02

Realtime linting

Quantas vezes você já escreveu um código JavaScript e para saber se havia algum problema na sintaxe teve que abrir uma página que fizesse uso do script para depois procurar mensagens de erro em seu console? Talvez sua resposta seja "umas zilhões de vezes".

Outra alternativa é você copiar e colar seu código em sites como (http://www.jshint.com/)http://www.jshint.com/ (http://www.jshint.com/). O site analisa seu código indicando onde você feriu a sintaxe da linguagem e ainda dá dicas de como melhor organizá-lo! Esse processo é chamado de **linting**.

Se você procura no console mensagens de erro ou usa sites que analisam seu código como o que eu acabei de citar você ainda estará realizando um processo manual. Abrir navegador, abrir console ou abrir a página X.

Imagina se pudéssemos saber se nosso script segue as normas da linguagem e ainda receber sugestões de melhoria à cada alteração que fizermos? Já sabemos disparar tasks toda vez que um arquivo é modificado através da task *watch*.

Será que não podemos usar algum plugin que execute a verificação de nossos scripts todas as vezes que ele for alterado? *Yes, we can*! É isso que aprenderemos neste capítulo!

O plugin de linting

Utilizaremos o plugin grunt-contrib-jshint (https://github.com/gruntjs/grunt-contrib-jshint) para realizar o *linting* de nossos arquivos modificados.

Baixando a task:

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

Registrando a task:

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

O plugin disponibiliza a task `jshint` . Vamos configurá-la:

```
jshint: {
   js: {
      src: ['public/js/**/*.js']
    }
}
```

Já podemos testar na linha de comando:

```
$ grunt jshint
Running "jshint:js" (jshint) task

   public/js/index.js
```

```
     14 |    }
              ^ Missing semicolon.

>> 1 error in 2 files
Warning: Task "jshint:js" failed. Use --force to continue.
```

Espere um pouco? Como pode haver um erro em nosso código se ele funciona? A omissão do ponto e vírgula após a *function declaration* foi corrigida pelo ASI (automatic semicolon insertion) do JavaScript, mas nem sempre ele é esperto o suficiente. O Jshint não perdoa e pede para que o desenvolvedor explicitamente adicione o ponto e vírgula.

Faça um teste: coloque o ponto e vírgula e rode a task novamente:

```
$ grunt jshint
Running "jshint:js" (jshint) task
>> 2 files lint free.

Done, without errors.
```

Que tal agora rodarmos nossa task quando o arquivo for alterado? Vamos adicionar mais uma subtask para task `watch` :

```
watch: {
  /*  subtask anteriores */

   js: {
      options: {
         event: ['changed']
      },
      files: 'public/js/**/*.js',
      tasks: 'jshint:js'
   }

}
```

Agora podemos rodar a task `watch` que rodará infinitamente no terminal e logo em seguida consertar nosso arquivo. Nosso arquivo deverá passar no teste:

```
$ grunt watch
Running "watch" task
Waiting...
>> File "public/js/index.js" changed.
Running "jshint:js" (jshint) task
>> 2 files lint free.

Done, without errors.
```

Repare que ele não analisa apenas o arquivo que modificamos, mas todos os arquivos da pasta, o que não é ruim, uma vez que outros arquivos podem ter sido adicionados no projeto.