Aplicaciones Node

Tiempo estimado: 15min

En la lección anterior, aprendimos a desarrollar paquetes de biblioteca. Aquellos que proporcionan una funcionalidad reutilizable en otros paquetes. En esta lección, describimos cómo desarrollar aplicaciones Node.

Comenzamos introduciendo el concepto de aplicación Node. A continuación, presentamos los comandos binarios, aquellos que representan la aplicación Node que se debe instalar en el directorio predeterminado global. Seguimos con las variable global process, muy utilizada en aplicaciones Node. Finalmente, mostramos el concepto de script npm.

Al finalizar la lección, el estudiante sabrá:

- Qué es una aplicación Node.
- Cómo configurar los comandos a instalar mediante npm install.
- Qué tipo de información contiene la variable global process.
- Cómo acceder a los argumentos pasados a la aplicación en la línea de comandos.
- Cómo registrar líneas de comandos en el archivo package.json para su ejecución mediante el comando npm.

Introducción

Una aplicación (*application*) no es más que un programa diseñado para hacer algo. Y una aplicación Node (*Node application*), aquella que se escribe para la plataforma Node. Más concretamente, un paquete NPM que instala uno o más *scripts* específicos en el directorio bin del directorio predeterminado.

Todo lo que hemos visto hasta ahora sobre los paquetes se aplica o se puede aplicar a una aplicación Node. En este lección, vamos a ver aspectos más propios de las aplicaciones Node que de paquetes reutilizables.

Una de las ventajas de las aplicaciones Node es que se pueden publicar mediante el repositorio NPM, facilitando así su instalación en los clientes con npm.

Comandos binarios

Una aplicación consiste en uno o más comandos que pueden ejecutar directamente los usuarios desde la línea de comandos. Generalmente, una aplicación sólo despliega un único comando. Pero si es necesario, puede desplegar varios.

Una aplicación Node no es más que un paquete. Pero que cuando se instala, despliega varios comandos en el subdirectorio bin del directorio predeterminado global. Se puede indicar a npm qué comandos debe instalar en este subdirectorio, mediante la propiedad bin del archivo package.json. Consiste en un objeto, donde la clave indica el nombre del archivo a crear y el valor, el archivo que debe copiar. Así, por ejemplo, si deseamos que npm install cree el archivos micmd, podríamos tener algo como sigue:

```
"bin": {
  "micmd": "bin/micmd.js"
}
```

Cuando npm instale el paquete, en sistemas Linux creará el archivo micmd.sh, cuyo contenido será el del archivo bin/micmd.js. En sistemas Windows, creará micmd.cmd.

Por convenio y buenas prácticas, se recomienda ubicar estos *scripts* en el directorio bin del proyecto. Ojo, los archivos de este directorio pueden ser de cualquier tipo. Pueden ser *scripts* de Bash, de PowerShell, de JavaScript, ejecutables o cualquier otro.

Variable global process

El motor de Node define automáticamente la variable global process con información sobre el proceso de ejecución actual. En esta variable, encontramos los argumentos pasados a la aplicación en la línea de comandos y otra información útil.

A continuación, mostramos los miembros del objeto process más frecuentemente utilizados. La lista completa se puede consultar en nodejs.org/dist/latest/docs/api/process.htm.

Argumentos pasados al programa

Se puede utilizar la propiedad process.argv para conocer los argumentos pasados a node en la línea de comandos. Consiste en un *array* de cadenas de texto donde cada elemento representa un argumento. El primer argumento, índice 0, es la ruta del intérprete node. El segundo argumento, índice 1, es la ruta al módulo JavaScript ejecutado, o sea, el que contiene el comando. A partir del tercero, se encuentra los argumentos reales de la aplicación.

Salida del programa

Para salir del programa explícitamente, se puede usar la función exit() del objeto process:

```
function exit()
function exit(code)
```

Parámetro	Tipo de datos	Descripción
code	number	Código de salida.

El código de salida (*exit code*) o estado de salida (*exit status*) es un número que devuelve el programa cuando termina, indicando con él cómo ha finalizado. Si todo ha ido bien, se recomienda devolver el valor cero. En cualquier otro caso, un valor mayor que cero.

PID del proceso

El PID es un número proporcionado por el sistema operativo que identifica al proceso actual de manera única en el sistema. Se puede consultar mediante la propiedad process.pid.

Arquitectura de la máquina

Para obtener información sobre la máquina en la que se está ejecutando el paquete, podemos utilizar las propiedades process.arch (string) y process.platform (string). La primera indica la arquitectura hardware, mientras que la segunda el sistema operativo.

Variables de entorno

La propiedad process.env es un objeto que contiene las variables de entorno y sus valores.

Directorio actual

El directorio actual se puede consultar mediante la función process.cwd(). Y se puede cambiar mediante process.chdir(). Veamos sus signaturas:

```
function cwd() : string
function chdir(dir)
```

Parámetro	Tipo de datos	Descripción
dir	string	Directorio al que cambiar.

Información de node

Para consultar la información sobre node, se puede usar dos propiedades: version y versions. La primera es una cadena de texto como v6.9.1 o v7.0.0. La segunda es un objeto que contiene propiedades con información más específica. Ejemplo:

```
$ node -p "process.version"
```



```
v7.0.0
$ node -p "process.versions"
{ http_parser: '2.7.0',
  node: '7.0.0',
  v8: '5.4.500.36',
  uv: '1.9.1',
  zlib: '1.2.8',
  ares: '1.10.1-DEV',
  icu: '57.1',
  modules: '51',
  openssl: '1.0.2j' }
$
```

Scripts

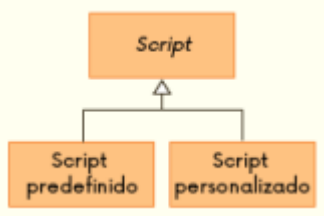
Es posible registrar, en el archivo package.json, líneas de comandos que podemos ejecutar mediante el comando npm. Estas líneas de ejecución se registran en la propiedad scripts. Se trata de un objeto donde cada propiedad representa una línea de comandos, donde la clave indica el nombre que le asociamos y con el que invocarla, mientras el valor representa la línea a ejecutar.

