10

Interpolação e data binding

Transcrição

Voltando para a declaração do nosso componente `App`, na parte que declara seu template, ou seja, sua apresentação temos a seguinte tag `h1`:

```
<!-- alurapic/src/App.vue -->
<!-- código anterior omitido -->

<h1>{{ msg }}</h1>

<!-- código posterior omitido -->
```

Se fizermos um "de, para" deste trecho do nosso template com o que visualizamos no browser, em seu lugar é exibido o texto "Welcome to Your Vue.js App". Como isso é possível? Para que possamos compreender essa mágica, vou apelar para uma analogia com o trabalho de um advogado.

Um advogado quando cria um memorando ou coisa parecida ele não faz do zero, ele usa um `template` do documento que deseja redigir personalizando apenas algumas lacunas desse documento. Não é à toa que o nome da nossa tag se chama `template`. Nela, lacunas são expressas através de `{{ }}` e a informação entre as chaves duplas é a informação que o template necessita para ficar completo. No caso, precisamos da informação `msg`, ou melhor, do dado `msg`. No caso de `App`, é o próprio componente que disponibiliza o dado de que seu template precisa através da função `data` lá dentro da tag `<script>`. Vejamos:

```
// alurapic/src/App.vue
// código anterior omitido

  data () {
    return {
      msg: 'Welcome to Your Vue.js App'
    }
  }

// código posterior omitido
```

Em `App.vue`, disponibilizamos dados para o template através da função `data`. Essa função sempre retorna um objeto e as propriedades desse objeto são acessíveis no template do componente usando a sintaxe `{{ }}`.

Interpolação e data binding

O resultado da sintaxe especial `{{ }}` com o nome da propriedade que desejamos ler é chamada de **interpolação**. Dizemos que o dado `msg` foi interpolado no template. Por fim, essa interpolação segue uma regra: os dados fluem sempre da sua origem para o template e nunca o caminho contrário.

Tecnicamente falando, o que a interpolação faz é uma associação de dados unidirecional chamada `data binding`. Aliás, uma característica dessa associação é que qualquer mudança no dado gera automaticamente uma atualização no

template do componente. Como o componente é exibido dentro de uma `view`, no caso `index.html`, podemos dizer que mudanças nos dados acarretam uma atualização da view.

Podemos fazer um teste alterando o valor da propriedade `msg` por outro qualquer: primeiro módulo caso você queria se familiarizar ainda mais com a sintaxe apresentada com logo do curso.

```
<!-- alurapic/src/App.vue -->
<!-- código anterior omitido -->

<script>
export default {
  name: 'app',
  data () {
    return {
      msg: 'Seja bem-vindo à sua  Vue.js App'
    }
  }
}
</script>

<!-- código posterior omitido -->
```

Veja que ao alterarmos o valor de `msg` do objeto retornado pela função `data` do nosso componente, mudamos o que é apresentado para o usuário. Mas espere um pouco, essa mudança entrou em vigor sem precisarmos recarregar nossa página. Como isso é possível?

Live reloading

Qualquer alteração que fizemos nossos arquivos do nosso projeto gerará um novo bundle em memória e fará com que o navegador recarregue automaticamente para refletir nossas últimas alterações. Isso só é possível porque o servidor criado pelo Vue CLI suporta `LiveReloading`.

Agora que já temos uma visão geral de como a aplicação é estruturada, inclusive alguns conceitos importantes do Vue.js, já podemos adequar nossa aplicação para ser tornar realmente a Alurapic!