ESLint

Tiempo estimado: 15min

Introducción

Una herramienta de análisis de código estático (*static code analysis*) que lee código, sin llegar a ejecutarlo, con objeto de detectar errores y patrones poco recomendables. Lo que se busca es que el código cumpla las reglas sintácticas de JavaScript, además de los estándares o convenciones definidos por la organización. Por ejemplo, algunas organizaciones no desean que las expresiones condicionales de las sentencias ifs dispongan de asignaciones, sólo está permitida la comparación, que en el código no aparezca el objeto console y cosas así. Son las herramientas de análisis de código las encargadas de detectar si alguna de las reglas definidas por la organización se está violando.

Formalmente, este tipo de herramientas se conocen como *linters*. Y en JavaScript, podemos encontrar varias como, por ejemplo, JSHint, JSLint y ESLint. En nuestro caso, nos vamos a centrar en ESLint.

ESLint

ESLint es una herramienta de análisis estático de código JavaScript. Desarrollada en JavaScript y Node en 2013 por Nicholas C. Zaker, se utiliza ampliamente hoy en día. Encontrando entre otras organizaciones usuarias a Airbnb, Apache Software Foundation, Atlassian, Box, Facebook, Google, IBM, jQuery Foundation, Microsoft, MongoDB, Mozilla Foundation, Node.js Foundation y PayPal.

Sus principales características son:

- Es fácil de aprender.
- Es fácil de usar.
- Analiza código JavaScript en varias especificaciones: ES5, ES6 e incluso ES7.
- Utiliza el concepto de regla para definir lo que debe cumplirse en el código.
- Viene con un conjunto bastante amplio de reglas predefinidas.
- Es muy fácil añadir nuevas reglas mediante *plugins*.
- Analiza código JSX, utilizado por el *framework* React de Facebook.

Su sitio web oficial es eslint.org.

Arquitectura de ESLint

ESLint tiene una arquitectura muy sencilla. Por un lado, tenemos el *linter* o herramienta que analiza el código JavaScript en busca de reglas que no se cumplen. Podemos encontrarlo en forma de aplicación de línea de comandos o bien de *plugin* para algunas herramientas de automatización como Grunt, Gulp o Justo. Por otro lado, tenemos las reglas, quienes representan cosas que deben cumplirse como, por ejemplo, la prohibición de usar el objeto console, la prohibición de utilizar variables no definidas previamente, la obligación de indicar la directiva de modo estricto, etc.

Cada vez que deseamos comprobar que las reglas se cumplen, no hay más que ejecutar el analizador, pasándole los archivos que debe procesar y analizar.

Instalación de ESLint

Lo primero es instalar el analizador. Los prerrequisitos son la plataforma Node y el administrador de paquetes NPM. Cumplidos estos requisitos, sólo nos queda instalarlo. Podemos hacerlo local o globalmente.

Instalación global

La instalación global consiste en instalarlo para toda la máquina. Se utiliza el comando npm y se puede hacer como sigue:

```
npm install -g eslint
```

Una vez instalado, se recomienda comprobar que se tiene acceso al comando eslint, por ejemplo, consultando su versión o ayuda:

```
eslint -v
eslint -h
```

Instalación local

También es posible instalar ESLint a nivel de proyecto. No hay más que añadir el paquete eslint en la lista de dependencias de desarrollo, esto es, en la propiedad devDependencies del archivo package.json:

```
"devDependencies": {
  "eslint": "*"
}
```

En este caso, el comando eslint se encontrará en ./node_modules/.bin/eslint.

Si usamos algún tipo de herramienta de automatización como Grunt, Gulp o Justo, bastará con añadir a las dependencias de desarrollo el *plugin* correspondiente. Así, por ejemplo, en el caso de Justo.js, tendremos:

```
"devDependencies": {
  "justo-plugin-eslint": "*"
}
```

Archivo de configuración

Para cada proyecto, hay que definir el archivo de configuración .eslintrc, en el cual definir las reglas que usaremos en el proyecto. También es posible hacerlo en el archivo package.json bajo la propiedad eslintConfig. Por buenas prácticas, se prefiere el archivo .eslintrc.

