02

Usar pré-processador dá trabalho. Será?

Começando deste ponto? Você pode fazer o DOWNLOAD (https://s3.amazonaws.com/caelum-online-public/gulp/stages/07-capitulo.zip) completo do projeto do capítulo anterior e continuar seus estudos a partir deste capítulo. Dentro da pasta **projeto**, não esqueça de executar no terminal o comando **npm install** para baixar novamente todas as dependências.

Neste treinamento estamos aprendendo a otimizar tarefas para sermos mais produtivos. Em termos de produtividade, será que podemos tornar o nosso projeto ainda melhor?

Neste projeto, já temos uma quantidade significativa de arquivos HTML, CSS e JavaScript escritos. Muitas vezes esses arquivos não são escritos por nós e vale dar uma olhada para ver se podemos tornar sua legibilidade e manutenção ainda melhores.

Código duplicado, problemas na manutenção

Vamos dar uma olhada no arquivo **projeto/src/estilos.css**. Temos uma cor em hexadecional definida em duas propriedades. E se a cor mudar? Teremos que lembrar de atualizar nesses dois lugares.:

```
.busca,
.menu-departamentos {
  background-color: #dcdcdc;
  font-weight: bold;
  text-transform: uppercase;
  margin-right: 10px;
  width: 230px;
}

.menu-departamentos ul li ul li {
  background-color: #dcdcdc;
}
```

Uma das causas de problema de manutenção de código é a duplicação de código, não queremos isso, certo?

Código complicado, mais problemas ainda

Escrutinando ainda mais `estilos.css` uma propriedade chama atenção, a `text-shadow` . Você lembra de cabeça quais são seus parâmetros? Mesmo lendo no arquivo CSS você lembra da ordem? O primeiro parâmetro é o quanto queremos descer o elemento, o segundo o quanto queremos afastá-lo para a direita, tudo relativo à posição do elemento. O terceiro é o `blur` , isto é, o esmaecimento do texto. Por fim, o último parâmetro é a cor da sombra:

```
.painel h2 {
  text-shadow: 1px 1px 2px rgba(255, 255, 255, 0.8);
}
...
.painel button {
  /* código omitido */
  text-shadow: 1px 0 1px black;
```

```
  transition: 0.3s;
}
```

Pré-processadores podem nos ajudar

Em linguagens de programação como JavaScript podemos criar funções que isolam determinada complexidade de código, mas infelizmente o mundo CSS ainda não suporta este recurso. Será que precisamos esperar o futuro para que possamos organizar ainda melhor nosso código? Com certeza não.

Para resolver problemas como esses, foram criados pré-processadores CSS. O conceito é simples: escrevemos um CSS usando a linguagem do pré-processador, mas como ela não é suportada pelo navegador, precisamos compilar o CSS transformando toda aquela sintaxe específica em sintaxe puramente CSS. Há também pré-processadores JavaScript que seguem o mesmo conceito.

Um pouquinho sobre LESS

No mercado há o Sass, Stylus, CSS-Crush, Myth, Less entre outros. Que tal experimentarmos um deles, por exemplo, o LESS?

Primeiro, vamos criar uma pasta chamada `projeto/src/less` e criar o arquivo `estilos.less` copiando para dentro dele todo o conteúdo `projeto/src/css/estilos.css` . Depois de copiarmos o conteúdo, vamos apagar **projeto/src/css/estilos.css**.

Agora que já temos nosso arquivo `less` , vamos resolver o problema da duplicação da nossa cor criando uma variável, **logo no início do arquivo**:

estilos.less

```
@cor-especial: #dcdcdc;
```

Este é um curso de Less ou de Gulp? De Gulp claro, mas não custa nada dar uma pequena explicação. Variáveis em Less são declaradas usando `@` como prefixo. A atribuição de valor é feita pelo dois pontos seguido do valor.

