02

Lidando com pré-processadores

Lidando com pré-processadores

Em algum momento em sua carreira de desenvolvedor você já deve ter ouvido falar de CoffeScript (http://coffeescript.org/), uma linguagem de sintaxe própria que no final é compilada para JavaScript. Tem que ser assim, já que o navegador entende apenas JavaScript. Vejamos um exemplo bem simples com CoffeeScript que declara uma variável.

```
// teste.coffee
nome = "Flávio Almeida"
```

O arquivo a seguir é o resultado da compilação de `teste.coffee` utilizando seu compilador de CoffeeScript preferido:

```
// teste.js
(function() {
  var nome;

  nome = "Flávio Almeida";

}).call(this);
```

O mundo dos pré-processadores não tem como centro apenas o JavaScript, há aqueles dedicados ao CSS como LESS (http://lesscss.org/) entre outros. No caso do LESS, podemos trabalhar com variáveis:

```
/* teste.less */
@color : red;

p {
    color: @color;
}

h1 {
    color: @color;
}
```

Podemos compilar o arquivo `teste.less` através do compilador `lessc` em no terminal que resultará no arquivo `teste.css` :

```
/* teste.css */
p {
  color: #ff0000;
}
h1 {
  color: #ff0000;
}
```

Os recursos e ganhos desses pré-processadores vão muito além desses exemplos, mas não podemos esquecer que independente do pré-processador utilizado, precisamos sempre compilar nossos arquivos toda vez que forem alterados.

Grunt e tasks de pré-processamento

O Grunt possui o plugin grunt-contrib-coffee (https://github.com/gruntjs/grunt-contrib-coffee) para compilar CoffeeScript, inclusive o grunt-contrib-less (https://github.com/gruntjs/grunt-contrib-less) para compilar LESS. O problema é que eles devem ser executados quando novos arquivos forem criados e quando os já existentes forem modificados. Para resolver este último problema, o Grunt possui o plugin grunt-contrib-watch (https://github.com/gruntjs/grunt-contrib-watch).

Primeiro, vamos executar o corriqueiro comando `npm` para instalar os dois pré-processadores que utilizaremos:

```
npm install grunt-contrib-coffee grunt-contrib-less  --save-dev
```

Inclusive registrá-los em nosso Gruntfile.js:

```
grunt.loadNpmTasks('grunt-contrib-coffee');
grunt.loadNpmTasks('grunt-contrib-less');
```

Agora só nos resta configurar as tasks.

As tasks coffee e less

A configuração das taks `coffee` e `less` são praticamente idênticas. Para cada uma delas, adicionaremos o target 'compilar':

```
coffee: {
   compilar: {
      expand: true,
      cwd: 'public/coffee',
      src: ['**/*.coffee'],
      dest: 'public/js'
   }
} ,

less: {
   compilar: {
      expand: true,
      cwd: 'public/less',
      src: ['**/*.less'],
      dest: 'public/css'
   }
}
```

Repare no código anterior que os arquivos `.coffee` e `.less` ficarão em suas respectivas pastas. A ideia é que cada arquivo arquivo `.coffee` e `.less` compilado vá para a pasta `js` e `css` respectivamente.

O problema é que no processo de compilação não muda a extensão do arquivo. Para isso, adicionamos a propriedade ext na configuração do target. Ela mudará a extensão do arquivo copiado para o valor que definimos na propriedade:

```
coffee: {
   compilar: {
      expand: true,
      cwd: 'public/coffee',
      src: ['**/*.coffee'],
      dest: 'public/js',
      ext: '.js'
   }
} ,

less: {
   compilar: {
      expand: true,
      cwd: 'public/less',
      src: ['**/*.less'],
      dest: 'public/css',
      ext: '.css'
   }
}
```

Que tal testarmos? Você pode criar as pastas 'public/coffee' e 'public/less' salvando os arquivos 'teste.coffee' e 'teste.less' respectivamente com os exemplos que vimos no início do capítulo:

```
nome = 'Flávio Almeida'
```

```
/* public/less/teste.less */
@color : red;

p {
    color: @color;
}

h1 {
    color: @color;
}
```

Podemos testar o resultado executando as duas tasks no terminal:

```
grunt coffee less
```

Se tudo correu bem, foram gerados os arquivos `public/js/teste.js` e `public/css/teste.css` . É importante lembrar que são esses arquivos que devem ser importados em sua página, frutos de pré-processamento.

Espere um minuto? Estamos gravando os arquivos na pasta de original! Sim, é verdade. Queremos os arquivos compilados ainda no projeto original, para que possamos já ver os resultados.

Apesar de tudo funcionar, não queremos rodar estas tasks manualmente toda vez que um arquivo mudar, não? É por isso que as executaremos automaticamente toda vez que novos arquivos forem criados ou modificados através da task `watch`.

Automatizando com a task watch

Para termos a tarefa `watch` disponível, precisamos instalar o plugin `grunt-contrib-watch` através do npm:

```
npm install grunt-contrib-watch --save-dev
```

E também registrá-la em nosso `Gruntfile.js`:

```
grunt.loadNpmTasks('grunt-contrib-watch');
```

A configuração da task é feita da seguinte forma: para a task `watch` criaremos dois targets. Um responsável em observar arquivos `.coffee` e outro arquivos `.less`:

```
watch: {

   coffee: {
      files: 'public/coffee/**/*.coffee',
      tasks: 'coffee:compilar'
   },

   less: {
       files: 'public/less/**/*.less',
       tasks: 'less:compilar'
   }
}
```

Toda vez que um arquivo `.coffee` ou `.less` for incluído (added), alterado (changed) ou deletado (deleted) as tasks `watch:coffee` e `watch:less` chamarão as tasks definidas na propriedade `task`. Essas tasks são as que compilarão nossos arquivos.

