cima)** para gerenciar eventos de forma mais eficiente e dinâmica.

Event Delegation – Casos de uso

- Listas;
- Tabelas;
- Menus;
- Cards dinâmicos;
- Resultados de pesquisa.

Síntese

- Definição de eventos no JavaScript;
- Utilização do objeto event para obter contexto sobre o evento ocorrido;
- Entendimento do percurso dos eventos pela árvore do DOM;
- Aplicação da técnica de escutar eventos no elemento pai para gerir múltiplos elementos filhos.

Innominatum CONSULTORIA

Conclusão

IPCA POLITÉCNICO DO CÁVADO E DO AVE
INSTITUTO DO EMPREGO E FORMAÇÃO PROFISSIONAL
PRR Plano de Recuperação e Resiliência
REPÚBLICA PORTUGUESA
Financiado pela União Europeia NextGenerationEU
IAPMEI Parcerias para o Crescimento
PORTUGAL DIGITAL

Innominatum CONSULTORIA

Programação em JavaScript

ServiceNow - Deloitte

Rodrigo Costa

IPCA POLITÉCNICO DO CÁVADO E DO AVE
INSTITUTO DO EMPREGO E FORMAÇÃO PROFISSIONAL
PRR Plano de Recuperação e Resiliência
REPÚBLICA PORTUGUESA
Financiado pela União Europeia NextGenerationEU
IAPMEI Parcerias para o Crescimento
PORTUGAL DIGITAL