Plugins (práctica)

Tiempo estimado: 25min

El objeto de esta práctica es afianzar, reforzar y consolidar los conocimientos teóricos presentados en la lección.

Al finalizar la práctica, el estudiante:

- Habrá creado un *plugin* simple.
- Habrá creado un *plugin* compuesto.
- Habrá usado el generador justo-generator-plugin para facilitar la creación de *plugins*.

Creación de un plugin simple

El objeto de esta sección es crear un *plugin* simple que consista en una tarea para copiar archivos. Las opciones de copia se indicarán mediante un objeto donde la propiedad src indica el archivo origen y dst, el destino. He aquí un ejemplo de invocación:

```
cp("Backup de Justo.js", {src: "Justo.js", dst: "Justo.js.bak"})
```

Preparación del entorno

Para comenzar, vamos a crear el directorio para el desarrollo de un *plugin* de Justo:

1. Abrir una consola.
2. Instalar globalmente el generador de *plugins*:

   ```
   > npm install -g justo-generator-plugin
   ```
3. Comprobar que lo tenemos instalado:

   ```
   > justo -g plugin help
   ```
4. Crear el directorio del *plugin*, justo-uplugin-cp, y entrar en él.
5. Crear la estructura de directorios del *plugin*:

   ```
   > justo -g plugin
   ```
   Rellenar los campos solicitados. No pasa nada si los deja vacíos o se los inventa, aunque se recomienda que finja que está trabajando en un *plugin* real y que desea subir a producción. Sólo tenga en cuenta que:

   - Cuando el generador le pregunte por el tipo de *plugin* a crear, seleccionar simple.
   - Cuando indique si la operación a crear es asíncrona, seleccionar N.
6. Listar el contenido del directorio:

   ```
   > dir
   ```
7. Abir el proyecto con su *IDE* favorito. En nuestro caso, usaremos Atom de GitHub:

   ```
   > atom .
   ```
8. Echar un vistazo al proyecto detenidamente.
9. Instalar dependencias:

   ```
   > npm install
   ```

El *plugin* lo vamos a desarrollar bajo la especificación ES2015. Por lo que tenemos que compilarlo para que pueda utilizarse en Node. Para ello, vamos a utilizar varios *plugins* oficiales: justo-plugin-babel para compilar; justo-plugin-jshint para comprobar buenas prácticas y gramática; justo-plugin-fs para borrar directorios y crear el paquete del *plugin* a publicar. Si fuésemos a publicar el paquete en NPM, podríamos utilizar justo-plugin-npm. Estos *plugins* deben indicarse en las propiedades de desarrollo del archivo package.json. Observe que el generador ya los ha añadido para ir más rápidos.

Por otra parte, el archivo Justo.js generado tiene definidas tareas para llevar a cabo la compilación,

pruebas, etc. También con objeto de ayudarnos a arrancar más rápidamente el desarrollo del *plugin*. Tal como veremos en un curso específico, uno de los principales objetivos de los generadores.

Desarrollo del plugin

El objetivo del *plugin* es crear una tarea simple reutilizable para copiar archivos. Esta tarea ya está implementada en el *plugin* justo-plugin-fs, pero necesitamos un punto de partida fácil con el que aprender a desarrollar *plugins*.

Como este *plugin* es simple, nos centraremos en los archivos siguientes: index.js y lib/op.js. Hay otros dos que son de vital importancia: test/unit/index.js y test/unit/lib/op.js. Contienen las pruebas de unidad, pero se salen del ámbito de este curso. Hay uno específico. Es importante saber que podemos realizarlas con Justo.

Recordemos que el archivo index.js debe exportar la función de tarea reutilizable:

1. Editar el archivo index.js.
2. Comprobar que el generador ya lo ha rellenado completamente. Evitándonos hacerlo nosotros mismos. Debido a que es un *plugin* simple, la función de tarea debe ser expuesta mediante el propio *plugin*:

```
//imports
import {simple} from "justo";

//internal data
const NS = "org.examples";

//api
module.exports = simple({ns: NS, name: "justo-uplugin-cp"},
require("./lib/op").default);
```

Ahora, vamos a centrarnos en el archivo lib/op.js. Por covenio, este archivo define la operación de tarea, aquella que se pasa a la función simple(). Esta función se define con el parámetro params. Recordemos, de tipo *array*, a través del cual el motor de Justo le pasará el objeto con las propiedades src y dst proporcionadas por el usuario en la invocación de la función de tarea.

1. Editar el archivo lib/op.js.
2. Implementar la operación para que copie un archivo. Para ello, usaremos la función copy del paquete justo-fs:

```
//imports
import * as fs from "justo-fs";

/**
 * Task operation.
 */
export default function op(params) {
  var opts, res;

  //(1) arguments
  if (params.length >= 1) opts = params[0];
  if (!opts) opts = {};
  if (!opts.src) throw new Error("src expected.");
  if (!opts.dst) throw new Error("dst expected.");

  //(2) run
  fs.copy(opts.src, opts.dst);

  //(3) return
  return 0;
}
```

3. Guardar cambios.

Una vez creado el *plugin*, sólo falta la parte de pruebas de unidad, pero eso se ve detenidamente en un curso posterior. Recordemos que Justo viene con su propia suite de pruebas. Por ahora, supongamos que todo lo hacemos bien a la primera, algo imposible, y no necesitamos automatizar nuestras pruebas.

