PLUGINS

Tiempo estimado: 15min

En software, es muy común desarrollar componentes reutilizables con objeto de no reinventar la rueda y mejorar el rendimiento, la calidad y la robustez, entre otras características. Justo.js permite extender su funcionalidad mediante *plugins*, componentes de software desarrollados en JavaScript que proporcionan tareas reutilizables.

Esta lección es muy sencilla. Comienza introduciendo el concepto de *plugin* y presentando algunos *plugins* oficiales. A continuación, muestra el directorio de un proyecto de desarrollo de *plugin*. Y finalmente, hace hincapié en cómo definir las operaciones de tarea y cómo exportar las funciones de tarea o de envoltura.

Al finalizar la lección, el estudiante sabrá:

- Qué es un *plugin*.
- Como definir *plugins* de usuario.

INTRODUCCIÓN

Un plugin es un componente que proporciona funcionalidad adicional a Justo.js. Concretamente, implementa una o más tareas para su reutilización. Por ejemplo, el *plugin* justo-plugin-fs proporciona varias tareas simples: clean o remove, para suprimir archivos y/o directorios; create, para crear archivos y/o directorios; y copy, para copiar archivos y/o directorios. justo-plugin-babel proporciona una única tarea, que permite automatizar trabajos de compilación mediante Babel.

Los *plugins* oficiales de Justo.js se encuentran en GitHub, github.com/justojsp. Y se publican en NPM. Cada *plugin* se encuentra en su correspondiente repositorio y se describe detalladamente mediante su archivo README.md. Actualmente, podemos encontrar varios *plugins*:

- justo-plugin-babel. Proporciona una tarea simple para ejecutar Babel.
- justo-plugin-bootlint. Proporciona una tarea simple para ejecutar bootlint, *lint* de Bootstrap.
- justo-plugin-browserify. Proporciona una tarea simple para ejecutar Browserify.
- justo-plugin-chrome. Proporciona tareas para trabajar con Google Chrome como, por ejemplo, abrirlo.
- justo-plugin-cli. Proporciona una tarea simple para ejecutar comandos en la máquina.
- justo-plugin-fs. Proporciona tareas simples para trabajar con archivos y/o directorios como, por ejemplo, copiar, suprimir, crear, etc.
- justo-plugin-gh-pages. Proporciona una tarea para publicar en GitHub Pages.
- justo-plugin-jshint. Proporciona una tarea simple para ejecutar JSHint.
- justo-plugin-less. Proporciona tareas para compilar hojas de estilo Less a CSS.
- justo-plugin-mocha. Proporciona una tarea simple para ejecutar mocha.
- justo-plugin-npm. Proporciona tareas simples para ejecutar npm, por ejemplo, para publicar paquetes en el repositorio NPM.
- justo-plugin-ping. Proporciona una tarea simple para realizar un *ping* a una máquina.
- justo-plugin-soffice. Proporciona tareas para ejecutar comandos de Apache OpenOffice o LibreOffice como, por ejemplo, la conversión de archivos .odt a .pdf.
- justo-plugin-tidy. Proporciona una tarea para ejecutar HTML TIDY.
- justo-plugin-zip. Proporciona una tarea simple para comprimir en formato zip.

Actualmente, se está trabajando en *plugins* específicos de sistemas de gestión de bases de datos como:

- justo-plugin-couchdb para CouchDB.
- justo-plugin-mongodb para MongoDB.
- justo-plugin-redis para Redis.
- justo-plugin-postgresql para PostgreSQL.

Desarrollo de plugins

Los *plugins* se escriben en JavaScript, concretamente, son paquetes de Node.js y generalmente se publican en algún tipo de repositorio como, por ejemplo, NPM. Por lo tanto, es necesario conocer la plataforma Node.js para su desarrollo.

Básicamente, distinguimos dos tipos de *plugins*, los simples y los compuestos. Un plugin simple (*simple plugin)* es aquel que proporciona una única tarea. Mientras que un plugin compuesto (*composite plugin*), varias.

La estructura de directorios inicial de un proyecto para el desarrollo de un *plugin* se puede crear mediante el generador justo-generator-plugin y el comando justo:

```
npm install -g justo-generator-plugin
justo -g plugin
```

El generador proporciona el parámetro type a través del cual indicar el tipo de *plugin* a crear. De manera predeterminada, se crea como simple. Si necesitamos crearlo como compuesto, usar type:composite:

```
justo -g plugin type:composite
```

Para conocer los parámetros del generador, podemos utilizar el comando siguiente:

```
justo -g plugin help
```

Archivo package.json

Como todo paquete de Node.js, el *plugin* debe definir el archivo package.json. Y debe de presentar como mínimo las siguientes propiedades:

- name. Nombre del *plugin*.

  Los *plugins* oficiales proporcionados por el equipo de Justo.js usan como formato de nombre justo-plugin-nombre, como por ejemplo justo-plugin-fs o justo-plugin-redis. Los *plugins* no oficiales, esto es, los de usuario, no deben seguir este convenio de nomenclatura, recomendándose el formato justo-uplugin-nombre.