Según el formato de archivo .eslintrc, se utiliza una extensión u otra:

- .js cuando el archivo está en formato JavaScript.
- .json o ninguna extensión particular cuando está en formato JSON.
- .yml o .yaml en YAML.

En nuestro caso, usaremos .eslintrc, sin extensión, un archivo de texto en formato JSON que contiene la configuración ESLint utilizada en el proyecto. Contiene un objeto JSON, con varias propiedades:

- parserOptions. Contiene configuración relacionada con el funcionamiento del analizador como, por ejemplo, la especificación JavaScript bajo la cual se debe encontrar el código o bien el analizador a usar.
- extends. Configura conjuntos de reglas comunmente aceptadas.
- rules. Contiene las reglas a utilizar en el proyecto.
- env. Contiene información sobre el entorno en el que se ejecutará la aplicación o componente JavaScript. Por ejemplo, el navegador, la plataforma Node o el intérprete de MongoDB.
- plugins. Indica los plugins a utilizar.
- globals. Define variables globales para las que el analizador no debe buscar su sentencia de definición let, var o const.

A continuación, se presenta la propiedad parserOptions, presentándose las demás detalladamente en secciones particulares.

ParserOptions

La propiedad parserOptions contiene configuración específica del analizador. Veamos sus propiedades más utilizadas:

- ecmaVersion (number). Indica la especificación bajo la cual debe encontrarse el código JavaScript: 3, 5, 6 ó 7.

  Valor predeterminado: 5.
- sourceType (string). Tipo de proyecto JavaScript: script o module.

  Valor predeterminado: script.
- ecmaFeatures (object). Características de JavaScript adicionales a tener en cuenta:
  - jsx (boolean). Permite código JSX.
  - impliedStrict (boolean). Activa el modo estricto.
  - globalReturn (boolean). Permite sentencias return a nivel global.
  - experimentalObjectRestSpread (boolean). Permite el uso del parámetro de resto y el operador *spread*.

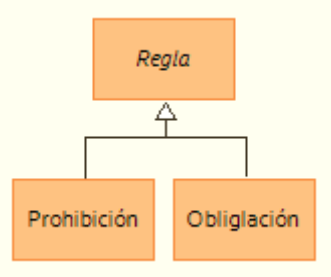
Creación del archivo de configuración

Para facilitar la creación del archivo de configuración, podemos usar la opción --init de eslint:

```
eslint --init
```

Reglas

Una regla (*rule*) representa algo que debe cumplirse. Básicamente, se distingue dos conceptos: las prohibiciones y las obligaciones.



Una prohibición (*prohibition*) impide el uso de algo como, por ejemplo, el objeto console o el operador de asignación en las condiciones asociadas a ifs. Mientras que una obligación (*obligation*) lo que hace es forzar o imponer algo como, por ejemplo, indicar la directiva de modo estricto en todos los archivos de código.

Las reglas se definen mediante la propiedad rules del archivo de configuración. Cada una de ellas, se define como una propiedad, cuyo nombre es la regla, y un valor, que indica cómo debe procesar la regla el analizador:

```
"regla": nivel
"regla": [nivel, opciones]
```

El nivel de la regla (*rule level*) indica, por un lado, si la regla está activa y, por otro lado, si lo está, si debe procesarse como aviso o error. Sus posibles valores son:

- off. Regla desactivada.
- warn. Regla activada y, cuando no se cumpla, debe procesarse como aviso.
- error. Regla activada y, cuando no se cumpla, debe procesarse como error.

La diferencia entre warn y error es muy sencilla. Cuando el analizador se encuentra con una regla configurada como aviso, lo que hace es mostrar que no se cumple y nada más. No afecta al resultado final del análisis. En cambio, una configurada como error, además de mostrar que no se cumple, afecta al resultado final del análisis.

eslint muestra las reglas que no se cumplen y además devuelve un código de estado que puede ser cero o uno. Los errores hacen que el código de estado devuelto sea 1. En cambio, los avisos no lo modifican. Se considera que el código pasa todas las reglas, si devuelve cero, esto es, si ninguna regla configurada como error es violada.