Veamos un ejemplo introductorio, para ir abriendo boca:

```
"scripts": {
  "start": "node ./bin/www.js",
  "start-dev": "./node_modules/.bin/nodemon --delay 2s ./bin/www.js",
  "test": "justo test"
}
```

Estos *scripts* se registran en todo tipo de paquetes. Pero en los que representan aplicaciones servidoras, es muy común registrar líneas de comandos con las que arrancar la aplicación.

Cada una de las líneas se conoce formalmente como *script*, distinguiéndose dos tipos, los predefinidos y los personalizados.



Un script predefinido (*built-in script*) es aquel que tiene un significado especial para npm. Por ejemplo, mediante start hay que registrar la línea de comandos con la que arrancar la aplicación; test se reserva para ejecutar la batería de pruebas de unidad del paquete; etc. Por su parte, un script personalizado (*custom script*) es aquel que no tiene ningún significado especial para npm, o sea, cualquiera que no sea predefinido.

npm permite dividir la lógica de los *scripts* en tres partes: preejecución, ejecución y posejecución. Cuando invocamos un *script*, automáticamente npm ejecuta, primero, su *script* de preejecución, a continuación, el *script* invocado y, finalmente, su *script* de posejecución. Los *scripts* previos y posteriores se registran con el mismo nombre, pero con el prefijo pre y post, respectivamente. Así pues, los *scripts* pre y pos de start son prestart y poststart. Estos *scripts* no tienen por qué existir siempre. Lo que ocurre es que si existen, npm los ejecutará implícitamente.

Por convenio y buenas prácticas, cuando un *script* tiene asociado un *script*, valga la redundancia, definido en el propio proyecto, es decir, es específico del proyecto, se recomienda ubicarlo en la carpeta scripts.

Código de salida

Los *scripts* deben devolver un valor numérico. Cuando devuelven el valor cero, npm considera que han terminado correctamente. En cualquier otro caso, considera que ha finalizado con algún tipo de error, pudiendo no ejecutar los que le sigan. Por ejemplo, si el *script* prestart falla, no ejecutará ni start ni poststart.

Scripts predefinidos

Los *scripts* predefinidos tienen un significado especial en los paquetes Node. npm define los siguientes:

- prepublish se ejecuta antes de publicar el paquete en el repositorio NPM.
- publish y postpublish se ejecutan después de publicar el paquete.
- preinstall se ejecuta antes de instalar el paquete.
- install y postinstall se ejecutan después de instalar el paquete.
- preuninstall y uninstall se ejecutan antes de desinstalar el paquete.
- postuninstall se ejecuta después de desinstalar el paquete.
- prestart se ejecuta antes de invocar el *script* start.
- start se ejecuta cuando se invoca el comando npm start.

  Este comando se recomienda para arrancar la aplicación. Debe indicar la línea de comandos que debe ejecutar npm.
- poststart se ejecuta después de invocar el comando start.
- prestop se ejecuta antes de invocar el *script* stop.
- stop se ejecuta mediante el comando npm stop.

  Se utiliza principalmente para detener la aplicación.
- poststop se ejecuta después de ejecutar el *script* stop.
- prerestart se ejecuta antes del *script* restart.
- restart se ejecuta mediante el comando npm restart.

  Se usa para reiniciar la aplicación.
- postrestart se ejecuta después de ejecutar el *script* restart.
- pretest se ejecuta antes del *script* test.
- test se ejecuta mediante el comando npm test.

  Se reserva para ejecutar la batería de pruebas de unidad del paquete.
- posttest se ejecuta después del *script* test.

Observe que algunos *scripts* se invocan cuando se ejecutan determinados comandos de npm como, por ejemplo, preinstall, install y postinstall que se llaman cuando ejecutamos el comando npm install. Otros requieren que indiquemos el comando que deseamos ejecutar. Por ejemplo, el comando start se asocia a la operación de inicio de la aplicación. Debiendo indicarse en el *script* start cuál es la línea con la que llevar a cabo esta operación. Y test se utiliza para la ejecución de las pruebas de unidad del paquete.

Scripts personalizados

Los *scripts* personalizados son aquellos que no tienen asociados un nombre predefinido, es decir, un nombre con un significado especial para npm. Se pueden ejecutar mediante el comando npm run:

```
npm run nombre-script
```

Por ejemplo, supongamos que tenemos el comando start-dev, que no tiene un nombre predefinido, aunque comience por start. Este *script* se suele usar para arrancar la aplicación en el entorno de desarrollo, mientras que start se reserva para producción. Recordemos que el script start se ejecuta mediante el comando específico npm start. En cambio, el *script* start-dev se debe invocar mediante el comando npm run como sigue:

```
npm run start-dev
```

Si lo deseamos, podemos pasar argumentos posicionales a los *scripts*. Esto se consigue mediante la siguiente sintaxis:

```
npm run nombre-script -- argumento argumento…
```

Observe los dos guiones (--). Son necesarios. Lo que sigue es la lista de argumentos a pasar a los *scripts*.

He aquí un ejemplo ilustrativo:

```
npm run start-dev -- ./logs
```

Generador justo-generator-node

Cuando cree la estructura de directorios del proyecto mediante el generador de Justo, seleccione app cuando le pregunte el tipo de paquete a crear.