Repare que elas compilarão todos os arquivos das pastas `coffee` e `less` respectivamente. Isso não é algo ruim, já que arquivos `.less` podem depender de outros, a mesma coisa com nossos arquivos `.coffee`.

Está quase pronto! Quase? Sim, porque não nos interessa executar as tarefas para arquivos deletados. Podemos sair do padrão 'all' e indicar que as tarefas devem escutar apenas os eventos 'added' e 'changed' deixando de fora o evento 'deleted'. Fazemos adicionando a propriedade **options**:

```
watch: {

   coffee: {
      options: {
           event: ['added', 'changed']
       },
      files: 'public/coffee/**/*.coffee',
      tasks: 'coffee:compilar'
```

```
    },

    less: {
        options: {
           event: ['added', 'changed']
        },
        files: 'public/less/**/*.less',
        tasks: 'less:compilar'
    }
}
```

Pronto, e agora? Como rodar as tasks? Diferente do que estamos acostumados, rodarmos a task `watch` diretamente sem qualificar seus targets.

```
$ grunt watch
Running "watch" task
Waiting...
```

Outro ponto importante é que seu console ficará travado. Isto mesmo! A task `watch` rodará infinitamente, algo necessário para que ela monitore os arquivos modificados por você no sistema de arquivos. Caso seja necessário executar outra tarefa do Grunt, você precisará abrir outro terminal.

Que tal testarmos algumas modificações? Abra seu editor de texto e apenas salve os arquivos `teste.coffee` e `teste.less` , mesmo que você não altere nada. A task watch ficará sabendo desta alteração e exibirá uma mensagem no console:

```
Waiting...
>> File "public/less/teste.less" changed.
Running "less:compilar" (less) task
File public/css/teste.css created: 0 B ? 66 B

Done, without errors.
Completed in 0.887s at Mon Jun 09 2014 17:19:47 GMT-0300 (BRT) - Waiting...
>> File "public/coffee/teste.coffee" changed.
Running "coffee:compilar" (coffee) task

Done, without errors.
Completed in 0.903s at Mon Jun 09 2014 17:19:53 GMT-0300 (BRT) - Waiting...
```

Nosso script com todas as nossas tasks ficou assim:

```
module.exports = function(grunt) {

   grunt.initConfig({
      /* Copia os arquivos para o diretório 'dist' */
      copy: {
         public: {
           expand: true,
           cwd: 'public',
           src: '**',
           dest: 'dist'
         }
```

```
 },

 clean: {
      dist: {
          src: 'dist'
      }
 },

 useminPrepare: {
   html: 'dist/**/*.html'
 },

 usemin: {
   html: 'dist/**/*.html'
 },

 imagemin: {
  public: {
    expand: true,
    cwd: 'dist/img',
    src: '**/*.{png,jpg,gif}',
    dest: 'dist/img'
  }
},

rev: {
  options: {
    encoding: 'utf8',
    algorithm: 'md5',
    length: 8
  },

  imagens: {
    src: ['dist/img/**/*.{png,jpg,gif}']
  },
  minificados: {
    src: ['dist/js/**/*.min.js', 'dist/css/**/*.min.css']
  }
},

coffee: {
  compilar: {
    expand: true,
    cwd: 'public/coffee',
    src: ['**/*.coffee'],
    dest: 'public/js',
    ext: '.js'
  }
},

less: {
  compilar: {
    expand: true,
    cwd: 'public/less',
    src: ['**/*.less'],
    dest: 'public/css',
    ext: '.css'
  }
```

```
    },

    watch: {
      coffee: {
        options: {
          event: ['added', 'changed']
        },
        files: 'public/coffee/**/*.coffee',
        tasks: 'coffee:compilar'
      },

      less: {
        options: {
          event: ['added', 'changed']
        },
        files: 'public/less/**/*.less',
        tasks: 'less:compilar'
      }
    }

  });

  //registrando task para minificação

  grunt.registerTask('dist', ['clean', 'copy']);

  grunt.registerTask('minifica', ['useminPrepare',
                                  'concat', 'uglify', 'cssmin', 'rev:imagens','rev:minificados',

  // registrando tasks
  grunt.registerTask('default', ['dist', 'minifica', ]);

  // carregando tasks
  grunt.loadNpmTasks('grunt-contrib-copy');
  grunt.loadNpmTasks('grunt-contrib-clean');
  grunt.loadNpmTasks('grunt-contrib-concat');
  grunt.loadNpmTasks('grunt-contrib-uglify');
  grunt.loadNpmTasks('grunt-contrib-cssmin');
  grunt.loadNpmTasks('grunt-usemin');
  grunt.loadNpmTasks('grunt-contrib-imagemin');
  grunt.loadNpmTasks('grunt-rev');
  grunt.loadNpmTasks('grunt-contrib-coffee');
  grunt.loadNpmTasks('grunt-contrib-less');
  grunt.loadNpmTasks('grunt-contrib-watch');
}
```