Macros

Tiempo estimado: 10min

En la lección anterior, presentamos el concepto de tarea simple como la unidad de trabajo más pequeña ejecutable y monitorizable. En esta lección, presentamos las tareas compuestas, particularmente las macros.

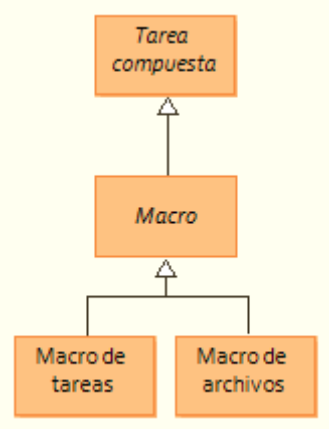
Primero, introducimos el concepto de tarea compuesta como un tipo especializado de tarea. Para después, presentar el concepto de macro, un tipo de tarea compuesta. Finalmente, mostramos detenidamente los dos tipos de macro, las de tareas y las de archivos.

Al finalizar la lección, el estudiante sabrá:

- Qué es una tarea compuesta.
- Qué es una macro.
- Cómo definir una macro, ya sea de tareas o de archivos.
- Cómo invocar una macro, independientemente de su tipo.
- Cómo registrar macros en el catálogo del proyecto.

Introducción

Una tarea compuesta (*composite task*) representa un trabajo formado por cero, una o más tareas. Se distingue entre macros y flujos de trabajo. Una macro (*macro*) es una secuencia ordenada de cero, una o más tareas. Cada vez que se invoca la macro, se ejecuta cada una de sus tareas, en el orden indicado. Por su parte, un flujo de trabajo (*work flow*), objeto de la siguiente lección, representa una actividad, proceso o trabajo que puede ejecutar unas tareas u otras atendiendo a determinadas condiciones de control de flujo.



Las macros se clasifican en dos tipos, de tareas y de archivos. Una macro de tareas (*task macro*) es aquella cuya secuencia está formada por tareas. Mientras que una macro de archivos (*file macro*) lo que hace es cargar o ejecutar una colección de archivos, uno detrás de otro.

Macros de tareas

Tal como acabamos de ver, una macro de tareas (*task macro*) es una secuencia de cero, una o más tareas. Cada vez que invocamos la macro, el motor de Justo ejecuta ordenadamente la secuencia de tareas. El resultado final de la macro, depende del de sus tareas. Si todas se ejecutan correctamente, la macro finalizará también correctamente; en cambio, si alguna acaba en fallo, la macro también lo hará.

Por lo general, una macro de tareas se crea para definir aliases de tareas o bien secuencias de tareas

cuya ejecución es frecuente y se desea invocar mediante una única llamada.

Definición de macros de tareas

Para definir una macro de tareas, se utiliza la función macro() del paquete justo:

```
function macro(name : string, tasks : object[]) : function
function macro(opts : object, tasks : object[]) : function
```

Parámetro	Tipo de datos	Descripción
name	string	Nombre identificativo de la macro.
opts	object	Propiedades de la macro. Las mismas que las presentadas en simple() como, por ejemplo, name, ns, title, desc, ignore, etc.
tasks	object[]	*Array* de tareas asociadas a la macro.

Las tareas se deben pasar en un *array*, el cual puede tener dos tipos de elementos: una cadena de texto o un objeto.

Cuando se indica un objeto, debe contener la información de la tarea a invocar. Las propiedades son las siguientes:

- task (string o function). La tarea a ejecutar. Se puede indicar su nombre, si es una tarea catalogada, o bien su función de tarea o envoltorio, recordemos, la función devuelta por simple(), macro() o workflow().
- params (object[]). Los argumentos predeterminados con los que invocar la tarea.
- title (string). El título asociado a la invocación de la tarea en el informe resultado.