Agora, podemos substituir todos os lugares que nosso hexadecimal aparece por nossa variável:

```
.busca,
.menu-departamentos {
  background-color: @cor-especial;
  font-weight: bold;
  text-transform: uppercase;
  margin-right: 10px;
  width: 230px;
}
...

.menu-departamentos ul li ul li {
  background-color: @cor-especial;
}
```

Podemos fazer ainda mais coisa, por exemplo, criar uma função, que no mundo Less chamamos de `mixin` para definir a sombra do texto. Depois de criada, podemos usar em qualquer local do nosso arquivo `estilo.less` :

estilos.css

```
@cor-especial: #dcdcdc;

.texto-sombreado(@top, @left, @blur @cor) {
    text-shadow: @top @left @blur @cor;
}
```

Usamos como nome do nosso `mixin` uma declaração parecida com a de uma classe, com a diferença de que ela recebe parâmetros como se fosse uma função. Seus parâmetros são prefixados com `@` , assim como fizemos com nossa variável `@cor-especial` . Com essa modificação, fica evidente quais são os parâmetros que serão usados pela propriedade `text-shadow` .

Por fim, vamos substuir a propriedade `text-shadow` em todos os lugares que aparece em nosso `projeto/src/lessestilos.less` passando os parâmetros corretos em cada situação:

```
...

.painel h2 {
  .texto-sombreado(1px, 1px, 2px, rgba(255, 255, 255, 0.8));
}

...
.painel button {
  /* código omitido */
  .texto-sombreado(1px, 0, 1px, black);
  transition: 0.3s;
}
```

Compilação manual

Excelente, mas e agora? Como compilar esse arquivo? Precisamos do compilador `lessc` . Sua instalação varia de plataforma para plataforma. Na minha máquina eu já tenho instalado, mas veremos logo logo que não é necessário gastar tempo instalando-o. O mais importante aqui é você ver os passos que precisamos fazer para compilar nosso arquivo manualmente.

Bom, vamos para o terminal e entrando na pasta do nosso projeto. Como o `lessc` está instalado, compilamos nosso arquivo `estilo.less` assim:

terminal e dentro da pasta projeto

```
lessc src/less/estilos.less src/css/estilos.css
```

O comando `lessc` gerará o arquivo `projeto/src/css/estilos.css` . Funciona, pois havíamos apagado o arquivo antes e agora um novo aparece.

Abrindo o novo arquivo, vemos que não há mixin nem variável, pois tudo foi transformado para CSS. E como nosso navegador já importa o arquivo `estilos.css`, nenhuma outra modificação foi necessária.

A grande questão é que toda modificação precisará ser feita em `estilos.less` e não em `estilos.css`. Além disso, não podemos nos esquecer de abrir o terminal e compilar o arquivo manualmente a cada alteração.

Automatizar é a solução, mais uma vez

Talvez, seja essa a burocracia que afasta muitos desenvolvedores do mundo dos pré-processadores, mas se automatizamos uma série de tarefas com o Gulp, por que não fazer a mesma coisa com a compilação `less`?

O primeiro passo é instalar o módulo `gulp-less`:

```
npm install gulp-less@3.0.3 --save-dev
```

E claro, importá-lo em nosso `gulpfile.js`:

gulfile.js

```
var gulp = require('gulp')
  ,imagemin = require('gulp-imagemin')
  ,clean = require('gulp-clean')
  ,concat = require('gulp-concat')
  ,htmlReplace = require('gulp-html-replace')
  ,uglify = require('gulp-uglify')
  ,usemin = require('gulp-usemin')
  ,cssmin = require('gulp-cssmin')
  ,browserSync = require('browser-sync')
  ,jshint = require('gulp-jshint')
  ,jshintStylish = require('jshint-stylish')
  ,csslint = require('gulp-csslint')
  ,autoprefixer = require('gulp-autoprefixer')
  ,less = require('gulp-less');
```

