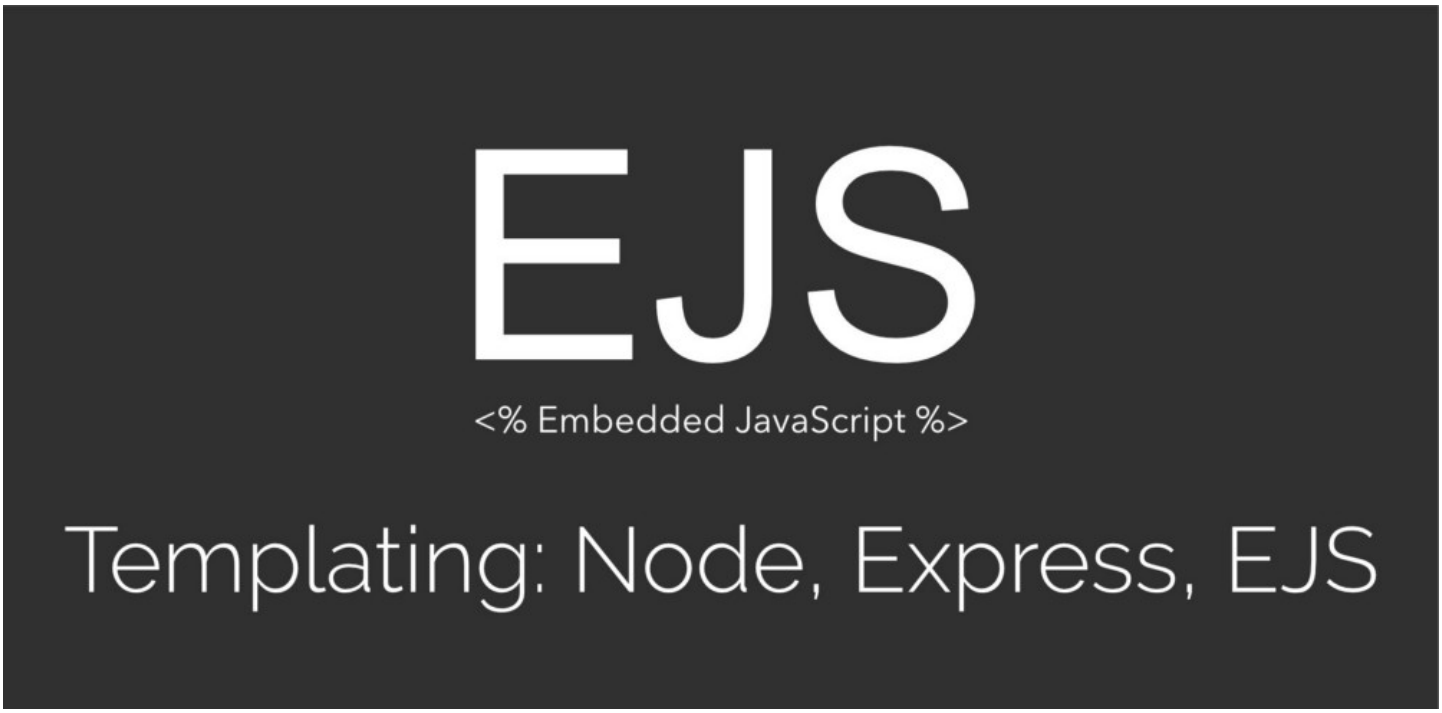
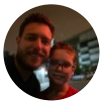


Node.js Express (4.x) EJS — Como usar funções durante a renderização



Marcelo Vismari
Sep 18, 2019 · 3 min read

Como chamar uma função de uma biblioteca ou de uma classe do próprio projeto durante a renderização da view utilizando EJS com Node.js Express?

Fazendo uma pequena introdução, Express é um framework MVC que roda em cima do Node-js e EJS (Embedded JavaScript templating) é um engine de renderização de views.

