JSHint

Victor Herrero Cazurro

Contenidos

1. Introduccion

Herramienta de calidad estatica, que inspecciona malas practicas en la codificación.

2. Instalacion

Se recomienda inicializar un proyecto **Node.js** lanzando el comando

```
> npm init
```

Obteniendo así el fichero **package.json**

Se ha de lazar el siguiente comando para crear un proyecto **JSHint**

```
> npm install jshint
```

Para ejecutar la herramienta, se dispone del comando

```
> node node_modules/jshint/bin/jshint --reporter=checkstyle --verbose
src/helloworld.js
```

Donde los parametros ha definir según la documentación que se puede consultar aquí son

- --verbose: Muestra los codigos de error de los mensajes
- --config: Permite indicar un fichero de configuracion de reglas especifico
- --reporter: Permite indicar la forma de generacion del reporte. Puede ser: jslint, checkstyle, unix.
- --exclude: Permite indicar patrones de ficheros a excluir del analisis, equivalente al uso de fichero **.jshintignore**
- --exclude-path: Permite indicar el path del fichero **.jshintignore**
- --show-non-errors: Permite mostrar inforacion extra generada por jshint
- -e, --extra-ext: Permite definir una lista separadas por comas de extensiones a analizar, por defecto es **.js**

2.1. Activacion/Desactivacion de Reglas

Se puede definir una configuracion a aplicar con JSHint, definiendo:

- Fichero con extension **.jshintrc**
- Con la propiedad **jshintConfig** en el fichero **package.json**.
- Con comentarios dentro de los ficheros a analizar.

