Módulos en Lua

Tiempo estimado: 15min

En este apéndice, describimos el concepto de módulo en Lua. Los componentes que permiten organizar nuestro código fuente y, si lo deseamos, reutilizarlo en otros proyectos.

Para comenzar, presentamos los dos conceptos claves de la reutilización de código Lua, los módulos y los paquetes. A continuación, nos centramos en los módulos para acabar la lección mostrando los paquetes.

Al finalizar la lección, el estudiante sabrá:

- Qué es un módulo.
- Cómo desarrollar módulos.
- Cómo importar o reutilizar módulos.
- Qué es un paquete.
- Cómo escribir paquetes.

Introducción

Un módulo (*module*) es un archivo de Lua o un archivo binario que define un objeto reutilizable. En Lua, todo módulo es un objeto de tipo tabla. Tiene una interfaz de aplicaciones (*API*) bien definida y lleva a cabo una o más tareas específicas y relacionadas entre sí con una funcionalidad o dominio.

Por su parte, un paquete (*package*) es un contenedor de módulos.

Los módulos, así como los paquetes, son elementos que facilitan la organización del código, su prueba y su reutilización. Esto ayuda a administrar el código fuente de la aplicación, haciéndola más fácil y eficiente.

Módulos

Un módulo (*module*) es un componente independiente y reutilizable. Se define mediante un archivo de código .lua, el cual podemos cargar en nuestra aplicación siempre que necesitemos su funcionalidad.

En Lua, en momento de ejecución, un módulo se representa mediante un objeto de tipo tabla. Este objeto contiene todos los objetos expuestos por el módulo para su reutilización. Lo que se conoce formalmente como su API (*Application Programming Interface*, Interfaz de Programación de Aplicaciones), recordemos, un conjunto de objetos que se ofrece para su reutilización en otros componentes.

Todo módulo tiene su propio espacio de nombres privado. Cada vez que definimos una variable con la sentencia local, se definirá como local al módulo. Pero cuidado, no lo olvidemos nunca, cuando definimos una variable sin la sentencia local, se comportará como global. Así pues, cuando vayamos a definir una variable dentro de un módulo, lo haremos siempre con local, independientemente de dónde lo hagamos dentro del módulo. Las funciones también se deben definir con local, si no están asociadas a ningún objeto explícitamente.

Objeto módulo

Todo módulo debe devolver un objeto tabla, recordemos, el que representa al módulo y contiene su API reutilizable. Este objeto se puede devolver mediante una sentencia return, al final del archivo del módulo, o bien se puede asignar a la variable global package.loaded. Generalmente, se prefiere la segunda opción, porque permite ubicar la API del módulo al comienzo facilitando así su documentación y visualización.

Algunos desarrolladores, sobre todo aquellos que proceden de Node.js, suelen definir este objeto en la

variable local exports. La cual se asignará a package.loaded como sigue:

```
--api
local exports = {}
package.loaded[...] = exports
```

A continuación, cualquier objeto que forme la API del módulo se asignará a esta variable.

Por ejemplo, supongamos que vamos a definir un módulo que representa la API de una calculadora. Podríamos tener algo muy parecido a lo siguiente:

```
--api
local exports = {}
package.loaded[...] = exports

--exports
function exports.sum(x, y) return x + y; end
function exports.sub(x, y) return x - y; end
```

Importación de módulos

La importación de un módulo (*module import*) es la operación mediante la cual solicitamos al intérprete de Lua que cargue un módulo y nos devuelva su objeto tabla, según nuestro convenio, la variable exports del módulo. La cual, recordemos, exponemos al entorno de ejecución mediante la variable global package.loaded.

Esta operación se realiza mediante la función require():

```
function require(module)
```

Parámetro	Tipo de datos	Descripción
module	string	Nombre del módulo a importar.

A continuación, se muestra un ejemplo que ilustra cómo importar el módulo definido en el archivo calcul.lua:

```
local calcul = require("calcul")
```

Por convenio, el objeto módulo se suele asociar a una variable con el mismo nombre que el módulo. A partir de este momento, podemos utilizar las propiedades expuestas por el objeto módulo. He aquí un ejemplo ilustrativo:

```
print(calcul.sum(1, 2))
```

Caché de módulos

La cache de módulos (*module cache*) es un contenedor en el que la función require() mantiene los módulos ya cargados, para no cargarlos una y otra vez cada vez que los importemos a lo largo del programa. Esta caché se encuentra disponible en la variable global package.loaded, de tipo tabla, la cual contiene un campo para cada módulo cargado.