> *Express is a minimal and flexible Node.js web application framework that provides a robust set of features for web and mobile applications.*

Indo direto ao ponto, considere o código abaixo `server.js` :

```
const express = require('express');
const path = require('path');
const app = express();
```

```
app.set('view engine', 'ejs');
app.set('views', path.join(__dirname, '/'));

app.get('/', (req, res) => {
    var dados = ['valor 1', 'valor 2', 'valor 3', 'valor 4'];
    res.render('myView', { dadosDaView: dados });
});

app.listen(3000, () => console.log(`App listening on port!`));
```

Template myView.ejs:

```
<% for(let item of dadosDaView) { %>
    <div>Item: <%= item %></div>
<% } %>
```

Quando acessamos http://localhost:3000/ o resultado será:

```
<div>Item: valor 1</div>
<div>Item: valor 2</div>
<div>Item: valor 3</div>
<div>Item: valor 4</div>
```

Agora vamos adicionar uma função ao nosso arquivo `server.js` chamada `formatarValor`:

```
const express = require('express');
const path = require('path');
const app = express();

app.set('view engine', 'ejs');
app.set('views', path.join(__dirname, '/'));

function formatarValor(entrada) {
    return entrada.toUpperCase();
}

app.get('/', (req, res) => {
    var dados = ['valor 1', 'valor 2', 'valor 3', 'valor 4'];
    res.render('myView', { dadosDaView: dados });
});

app.listen(3000, () => console.log(`App listening on port!`));
```

Agora vamos utilizar a função dentro da view:

```
<% for(let item of dadosDaView) { %>
    <div>Item: <%= formatarValor(item) %></div>
<% } %>
```

O resultado:

```
ReferenceError: ...\myView.ejs:2
    1| <% for(let item of dadosDaView) { %>

>> 2|     <div>Item: <%= formatarValor(item) %></div>

3| <% } %>

4|

formatarValor is not defined
    at eval (eval at compile
(...)
```

Como o motor de renderização atua de forma isolada, ou seja, precisa receber todas as informações necessárias para concluir sua renderização, a função `formatarValor` não foi reconhecida.

Seguindo a mesma lógica de como passamos os dados a view (`res.render("myView", { dadosDaView: … });`) vamos passar também a função como parâmetro:

```
...

app.get('/', (req, res) => {
    var dados = ['valor 1', 'valor 2', 'valor 3', 'valor 4'];
    res.render('myView', { dadosDaView: dados, formatarValor:
formatarValor });
});

...
```

O resultado será:

```
<div>Item: VALOR 1</div>
<div>Item: VALOR 2</div>
<div>Item: VALOR 3</div>
<div>Item: VALOR 4</div>
```

Esta é uma das formas de se utilizar uma função durante a renderização da view, porém como poderíamos fazer se fosse necessário utilizar esta função em outras views? Teríamos que passar a função por parâmetro em cada chamada da view?

Há uma forma mais prática de executar esta tarefa utilizando um dos seguintes recursos que o Express nos fornece:

- app.locals
- res.locals

Os dois objetos tem a mesma ideia diferindo apenas no ciclo de vida. O `app.locals` persiste ao longo da aplicação toda durante toda sua vida, já o `res.locals` persiste apenas durante o ciclo de vida de um response.

A forma de utilizá-los é bem simples:

```
function formatarValor(entrada) {
    return entrada.toUpperCase();
}

app.locals.formatarValor = formatarValor; // <<<

app.get('/', (req, res) => {
    var dados = ['valor 1', 'valor 2', 'valor 3', 'valor 4'];
    res.render('myView', { dadosDaView: dados });
});
```

Neste caso a função `formatarValor` ficará disponível para todas as views.

Já para informações mais ligadas ao escopo da requisição como dados do usuário logado, preferências do usuário, caminho da requisição, entre outros, é mais indicado utilizar o `res.locals` . Veja um exemplo simples de um middleware que é executado antes dos *controllers* com o objetivo de validar um Json Web Token e disponibilizar as informações ao longo da resposta da requisição:

```
const express = require('express');
const path = require('path');
const app = express();

(...)

app.use(function (req, res, next) {
    const token = req.get('my_auth_token');
    var jwt = require('jsonwebtoken');
    jwt.verify(token, 'shhhhh', function(err, decoded) {
        if (err) {
            res.locals.authenticated = false;
            return next();
        }

        res.locals.authenticated = true;
        res.locals.user = decoded.user;
        next();
    });
});

(...)
// Mapeamento das rotas
(...)
```

Considerações

As formas abordadas neste post podem ajudar no reaproveitamento de código visto que as informações podem ser compartilhadas entre o processamento da requisição e o processamento da view.

Além das formas abordadas também podemos externalizar funções em arquivos com a extensão `.ejs` e importá-los direto na view utilizando o comando `<%- include("minhaFuncao") -%>` mas isto é assunto para outro post.

Como sugestão, sempre que possível façam uma visita á documentação oficial: https://expressjs.com/en/4x/api.html

Nodejs Mvc Express Ejs JavaScript


Download on the
App Store


GET IT ON
Google Play