La propiedad task debe contener una función de tarea o envoltorio. Representando así la tarea que debe invocarse. Si se desea hacer referencia a una tarea catalogada, se puede indicar su nombre. Veamos un ejemplo ilustrativo:

```
catalog.macro("default", [
  {title: "uno", task: "nombreTareaCatalogada", params: [1, 2, 3]},
  {title: "dos", task: fnTarea, params: [3, 2, 1}
]);
```

También es posible indicar simplemente el nombre de una tarea catalogada. Ejemplo:

```
catalog.macro("default", [
  "uno",
  {title: "dos", task: fnTarea, params: [3, 2, 1]}
]);
```

Registro de macros catalogadas

Tal como acabamos de ver en el ejemplo anterior, se puede registrar una macro de tareas en el catálogo mediante la función catalog.macro(), cuya sintaxis es similar a macro():

```
function catalog.macro(name : string, tasks : object[]) : function
function catalog.macro(props : object, tasks : object[]) : function
```

La función registra la macro en el catálogo y devuelve su función de tarea o envoltorio.

Invocación de macros de tareas

La invocación de una macro de tarea se solicita de manera similar a lo presentado en las tareas simples, mediante la función de tarea o envoltura devuelta por la función macro(). Su sintaxis es la misma. El primer argumento es interno de Justo; y el resto, los parámetros que se pasarán a las tareas de la macro.

Hay que hacer un poco de hincapié en los argumentos pasados a la macro. Como vimos con las tareas simples, los argumentos del segundo en adelante se pasan a la operación de tarea a través del parámetro params. Las macros son similares, pero tienen algunas peculiaridades. Tal como acabamos de ver, a la invocación de una tarea de macro se le puede especificar una propiedad params. Cuando no se pasa ningún argumento en la invocación de la macro, se pasará el valor de esta propiedad, la cual debe ser de tipo *array*. Si se pasa alguno, a todas las tareas de la macro, se les pasará los argumentos pasados en la invocación de la macro.

Veámoslo mediante un ejemplo. Supongamos una macro, m, con dos tareas, t1 y t2:

```
m: [
  {task: t1, params[1, 2, 3]},
  {task: t2}
]
```

Ahora, supongamos la siguiente invocación de m:

```
m("ejemplo");
```

Bajo esta situación, Justo invocará cada tarea de la macro con sus parámetros predeterminados. Esto es, t1 con [1, 2, 3] y t2 con [].

Ahora bien, qué pasa si tenemos la siguiente invocación:

```
m("ejemplo", 1, 3, 5);
```

Pues que a cada tarea de la macro se le pasará como parámetros el *array* [1, 3, 5], omitiendo los predeterminados que pudiera tener definidos.

Macros de archivos

Una macro de archivos (*file macro*) es aquella que ejecuta, secuencialmente, una serie de archivos.

Definición de macros de archivos

Para definir una macro de este tipo, se utiliza también la función macro(), pero con las siguientes signaturas:

```
function macro(name : string, info : object) : function
function macro(opts : object, info : object) : function
```

Parámetro	Tipo de datos	Descripción
name	string	Nombre identificativo de la macro.
opts	object	Propiedades de la macro.
info	object	Información de ejecución: • src (string o string[]). Rutas de los archivos y/o directorios a cargar. • require (string o string[]). Nombre de los paquetes a importar.

A través de la propiedad src, se indica los archivos a invocar. Se puede indicar tanto archivos como directorios. Cuando se indica un directorio, se ejecutará cada uno de los archivos ahí indicados. Por su parte, la propiedad require se utiliza para indicar módulos de Node.js que deseamos se carguen antes de cargar los archivos.

Por lo general, este tipo de macros se utiliza para definir tareas de pruebas de unidad. He aquí un ejemplo ilustrativo:

```
catalog.macro({name: "test", desc: "Unit test."}, {
  require: "justo-assert",
  src: ["test/unit/index.js", "test/unit/lib/"]
});
```

Registro de macros catalogadas

El registro de una macro de archivos en el catálogo se realiza mediante la función catalog.macro():

```
function catalog.macro(name : string, info : object) : function
function catalog.macro(props : object, info : object) : function
```

La función registra la macro en el catálogo y devuelve su función de tarea o envoltorio.

Invocación de macros de archivos

La invocación de una macro de archivos se solicita de manera similar a lo presentado hasta ahora con las tareas simples y las macros de tareas. Mediante la función de tarea o envoltura devuelta por la función macro(). La única diferencia es que, actualmente, a una macro de archivos no se le puede pasar parámetros, más allá del utilizado internamente por Justo.

