Tareas simples (práctica)

Tiempo estimado: 15min

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

Al finalizarla, el estudiante:

- Habrá creado el directorio de un módulo de Justo.
- Habrá creado tareas simples.
- Habrá registrado tareas en el catálogo de un proyecto.
- Habrá listado las tareas registradas de un proyecto.
- Habrá invocado tareas del catálogo de un proyecto.

Objetivos

El objetivo de la práctica es crear tareas simples. Para ello, crearemos dos tareas simples: una síncrona y otra asíncrona. Y las invocaremos.

Creación del proyecto

Para comenzar, vamos a crear el directorio de un módulo de Justo:

1. Abrir una consola.
2. Crear el directorio de la práctica e ir a él.
3. Crear el proyecto de módulo mediante el generador justo-generator-justo:

   ```
   $ justo -g justo module
   ```
   Responda con los valores predeterminados.
4. Instalar las dependencias del proyecto mediante npm:

   ```
   $ npm install
   ```
5. Listar las tareas del catálogo del proyecto:

   ```
   $ justo -c
   ```
6. Listar las tareas del catálogo de desarrollo:

   ```
   $ justo -dc
   ```

Registro de tareas simples síncronas

A continuación, vamos a crear una tarea simple síncrona, que comprueba si un archivo, pasado como argumento, existe. Para ello, usaremos la función fs.accessSync() de Node.js que, recordemos, propaga un error cuando el archivo no pasa la comprobación.

1. Ir a la consola.
2. Editar el archivo Justo.js.
3. Registrar una tarea simple con el nombre comprobar que realice lo deseado. El archivo podría quedar como sigue:

   ```
   //imports
   const justo = require("justo");
   const catalog = justo.catalog;
   const fs = require("fs");

   //catalog
   catalog.simple(
   ```



```
  {name: "comprobar", desc: "Comprobar que un archivo existe."},
  function(params) {
    fs.accessSync(params[0]);
  }
);

catalog.macro({name: "default", desc: "Default task."}, []);
```

4. Guardar cambios.
5. Ir a la consola.
6. Listar las tareas registradas en el catálogo del proyecto:

```
$ justo -c
Name       Description
comprobar  Comprobar que un archivo existe.
default    Default task.
$
```

7. Ejecutar la tarea comprobar para que compruebe si existe el archivo Justo.js:

```
$ justo comprobar:Justo.js

  comprobar
  [ OK ] comprobar (0 ms)

  OK 1 | Failed 0 | Ignored 0 | Total 1

$
```

Observar que al finalizar la ejecución, Justo muestra un resumen de las tareas ejecutadas.

8. Ejecutar la tarea comprobar pasando como argumento desconocido.txt:

```
$ justo comprobar:desconocido.txt

  comprobar
  [ ER ] comprobar (1 ms)
    Error: ENOENT: no such file or directory, access 'desconocido.txt'

  OK 0 | Failed 1 | Ignored 0 | Total 1

$
```

Observe que si la operación de tarea propaga un error, el motor considerará que la invocación ha acabado en fallo y así lo mostrará en el informe resumen.

Tareas simples asíncronas

Ahora, vamos a hacer lo mismo que antes, pero mediante una tarea asíncrona. En este caso, usaremos la función fs.access().

1. Editar el archivo Justo.js.
2. Añadir una tarea simple asíncrona con el nombre comprobar2 que realice lo deseado:

```
catalog.simple(
  {name: "comprobar2", desc: "Comprobar que un archivo existe asíncronamente."},
  function(params, done) {
    fs.access(params[0], done);
  }
);
```

En este caso, la función fs.access() es asíncrona. Como primer parámetro, espera el archivo a comprobar; como segundo, la función que debe invocar cuando haya finalizado su operación completamente. Esta función la invocará sin argumentos, si todo ha ido bien, o pasándole el error como primer argumento. De ahí que podamos pasarle nuestro parámetro done sin problemas.

Otra manera de hacer lo mismo sería como sigue:

```
catalog.simple(
  {name: "comprobar2", desc: "Comprobar que un archivo existe asíncronamente."},
  function(params, done) {
```



```
    fs.access(params[0], function(error) {
      if (error) done(error);
      else done();
    });
  }
);
```

Recordemos que las tareas utilizan el parámetro función done() para indicarle al motor de Justo que han terminado. Si no se invoca en una tarea asíncrona, el motor se quedará esperando.

3. Guardar cambios.
4. Ir a la consola.
5. Listar el catálogo de tareas:

```
$ justo -c
Name        Description
comprobar   Comprobar que un archivo existe.
comprobar2  Comprobar que un archivo existe asíncronamente.
default     Default task.
$
```

6. Comprobar que el archivo Justo.js existe asíncronamente:

```
$ justo comprobar2:Justo.js

  comprobar2
  [ OK ] comprobar2 (5 ms)

  OK 1 | Failed 0 | Ignored 0 | Total 1

$
```

Tarea predeterminada

Finalmente, vamos a configurar la tarea predeterminada para que compruebe que los archivos Justo.js y noexiste.txt existen:

1. Editar el archivo Justo.js.
2. Modificar la definición de la tarea predeterminada para que invoque la tarea del catálogo comprobar dos veces, una para cada archivo:

```
catalog.macro(
  {name: "default", desc: "Default task."},
  [
    {task: "comprobar", params: ["noexiste.txt"]},
    {task: "comprobar", params: ["Justo.js"]}
  ]
);
```

En una lección posterior, se expone detalladamente las macros.

3. Guardar cambios.
4. Ir a la consola.
5. Ejecutar la tarea predeterminada:

```
> justo

  default
    default
    [ ER ] comprobar (1 ms)
    Error: ENOENT: no such file or directory, access 'noexiste.txt'
    [ OK ] comprobar (0 ms)

  OK 1 | Failed 1 | Ignored 0 | Total 2

>
```

La tarea predeterminada se puede invocar explícitamente mediante justo default:

```
> justo default
```



```
default
  default
  [ ER ] comprobar (1 ms)
  Error: ENOENT: no such file or directory, access 'noexiste.txt'
  [ OK ] comprobar (0 ms)

OK 1 | Failed 1 | Ignored 0 | Total 2
```

```
>
```

6. Invocar la tarea predeterminada con la opción -b:

```
> justo -b

default
  default
  [ ER ] comprobar (1 ms)
  Error: ENOENT: no such file or directory, access 'noexiste.txt'

OK 0 | Failed 1 | Ignored 0 | Total 1
```

```
>
```

De manera predeterminada, el motor no se detiene cuando falla una tarea. Continúa. Si deseamos que deje de ejecutar tareas si alguna falla, hay que indicar la opción -b o --break.

7. Cerrar la consola.