Precisamos compilar qualquer arquivo LESS que seja alterado, para isso, vamos adicionar um novo `watcher` para arquivos `less`:

```
gulp.task('server', function() {
    browserSync.init({
        server: {
            baseDir: 'src'
        }
    });

   // wachers anteriores omitidos

    gulp.watch('src/less/**/*.less').on('change', function(event) {
       gulp.src(event.path)
            .pipe(less())
            .pipe(gulp.dest('src/css'));
    });
```

Não há muita novidade até aqui. Nosso watcher será disparado toda vez que um arquivo `less` for modificado. Para o arquivo modificado, executaremos `less()` gravando o resultado na pasta `src/css` . Vamos testar? Mas primeiro, vamos apagar o arquivo `projeto/src/css/estilos.css` para que possamos vê-lo sendo gerado por nossa tarefa.

Agora, basta executarmos a tarefa `server` no terminal:

```
npm run gulp server
```

Streams podem explodir

Agora, vamos alterar e salvar o arquivo `estilos.less` , isso será suficiente para que ele seja compilado. O resultado é o arquivo `projeto/src/css/estilos.css` gerado, perfeito! Como estamos usando o `livereloading` do BrowseSync o arquivo `css` gerado será sempre recarregado pelo navegador, isto é, basta alterarmos o arquivo `less` que ele será compilado e o `css` resultante será recarregado pelo navegador. Vamos testar isso.

Vamos alterar a cor da variável para `red` :

estilos.css

```
/* erro intencional, está faltando os dois pontos */
@cor-especial red;
```

O que aconteceu? Nosso servidor foi derrubado! Qual a razão disso?

Nosso arquivo `estilos.less` feriu a sintaxe do `less` , pois está faltando os dois pontos. Erros de sintaxe no módulo `less` são erros de sistema e um erro não capturado de um stream derruba todos os outros! Não faz sentido termos nosso sistema desse jeito, não é incomum erramos uma ou outra coisa de uma sintaxe qualquer, assim como o less. Como resolver?

Evitando a explosão!

Podemos definir um tratamento do erro, fazendo com que o módulo `less` esteja de prontidão para lidar com erros. Fazemos isso adicionando um evento `error` :

```
gulp.watch('src/less/**/*.less').on('change', function(event) {
   var stream = gulp.src(event.path)
        .pipe(less().on('error', function(erro) {
          console.log('LESS, erro compilação: ' + erro.filename);
          console.log(erro.message);
        }))
        .pipe(gulp.dest('src/css'));
});
```

É através da função `on` que indicamos em seu primeiro parâmetro que estamos interessado em um evento do tipo `error` . O segundo parâmetro é uma função de callback que recebe como parâmetro um objeto com uma série de informações sobre o erro gerado. Duas informações são de nosso interesse: `filename` , para sabermos qual arquivo está com erro de sintaxe e `message` para sabermos onde o arquivo feriu a sintaxe less.

Vamos rodar novamente nossa tarefa `server`:

```
npm run gulp server
```

Agora, vamos salvar o arquivo novamente, mesmo sem qualquer alteração para verificar se o nosso servidor está de pé:

```
LESS, erro compilação: /Users/flavioalmeida/Desktop/projeto/src/less/estilos.less
directive options not recognised in file /Users/flavioalmeida/Desktop/projeto/src/less/estilos.
```

Sensacional! Nosso servidor continua de pé, inclusive sabemos agora qual arquivo feriu a sintaxe, a descrição do problem e a linha afetada! Já podemos corrigir a declaração de variável:

estilos.less

```
@cor-especial: red;
```

Assim que salvarmos, o arquivo `estilos.css` será criado e nosso navegador recarregará considerando o novo arquivo gerado.

Há IDE's e editores de texto que fazem compilação automática de LESS, porém com o Gulp, temos um ambiente independente de IDE ou editor, inclusive podemos adicionar outros módulos durante o processo de compilação se assim desejarmos.