03

Evitando percorrer o DOM muitas vezes

Transcrição

O código funciona, mas se adicionarmos dez negociações e clicarmos dez vezes em "Incluir", o `querySelector` buscará a `#data` , `#quantidade` e `#valor` dez vezes também. No entanto, devemos evitar ao máximo percorrer o DOM. Sempre que executamos o `querySelector` , ele irá no DOM - que é uma árvore de elementos. O `querySelector` terá que buscar entre todos estes elementos. Porém, a API de DOM é um tanto "preguiçosa" e não gosta de executar essa ação.

Então, se adicionamos 100 negociações, teremos que fazer esta busca 100 vezes. Apesar de só estarmos trabalhando com três elementos no caso, queremos melhorar a nossa performance. Atualmente, nosso código está assim:

```
class NegociacaoController {

  adiciona(event) {
        event.preventDefault();

        let $ = document.querySelector.bind(document);
        let inputData = $('#data');
        let inputQuantidade =  $('#quantidade');
        let inputValor = $('#valor');

        console.log(inputData.value);
        console.log(inputQuantidade.value);
        console.log(inputValor.value);

  }
}
```

Para melhorar a performance, adicionaremos o `constructor` e moveremos os `input` s para dentro dele. Mas em vez de criarmos uma variável, criaremos atributos de instâncias com o `this` .

```
class NegociacaoController {

  constructor() {

          let $ = document.querySelector.bind(document);

          this.inputData = $('#data');
          this.inputQuantidade =  $('#quantidade');
          this.inputValor = $('#valor');
  }

  adiciona(event) {
        event.preventDefault();


        console.log(this.inputData.value);
        console.log(this.inputQuantidade.value);
        console.log(this.inputValor.value);
```

```
    }
}
```

Quando o `NegociacaoController` for criado pela primeira vez, ele buscará os elementos do DOM do `document`, que serão guardados nas propriedades da classe. Adicionamos o `this` no `console` também.

Agora, mesmo que façamos 300 negociações, ele só fará uma busca no DOM pelos elementos. Com isto, conseguimos melhorar a performance. O impacto neste cenário é ínfimo, mas poderia ser maior em outros cenários. Se executarmos o código, veremos que será exibido corretamente no console.

Usaremos esta estratégia como se fosse um caching até o fim do curso.