Comentários

samuelamaro96746313

March 2021

Comentários

Manipuladores de JavaScript inline

Note que às vezes você vai encontrar código JavaScript escrito dentro do HTML. Isso deve ser algo como

```
    https://www.overleaf.com/project/60592aee71d1c6eeb729d2ca
        function criarParagrafo() {
          let para = document.createElement('p');
          para.textContent = 'Voc  clicou o botao!';
          document.body.appendChild(para);
        }

        <button onclick="criarParagrafo()">Me clique!</button>

```

Listing 1: Codigo javascript escrito dentro do HTML

Essa demonstração tem exatamente a mesma funcionalidade que vimos nas primeiras duas seções, exceto que o elemento `<button>|` inclui um manipulador inline onclick para fazer a função ser executada quando o botão é clicado.

Contudo, por favor, não faça isso. É uma má prática poluir seu HTML com JavaScript, e isso é ineficiente — você teria que incluir o atributo `onclick="criarParagrafo()"` em todo botão que você quisesse aplicar JavaScript.

Usando uma estrutura feita de puro JavaScript permite a você selecionar todos os botões usando uma instrução. O código que nós usamos acima para servir a esse propósito se parece com isso:

```

        const botoes = document.querySelectorAll('button');
```

```
3
4         for(var i = 0; i < botoes.length ; i++) {
5             botoes[i].addEventListener('click', criarParagrafo);
6         }
7
8
```

Isso talvez parece ser mais do que o atributo `onclick`, mas isso vai funcionar para todos os botões, não importa quantos tem na página, e quantos forem adicionados ou removidos. O JavaScript não precisará ser mudado.

Estratégias para o carregamento de scripts

Há um considerável número de problemas envolvendo o carregamento de scripts na ordem correta. Infelizmente, nada é tão simples quanto parece ser! Um problema comum é que todo o HTML de uma página é carregado na ordem em que ele aparece. Se você estiver usando Javascript para manipular alguns elementos da página (sendo mais preciso, manipular o **DocumentDocument Object Model**), seu código não irá funcionar caso o JavaScript for carregado e executado antes mesmo dos elementos HTML estarem disponíveis.

Nos exemplos acima, tanto nos scripts internos ou externos, o JavaScript é carregado e acionado dentro do cabeçalho do documento, antes do corpo da página ser completamente carregado. Isso poderá causar algum erro. Assim, temos algumas soluções para isso.

No exemplo interno, você pode ver essa estrutura em volta do código:

```
1
2         document.addEventListener("DOMContentLoaded", function() {
3             ...
4         });
5
6
```

Isso é um event listener (ouvidor de eventos), que ouve e aguarda o disparo do evento "DOMContentLoaded"vindo do browser, evento este que significa que o corpo do HTML está completamente carregado e pronto. O código JavaScript que estiver dentro desse bloco não será executado até que o evento seja disparado, portanto, o erro será evitado.

No exemplo externo, nós usamos um recurso moderno do JavaScript para resolver esse problema: Trata-se do atributo `defer`, que informa ao browser para continuar renderizando o conteúdo HTML uma vez que a tag `<script>|` foi atingida.

```
1
2         <script src="script.js" defer></script>
```

```
3
4
```

Neste caso, ambos script e HTML irão carregar de forma simultânea e o código irá funcionar.

Nota: No caso externo, nós não precisamos utilizar o evento `DOMContentLoaded` porque o atributo `defer` resolve o nosso problema. Nós não utilizamos defer como solução para os exemplos internos pois defer funciona apenas com scripts externos.

Uma solução à moda antiga para esse problema era colocar o elemento script bem no final do body da página (antes da tag `</body>|`). Com isso, os scripts iriam carregar logo após todo o conteúdo HTML. O problema com esse tipo de solução é que o carregamento/renderização do script seria completamente bloqueado até que todo o conteúdo HTML fosse analisado. Em sites de maior escala, com muitos scripts, essa solução causaria um grande problema de performance e deixaria o site lento.

async e defer

Atualmente, há dois recursos bem modernos que podermos usar para evitar o problema com o bloqueio de scripts — `async e defer` (que vimos acima). Vamos ver as diferenças entre esses dois?

Os scripts que são carregados usando o atributo `async` (veja abaixo) irão baixar o script sem bloquear a renderização da página e irão executar imediatamente após o script terminar de ser disponibilizado. Nesse modo você não tem garantia nenhuma que os scripts carregados irão rodar em uma ordem específica, mas saberá que dessa forma eles não irão impedir o carregamento do restante da página. O melhor uso para o async é quando os scripts de uma página rodam de forma independente entre si e também não dependem de nenhum outro script.