- version. Versión del *plugin*.
- description. Breve descripción del *plugin*.
- author. Desarrollador del *plugin*.
- keywords. Palabras claves que describen el *plugin*. Todo *plugin* debe tener entre sus palabras claves justo-plugin. Esto ayudará a buscarlo en el repositorio de NPM, donde se suelen publicar.
- bugs. Información para comunicar los *bugs* a su desarrollador.
- dependencies. Dependencias de ejecución del *plugin*.
- devDependencies. Debe tener justo entre sus dependencias de desarrollo.
- main. Archivo principal que se cargará cuando se cargue el *plugin*. Por convenio y buenas prácticas, index.js.
- files: Archivos y directorios que se instalarán. Generalmente, el directorio lib.

Para ver un ejemplo, consulte cualquiera de los *plugins* oficiales en su repositorio de GitHub. Son un buen punto de partida.

Archivo README.md

Se recomienda encarecidamente escribir un archivo README.md, formato Markdown, con las instrucciones de uso del *plugin* y de sus tareas reutilizables. Markdown es muy sencillo de aprender y utilizar.

Archivos justo.js y justo.json

Se recomienda encarecidamente utilizar Justo.js para la automatización de las tareas de desarrollo y pruebas del *plugin*. Si así lo hacemos, está claro que deberemos definir los archivos Justo.js y Justo.json. De ahí que tengamos como dependencia de desarrollo el paquete justo en el archivo package.json.

Directorio lib

Por convenio y buenas prácticas, el código del *plugin* se ubica en el directorio lib del paquete, con un archivo para cada operación de tarea.

Directorio test

Por buenas prácticas, todo *plugin* debe tener una batería de pruebas. La cual, recordemos, se puede implementar usando Justo.js. Las pruebas de unidad se ubican en el directorio test/unit, el cual contiene dos directorios adicionales, data, donde ubicar los archivos de datos usados en las pruebas; y lib, las suites de prueba de las tareas.

Esto es sólo un convenio, muy recomendable, pero cada organización puede definir los suyos propios.

Definición de operaciones de tarea

Los *plugins* son componentes que contienen tareas reutilizables, ya sean tareas simples, macros o flujos de trabajo. Por buenas prácticas, cada operación de tarea se define en su propio archivo, mientras que la función de tarea o envoltorio se deja para el archivo index.js.

Por ejemplo, el *plugin* justo-plugin-fs tiene los archivos lib/create.js, lib/clean.js, lib/copy.js, etc. donde define las operaciones de tarea. Cuando el *plugin* contiene una única tarea simple y se exporta de manera predeterminada, su operación se define en el archivo lib/op.js. Esto son sólo convenios, no obligaciones, pudiendo cada organización disponer de sus propios estándares.

He aquí una plantilla de archivo operación:

```
/**
 * Task operation.
 */
export default function op(params) {

}
```

Archivo index.js

Una vez definidas las operaciones de tarea, lo siguiente que nos queda es definir la API del *plugin*, esto es, las tareas reutilizables. Esto se deja para el archivo index.js. Nada mejor que un ejemplo ilustrativo para entender cuál debe de ser su contenido. Comencemos con un *plugin* que exporta una única tarea y lo hace de manera predeterminada:

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

//api
module.exports = simple({ns: "org.justojs.plugin", name: "jshint"}, require("./lib/op").default);
```

Este ejemplo es muy sencillo. El *plugin* exporta una única tarea, además de hacerlo de manera predeterminada. Concretamente, es el archivo index.js del *plugin* justo-plugin-jshint. Todo *plugin* debe exponer las funciones de tarea o envoltura devueltas por simple(), macro() o workflow().

Ahora, un ejemplo de *plugin* compuesto:

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



```
//internal data
const NS = "org.justo.plugin.fs";
var clean, copy, create;

module.exports = {
  get clean() {
    if (!clean) clean = simple({ns: NS, name: "clean"}, require("./lib/clean").default);
    return clean;
  },

  get remove() {
    return this.clean;
  },

  get copy() {
    if (!copy) copy = simple({ns: NS, name: "copy"}, require("./lib/copy").default);
    return copy;
  },

  get create() {
    if (!create) create = simple({ns: NS, name: "create"}, require("./lib/create").default);
    return create;
  }
};
```

Este *plugin* sólo crea las tareas y carga sus archivos de operación si el usuario las usa. Si por ejemplo el usuario usa la tarea clean, no cargará ni copy ni create. Haciendo así su carga más rápida. Si no importa la velocidad de carga, lo anterior podría definirse mucho más fácilmente como sigue:

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

//internal data
const NS = "org.justo.plugin.fs";

module.exports = {
  clean: simple({ns: NS, name: "clean"}, require("./lib/clean").default),
  copy: simple({ns: NS, name: "copy"}, require("./lib/copy").default),
  create: simple({ns: NS, name: "create"}, require("./lib/create").default),
  get remove() {
    return this.clean;
  }
};
```

Recordemos que los espacios de nombres org.justojs y com.justojs se encuentran reservados y no se debe de definir ningún *plugin* de usuario bajo estos espacios de nombres.