2.2. .jshintrc

Ejemplo de fichero **.jshintrc**

```
{
    // JSHint Default Configuration File (as on JSHint website)
    // See http://jshint.com/docs/ for more details

    "maxerr"        : 50,       // {int} Maximum error before stopping

    // Enforcing
    "bitwise"       : true,     // true: Prohibit bitwise operators (&,
|, ^, etc.)
    "camelcase"     : false,    // true: Identifiers must be in camelC
ase
    "curly"         : true,     // true: Require {} for every new block
or scope
    "eqeqeq"        : true,     // true: Require triple equals (===) f
or comparison
    "forin"         : true,     // true: Require filtering for..in loo
ps with obj.hasOwnProperty()
    "freeze"        : true,     // true: prohibits overwriting prototy
pes of native objects such as Array, Date etc.
    "immed"         : false,    // true: Require immediate invocations
to be wrapped in parens e.g. `(function () { } ());`
    "latedef"       : false,    // true: Require variables/functions to
be defined before being used
    "newcap"        : false,    // true: Require capitalization of all
constructor functions e.g. `new F()`
    "noarg"         : true,     // true: Prohibit use of `arguments.ca
ller` and `arguments.callee`
    "noempty"       : true,     // true: Prohibit use of empty blocks
    "nonbsp"        : true,     // true: Prohibit "non-breaking
whitespace" characters.
    "nonew"         : false,    // true: Prohibit use of constructors
```

```
for side-effects (without assignment)
    "plusplus"      : false,    // true: Prohibit use of `++` and `--`
    "quotmark"      : false,    // Quotation mark consistency:
                                //   false    : do nothing (default)
                                //   true     : ensure whatever is used
is consistent
                                //   "single" : require single quotes
                                //   "double" : require double quotes
    "undef"         : true,     // true: Require all non-global variab
les to be declared (prevents global leaks)
    "unused"        : true,     // Unused variables:
                                //   true     : all variables, last fu
nction parameter
                                //   "vars"   : all variables only
                                //   "strict" : all variables, all fun
ction parameters
    "strict"        : true,     // true: Requires all functions run in
ES5 Strict Mode
    "maxparams"     : false,    // {int} Max number of formal params a
llowed per function
    "maxdepth"      : false,    // {int} Max depth of nested blocks (w
ithin functions)
    "maxstatements" : false,    // {int} Max number statements per fun
ction
    "maxcomplexity" : false,    // {int} Max cyclomatic complexity per
function
    "maxlen"        : false,    // {int} Max number of characters per
line
    "varstmt"       : false,    // true: Disallow any var statements.
Only `let` and `const` are allowed.

    // Relaxing
    "asi"           : false,     // true: Tolerate Automatic Semicolon
Insertion (no semicolons)
    "boss"          : false,     // true: Tolerate assignments where c
omparisons would be expected
    "debug"         : false,     // true: Allow debugger statements e.
g. browser breakpoints.
    "eqnull"        : false,     // true: Tolerate use of `== null`
    "esversion"     : 5,         // {int} Specify the ECMAScript versi
on to which the code must adhere.
    "moz"           : false,     // true: Allow Mozilla specific syntax
(extends and overrides esnext features)
                                 // (ex: `for each`, multiple try/catc
h, function expression…)
```

```
    "evil"          : false,     // true: Tolerate use of `eval` and `
new Function()`
    "expr"          : false,     // true: Tolerate `ExpressionStatemen
t` as Programs
    "funcscope"     : false,     // true: Tolerate defining variables
inside control statements
    "globalstrict"  : false,     // true: Allow global "use strict" (a
lso enables 'strict')
    "iterator"      : false,     // true: Tolerate using the `__iterat
or__` property
    "lastsemic"     : false,     // true: Tolerate omitting a semicolon
for the last statement of a 1-line block
    "laxbreak"      : false,     // true: Tolerate possibly unsafe line
breakings
    "laxcomma"      : false,     // true: Tolerate comma-first style c
oding
    "loopfunc"      : false,     // true: Tolerate functions being def
ined in loops
    "multistr"      : false,     // true: Tolerate multi-line strings
    "noyield"       : false,     // true: Tolerate generator functions
with no yield statement in them.
    "notypeof"      : false,     // true: Tolerate invalid typeof oper
ator values
    "proto"         : false,     // true: Tolerate using the `__proto_
_` property
    "scripturl"     : false,     // true: Tolerate script-targeted URLs
    "shadow"        : false,     // true: Allows re-define variables l
ater in code e.g. `var x=1; x=2;`
    "sub"           : false,     // true: Tolerate using `[]` notation
when it can still be expressed in dot notation
    "supernew"      : false,     // true: Tolerate `new function () {
... };` and `new Object;`
    "validthis"     : false,     // true: Tolerate using this in a non-
constructor function

    // Environments
    "browser"       : true,     // Web Browser (window, document, etc)
    "browserify"    : false,    // Browserify (node.js code in the bro
wser)
    "couch"         : false,    // CouchDB
    "devel"         : true,     // Development/debugging (alert, confi
rm, etc)
    "dojo"          : false,    // Dojo Toolkit
    "jasmine"       : false,    // Jasmine
    "jquery"        : false,    // jQuery
```

```
    "mocha"          : true,     // Mocha
    "mootools"       : false,    // MooTools
    "node"           : false,    // Node.js
    "nonstandard"    : false,    // Widely adopted globals (escape, une
scape, etc)
    "phantom"        : false,    // PhantomJS
    "prototypejs"    : false,    // Prototype and Scriptaculous
    "qunit"          : false,    // QUnit
    "rhino"          : false,    // Rhino
    "shelljs"        : false,    // ShellJS
    "typed"          : false,    // Globals for typed array constructio
ns
    "worker"         : false,    // Web Workers
    "wsh"            : false,    // Windows Scripting Host
    "yui"            : false,    // Yahoo User Interface

    // Custom Globals
    "globals"        : {}        // additional predefined global variab
les
}
```

2.2.1. Metainformacion

Se pueden ignorar/aplicar reglas a ficheros con la sintaxis

```
/* jshint undef: true, unused: true */
/* jshint -W033 */
```

Donde **undef** permite validar variables no definidas y **unused** variables no empleadas.

Donde **W033** es el codigo de error de JSHint y **-** indica que no se aplique, si se desea aplicar, se pondria **+**.

En este caso, el siguiente codigo

```
/* jshint undef: true, unused: true */
/* jshint +W033 */
var miFuncion = function(param){
    x = 0
}
```

Daria un informe

```
> node node_modules/jshint/bin/jshint --reporter=checkstyle
src/helloworld.js
<?xml version="1.0" encoding="utf-8"?>
<checkstyle version="4.3">
        <file name="src/helloworld.js">
                <error line="3" column="26" severity="warning"
message="&apos;param&apos; is defined but never used."
source="jshint.W098" />
                <error line="5" column="2" severity="warning"
message="Missing semicolon." source="jshint.W033" />
                <error line="4" column="5" severity="warning"
message="&apos;x&apos; is not defined." source="jshint.W117" />
                <error line="3" column="5" severity="warning"
message="&apos;miFuncion&apos; is defined but never used."
source="jshint.W098" />
        </file>
</checkstyle>
```

2.3. Plugins de editores

Se dispone de plugins para los principales editores, que son capaces de leer la configuración del modulo **Node.js** y según se escribe el codigo avisar de los errores de estilo de codificación que se van introduciendo.