Por exemplo, se você tiver os seguintes elementos script:

```
1
2       <script async src="js/vendor/jquery.js"></script>
3
4       <script async src="js/script2.js"></script>
5
6       <script async src="js/script3.js"></script>
7
8
```

Listing 2: Exemplo de carregamento de script externo em um documento HTML

Você não pode garantir que o script. `jquery.js` carregará antes ou depois do `script2.js` e `script3.js` . Nesse caso, se alguma função desses scripts dependerem de algo vindo do jquery, ela produzirá um erro pois o jquery ainda não foi definido/carregado quando os scripts executaram essa função.

async deve ser usado quando houver muitos scripts rodando no background, e você precisa que estejam disponíveis o mais rápido possível. Por exemplo, talvez você tenha muitos arquivos de dados de um jogo para carregar que serão necessários assim que o jogo iniciar, mas por enquanto, você só quer entrar e ver a tela de carregamento, a do titulo do jogo e o lobby, sem ser bloqueado pelo carregamento desses scripts.

Scripts que são carregados utilizando o atributo `defer` (veja abaixo) irão rodar exatamente na ordem em que aparecem na página e serão executados assim que o script e o conteúdo for baixado.

```
1
2       <script defer src="js/vendor/jquery.js"></script>
3
4       <script defer src="js/script2.js"></script>
5
6       <script defer src="js/script3.js"></script>
7
8
```

Listing 3: Carregamento de script externo em HTML usando atributo defer

Todos os scripts com o atributo defer irão carregar na ordem que aparecem na página. No segundo exemplo, podemos ter a certeza que o script `jquery.js` irá carregar antes do `script2.js` e `script3.js` e o `script2.js` irá carregar antes do `script3.js`. Os scripts não irão rodar sem que antes todo o conteúdo da página seja carregado, que no caso, é muito útil se os seus scripts dependem de um DOM completamente disponibilizado em tela (por exemplo, scripts que modificam um elemento).

Resumindo:

- `async` e `defer` istruem o browser a baixar os scripts numa thread (processo) á parte, enquanto o resto da página (o DOM, etc.) está sendo baixado e disponibilizado de forma não bloqueante.
- Se os seus scripts precisam rodar imediatamente, sem que dependam de outros para serem executados, use `async`.
- Se seus scripts dependem de outros scripts ou do DOM completamente disponível em tela, carregue-os usando `defer` e coloque os elementos `<script>|` na ordem exata que deseja que sejam carregados.

Assim como HTML e CSS, é possível escrever comentários dentro do

seu código JavaScript que serão ignorados pelo navegador, e existirão simplesmente para prover instruções aos seus colegas desenvolvedores sobre como o código funciona (e pra você, se você tiver que voltar ao seu código depois de 6 meses e não se lembrar do que fez). Comentários são muito úteis, e você deveria usá-los frequentemente, principalmente quando seus códigos forem muito grandes. Há dois tipos:

- Um comentário de uma linha é escrito depois de duas barras. Por exemplo:

```
1
2           // Eu sou um comentario
3
4
```

Listing 4: Exemplo de comentario de linha em JavaScript

- Um comentário de múltiplas linhas é escrito entre os caracteres `/*` e `*/`. Por exemplo:

```
1
2           /*
3             Eu tambem sou
4             um comentario
5           */
6
7
```

Listing 5: Exemplo de comentario de multiplas linhas em JavaScript

Então, por exemplo, você poderia fazer anotações na nossa última demonstração de código JavaScript, da seguinte forma:

```
1
2       // Funcao: Cria um novo paragrafo e o insere no fim do
    arquivo HTML.
3
4       function criarParagrafo() {
5         var para = document.createElement('p');
6         para.textContent = 'Voce clicou no botao!';
7         document.body.appendChild(para);
8       }
9
10      /*
11        1. Captura referencias de todos os botoes na pagina e
    armazena isso em um array.
12        2. Vai at  todos os botees e adiciona um event listener
    click a cada um deles.
13
14        Quando cada botao e clicado, a funcao criarParagrafo()
    sera executada.
15      */
16
17      const botoes = document.querySelectorAll('button');
```

```
18
19          for(var i = 0; i < botoes.length ; i++) {
20            botoes[i].addEventListener('click', criarParagrafo);
21          }
22
23
```