Creación del paquete

En este punto, vamos a obtener la copia distribuible del *plugin*. Para ello, podemos usar la tarea catalogada build que crea automáticamente el generador:

1. Ir a la consola.
2. Listar las tareas catalogadas del proyecto:

   ```
   > justo -c
   ```
3. Generar el paquete distribuible:

   ```
   > justo build
   ```
4. Echar un vistazo al directorio dist, donde la tarea deja el *plugin* listo para su instalación.

   Si decidimos publicar el *plugin* en el repositorio NPM, tendremos que publicar el contenido del directorio dist/es5/nodejs/justo-uplugin-cp.

Utilización del plugin

Ahora, vamos a utilizar el *plugin*.

1. Abrir una nueva consola.
2. Crear un directorio de prueba e ir a él.
3. Crear un proyecto *standalone* de Justo con su generador:

   ```
   > justo -g justo
   ```
4. Instalar dependencias:

   ```
   > npm install
   ```
5. Instalar el *plugin* desde su directorio distribuible:

   ```
   > npm install su-dir-de-justo-uplugin-cp/dist/es5/nodejs/justo-uplugin-cp
   ```
   Si el *plugin* lo tuviéramos publicado en NPM, lo mejor es añadirlo a las dependencias del proyecto mediante las propiedades dependencies o devDependencies. En nuestro caso, lo vamos a usar sin publicación previa.
6. Comprobar que se ha instalado el *plugin*:

   ```
   > npm ls justo-uplugin-cp
   ```
7. Editar el archivo Justo.js.
8. Importar el *plugin*:

   ```
   const cp = require("justo-uplugin-cp");
   ```
9. Añadir la tarea copiar al catalogo:

   ```
   catalog.workflow(
     {name: "copiar", desc: "Copia un archivo pasado como argumento."},
     function(params) {
       cp("Copia archivos", {
         src: params[0],
         dst: params[1]
       });
     }
   );
   ```
10. Guardar cambios.
11. Ir a la consola.
12. Listar tareas catalogadas:

    ```
    > justo -c
    ```
13. Ejecutar la tarea para que copie el archivo Justo.js como Justo.js.bak:

    ```
    > justo copiar:Justo.js:Justo.js.bak
    ```
    Recuerde que los argumentos de la tarea se pasan junto al nombre, separándolos por dos puntos (:).
14. Listar los archivos que comienzan por Justo.js:

    ```
    > dir Justo.js*
    ```


Comprobar que en efecto se ha creado el archivo Justo.js.bak.

Creación de un plugin compuesto

Ahora, vamos a crear un *plugin* compuesto. Esta vez, el *plugin* tendrá también una única tarea: cp, igual que la anterior. Como ejercicio, le dejamos al estudiante que añada nuevas tareas, tal como vamos a ver.

Desarrollo del plugin

Comencemos escribiendo el *plugin*:

1. Abrir una nueva consola.
2. Crear el directorio justo-uplugin-fs e ir a él.
3. Crear la estructura inicial del *plugin* con el generador justo-generator-plugin:

   ```
   > justo -g plugin
   ```
   Esta vez, seleccionar composite como tipo de *plugin*.
4. Instalar dependencias:

   ```
   > npm install
   ```
5. Abrir el proyecto del *plugin* con su editor favorito. En caso de Atom:

   ```
   > atom .
   ```
6. Echar un vistazo al proyecto generado.
7. Añadir la tarea cp al *plugin* mediante el generador:

   ```
   > justo -g plugin add op
   ```
   Indique cp como nombre y seleccione N cuando le pregunte si se trata de una operación asíncrona.

   Observe lo que hace el generador. Crea el archivo lib/cp.js para que escribamos en él el código de la operación de tarea. Y por otra parte, registra la tarea en el archivo index.js.
8. Editar el archivo lib/cp.js:

   ```
   //imports
   import * as fs from "justo-fs";

   /**
    * Task operation.
    */
   export default function op(params) {
     var opts, res;

     //(1) arguments
     if (params.length >= 1) opts = params[0];
     if (!opts) opts = {};
     if (!opts.src) throw new Error("src expected.");
     if (!opts.dst) throw new Error("dst expected.");

     //(2) run
     fs.copy(opts.src, opts.dst);

     //(3) return
     return 0;
   }
   ```
9. Crear el paquete:

   ```
   > justo build
   ```

Utilización del plugin

Ahora, a utilizar el *plugin*.

1. Abrir una nueva consola.

2. Crear un directorio de prueba e ir a él.
3. Crear un proyecto *standalone* de Justo con el generador:

```
> justo -g justo
```

4. Instalar dependencias:

```
> npm install
```

5. Instalar el *plugin* desde su directorio distribuible:

```
> npm install su-dir-de-justo-uplugin-fs/dist/es5/nodejs/justo-uplugin-fs
```

6. Comprobar que se ha instalado el *plugin*:

```
> npm ls justo-uplugin-fs
```

7. Editar el archivo Justo.js.
8. Importar el *plugin*:

```
const fs = require("justo-uplugin-fs");
```

9. Añadir la tarea copiar al catálogo:

```
catalog.workflow(
  {name: "copiar", desc: "Copia un archivo pasado como argumento."},
  function(params) {
    fs.cp("Copia archivos", {
      src: params[0],
      dst: params[1]
    });
  }
);
```

Observe cómo se invoca la tarea del *plugin* compuesto: fs.cp().

10. Guardar cambios.
11. Ir a la consola.
12. Listar tareas catalogadas:

```
> justo -c
```

13. Ejecutar la tarea para que copie el archivo Justo.js como Justo.js.bak:

```
> justo copiar:Justo.js:Justo.js.bak
```

14. Listar los archivos que comienzan por Justo.js:

```
> dir Justo.js*
```

Comprobar que en efecto se ha creado el archivo Justo.js.bak.