Veamos un pequeño ejemplo ilustrativo:

```
"rules": {
  "eqeqeq": "off",
  "no-cond-assign": "error",
  "no-empty": "warn"
}
```

La lista de reglas predefinidas se encuentra en eslint.org/docs/rules. He aquí las más utilizadas:

Regla	Tipo	Descripción
no-cond-assign	Prohibición	No se puede usar el operador de asignación en las condiciones de las sentencias if, while, for y do..while.
no-console	Prohibición	No se puede usar el objeto console.
no-constant-condition	Prohibición	No se permite que la condición asociada a un if, while, for o do..while sea el literal true o false.
no-debugger	Prohibición	No se puede usar la sentencia debugger.
no-dupe-args	Prohibición	No se permite que dos o más parámetros de una función tengan el mismo nombre.
no-dupe-keys	Prohibición	No se permite que dos o más propiedades en un literal de tipo objeto tengan el mismo nombre.
no-empty	Prohibición	No se permite que un bloque o cuerpo de una sentencia if, while, for o do..while no tenga ninguna proposición.
no-ex-assign	Prohibición	No se puede modificar el valor de una variable excepción definida en una cláusula catch.
no-extra-parens	Prohibición	No se permite definir paréntesis innecesarios.
no-func-assign	Prohibición	No se puede asignar un nuevo valor a una variable cuyo valor contiene una función definida mediante la sentencia function.
no-inner-declarations	Prohibición	No se puede definir funciones dentro de otras funciones, ni variables con var dentro de sentencia de control de flujo como if, while, for y do..while.
no-invalid-regexp	Prohibición	No se puede definir expresiones regulares inválidas.
valid-typeof	Obligación	Cuando se compare el valor devuelto por el operador typeof, deberá hacerse sólo con un valor posible.
curly	Obligación	Cuando se defina el cuerpo de una sentencia de control de flujo, debe hacerse siempre mediante { y }.
default-case	Obligación	Toda sentencia switch debe definir una cláusula default.
no-redeclare	Prohibición	No se puede redefinir variables en el mismo bloque.
strict	Obligación	Es necesaria la directiva de modo estricto.

De manera predeterminada, ninguna regla se encuentra configurada. Como conocer y configurar todas las reglas es una empresa titánica, ESLint permite configurar un bloque muy comunmente utilizado mediante la propiedad extends de la raíz del archivo de configuración:

```
"extends": "eslint:recommended"
```

Entornos

El entorno (*environment*) indica dónde se ejecutará el *script* o componente, por ejemplo, en la plataforma Node o en un navegador. Se especifica mediante la propiedad env del archivo de configuración, que consiste en un objeto cuyas propiedades indican los entornos para los cuales se ha diseñado el software. Los más comunes son:

- browser (boolean). Para navegador.
- node (boolean). Para la plataforma Node.
- shared-node-browser (boolean). Tanto para navegador como Node.

- mongo (boolean). Para el comando mongo.
- jquery (boolean). Como aplicación jQuery.
- meteor (boolean). Como aplicación Meteor.

Cuando se define un determinado entorno, automáticamente se permite el uso de aquellos objetos globales propios. Por ejemplo, en el navegador, objetos como window y XMLHttpRequest.

Ejemplo ilustrativo:

```
"env": {
  "browser": true,
  "node": true
}
```

Para conocer la lista completa de entornos, eslint.org/docs/user-guide/configuring#specifying-environments.

Variables globales

Una variable global (*global variable*) es aquella accesible a lo largo de todo el programa. Si se desea, se puede configurar una regla que obligue a que todo contenedor de datos se haya definido previamente a su uso mediante una sentencia var, let, const, function, class, etc. Pero en ocasiones, algunas plataformas llevan consigo determinados objetos globales, no definidos en el programa. Así, por ejemplo, el navegador tiene objetos como console, window o XMLHttpRequest.