Cuando se importa un módulo con require(), la función primero consulta si tiene una entrada en la caché. Si es así, devuelve el objeto ahí indicado. En otro caso, lo busca, lo registra en la caché y devuelve el objeto módulo al usuario.

Cuando carga un módulo, si éste no devuelve nada, devuelve el objeto de package.loaded. Esta es la razón por la que podemos exponer la variable exports directamente en la caché mediante package.loaded[...], evitándonos así añadir una sentencia return al final del módulo. Tengamos en cuenta que si el módulo está ejecutando esa proposición, es porque no está cargado, por lo que podemos añadir el objeto módulo a la caché directamente.

En muy contadas ocasiones, generalmente durante las pruebas, se suele necesitar recargar un módulo. Para obligar a que la función require() vuelva a cargar el contenido de un módulo, bastará con asignar el valor nil al módulo en la caché. Veamos un ejemplo:

```
package.loaded["calcul"] = nil;
```

Búsqueda de módulos

Cuando la caché no dispone del módulo solicitado, la función require() debe buscarlo y cargarlo en ella. La búsqueda de un módulo (*module search*) es, pues, la operación mediante la cual se busca un

módulo en el disco. Para reducir los directorios donde debe buscar, la función utiliza el valor de la variable global package.path, la cual se inicializa al comenzar el intérprete con el valor de la variable de entorno LUA_PATH. Si LUA_PATH no existe, el intérprete le fijará su valor predeterminado.

Una ruta (*path*) es la ubicación concreta de disco donde se encuentra un módulo. El valor de package.path es una cadena de texto que contiene las rutas de los módulos reutilizables, separadas entre sí mediante punto y coma (;), independientemente del sistema operativo. He aquí un ejemplo ilustrativo:

```
> package.path
/usr/local/share/lua/5.3/?.lua;/usr/local/share/lua/5.3/?/init.lua;/usr/local/lib/lua/5.3/?.
lua;/usr/local/lib/lua/5.3/?/init.lua;./?.lua;./?/init.lua
>
```

Cada ruta representa un posible archivo. Para permitir que las rutas se adapten a cada módulo buscado, evitando así tener que indicar las rutas de todos los módulos, se utiliza rutas dinámicas. En este tipo de rutas, se utiliza el símbolo de interrogación (?), el cual representa el nombre del módulo importado, adaptándose así a cada importación.

En el momento que encuentra uno de los archivos, la función lo lee, registra su objeto módulo en la caché y se lo devuelve al usuario. Dejando de buscar en el resto de rutas. Pero si no lo encuentra en ninguna de ellas, propagará un error.

La función require() utiliza una segunda variable global en las búsquedas, package.cpath. A diferencia de la anterior, que se utiliza para la búsqueda de módulos escritos íntegramente en Lua, ésta se utiliza para importar y reutilizar bibliotecas dinámicas. En Linux, archivos .so; y en Windows, .dll. Veamos un ejemplo ilustrativo:

```
> package.cpath
/usr/local/lib/lua/5.3/?.so;/usr/local/lib/lua/5.3/loadall.so;./?.so
>
```

Paquetes

Otro contenedor de código reutilizable es el paquete (*package*). No es más que una colección de módulos. Se representa mediante un directorio del disco, donde cada módulo será un módulo del paquete. Si el paquete contiene otros paquetes, éstos se representarán como subdirectorios.

La importación de módulos ubicados dentro de paquetes es similar a la de un módulo independiente, salvo que se precede por el nombre del paquete seguido de un punto. Así, por ejemplo, el módulo url del paquete socket se importará como sigue:

```
local url = require("socket.url");
```

En este caso, la función buscará el directorio socket y dentro de él un archivo url.lua o bien url.so o url.dll según el sistema operativo.

Si el módulo se encuentra en un subpaquete, basta con indicar su jerarquía separando los paquetes por puntos. Ejemplo: p1.p2.p3.módulo.

Iniciador del paquete

El iniciador del paquete (*package initializer*) es un archivo init.lua que contiene la lógica inicial del paquete. Se ubica en el directorio del paquete y contendrá cualquier operación inicial que deba llevar a cabo si se importa uno de sus módulos.

La función require() sólo ejecutará este archivo si se importa uno de sus módulos. Y en caso de hacerlo, sólo lo hará una única vez. Si importamos dos módulos a y b, en este orden, el archivo se ejecutará con la importación del módulo a, no así con el de b.