Acabamos de ver hace unos instantes, que la especificación de un determinado entorno lleva impícito la posibilidad de usar, en nuestro código, aquellas variables globales que son específicas de él. Pero ¿qué ocurre si deseamos poder utilizar otras variables globales no definidas por el entorno ni en el propio código? Pues que tendremos que dárselas a conocer a ESLint. Para este fin, se encuentra la propiedad globals del archivo de configuración. Consiste en un objeto donde cada propiedad es el nombre de una variable global y cuyo valor indica si se puede usar: true, sí; false, no.

He aquí un ejemplo:

```
"globals": {
  "fs": true,
  "console": false
}
```

Plugins

Una de las características más importantes de ESLint es su extensibilidad. Se puede definir plugins, componentes que definen nuevas reglas. Por ejemplo, de manera predeterminada, ESLint para analiza código JSX, una sintaxis de JavaScript utilizada en el *framework* React de Facebook. Ahora bien, si añadimos el *plugin*, el analizador analizará también código JSX.

La propiedad plugins del archivo de configuración consiste en un *array* que contiene el nombre de los *plugins* a utilizar:

```
"plugins": [
  "eslint-plugin-uno",
  "eslint-plugin-dos"
]
```

Por convenio, los *plugins* se definen en paquetes NPM cuyo nombre sigue la siguiente sintaxis: eslint-plugin-nombre. Por ejemplo, el *plugin* para React es eslint-plugin-react.

Para utilizar un *plugin*, lo mejor es leer su documentación detenidamente.

Instalación de plugins

Para poder utilizar un *plugin* de ESLint, hay que importarlo. Si estamos usando ESLint mediante una instalación global, hay que instalar los *plugins* también globalmente. En cambio, si lo estamos usando localmente al proyecto, por lo general, se indica en la propiedad devDependencies del archivo de configuración package.json.

Comando eslint

Una vez tenemos claro cómo instalar y usar ESLint en un proyecto, lo siguiente es usarlo. Recordemos que podemos tenerlo instalado local o globalmente. En esta sección, nos centramos en eslint, ya sea local o global, no en *plugins* disponibles en herramientas de automatizción como Grunt, Gulp o Justo.

Listado de opciones

Para conocer la lista de opciones del comando eslint, utilizar la opción -h:

```
eslint -h
```

Análisis de archivos

Para analizar uno o más archivos, no hay más que indicarlos al final de la línea de comandos:

```
eslint [opciones] archivos
```

Ejemplo:

```
eslint archivo.js archivo.jsx
```

Se puede indicar tanto rutas de archivos como de directorios. En este último caso, para indicar qué archivos debe analizar, se utiliza la opción --ext, la cual contiene una lista, separada por comas, de las extensiones de archivo a procesar. Así, por ejemplo, para analizar los archivos con extensión .js, la predeterminada, y .jsx, podremos hacerlo como sigue:

```
eslint --ext .js,.jsx directorio
```

Archivo de configuración

Mediante la opción -c se puede indicar un determinado archivo de configuración:

```
eslint -c .eslintrc archivo.js
```

Archivo .eslintignore

En algunas ocasiones, es necesario indicar que no se procesen determinados archivos. Esto se puede conseguir mediante la opción --ignore-path del comando eslint o bien el archivo .eslintignore. Este archivo de texto contiene una línea por cada ruta a ignorar, de manera muy similar a .gitignore.

Lo mejor un ejemplo ilustrativo.

```
#archivos de build/: ignorarlos todos salvo index.js
build/*
!build/index.js

#archivos de node_modules: ignorarlos todos
node_modules/*
```

Información de publicación

Título	ESLint
Autor	Raúl G. González - raulggonzalez@nodemy.com
Primera edición	Agosto de 2016
Última actualización	Octubre de 2016
Versión actual	1.0.1
Contacto	hola@nodemy.com

