Iteradores

Un iterador (iterator) en Lua es una estructura del lenguaje que permite recorrer los elementos de una tabla en el orden en el que están definidos. Los iteradores se presentan como una función, la cual, a cada iteración devuelve el siguiente elemento de la lista. El uso más común de un iterador es el de recorrer los elementos de una tabla dentro de un bucle para, por ejemplo, imprimir por la consola los valores.

El principio de funcionamiento de un iterador es simple: en un bucle for, el iterador es llamado a cada iteración. El iterador realiza la lógica necesaria y devuelve un valor. Cuando la tabla ha sido recorrida completamente el iterador devuelve `nil` y el bucle finaliza.

En Lua encontramos tres tipos de iteradores:

- **Iteradores genéricos**.
- **Iteradores no genéricos sin estado**.
- **Iteradores no genéricos con estado complejo**.

Veamos a continuación los diferentes tipos de iteradores que están disponibles en Lua.

Iterador genérico para bucles for

Los iteradores genéricos para bucles for están definidos en la librería estándar de Lua. Son dos funciones que extraen cada uno de los elementos de la tabla en forma de pares tipo clave/valor. La clave se corresponde con el índice/clave que ocupa el elemento en la tabla. Veamos a continuación un ejemplo:

```
paises = {"Suiza", "Espana", "Colombia", "Italia", "Francia"}


for clave, valor in ipairs(paises)
do
  print (clave, valor)
end
```

Al ejecutar el ejemplo anterior obtenemos el siguiente resultado:

```
1       Suiza
2       Espana
```

```
3       Colombia
4       Italia
5       Francia
```

Iteradores genéricos `ipairs()` y `pairs()`

En el ejemplo anterior, se ha presentado la funcion `ipairs()` que extrae los elementos en forma de índice/valor. `ipairs()` está orientada a las tablas de tipo array, como en el ejemplo anterior. Para cada elemento de la tabla, `ipairs()` nos devuelve el índice y su valor.

Existe otro iterador genérico, `pairs()`, que permite extraer los elementos de una tabla tipo diccionario. En esta estructura, la clave está definida con un valor no numérico. Veamos un ejemplo:

```
parametros = { ["host"] = "localhost", ["port"] = "5432", ["user"] = "postgres", ["database"]
= "postgres" }

for clave, valor in pairs(parametros) do
    print(clave, valor)
end
```

El resultado del código anterior es:

```
database        postgres
port    5432
user    postgres
host    localhost
```

Como puedes observar, el valor devuelto en `clave` es la clave que ha sido definida para el elemento y, `valor`, devuelve el valor asignado a esa clave.

La función `pairs()` tiene algunas particularidades comparada con `ipairs()`. Por ejemplo, `pairs()` soporta solamente tablas de tipo hash (diccionarios). Si le pasamos un array no devuelve nada. Contrariamente si le pasamos un array a `pairs()` lo itera correctamente del mismo modo que lo hace `ipairs()`. Otro aspecto importante, es que `pairs()` no garantiza que la iteración de los elementos se produzca en el mismo orden en el que están definidos en la tabla. Por el contrario `ipairs()` sí garantiza que el orden de iteración corresponde con el orden dentro de la tabla. Veamos un ejemplo comparativo.

```
parametros = { ["host"] = "localhost", ["port"] = "5432", ["user"] = "postgres", ["database"]
= "postgres" }
paises = {"Suiza", "Espana", "Colombia", "Italia", "Francia"}
```

```
print ("Iterando diccionario con pairs")

for clave, valor in pairs(parametros) do
    print(clave, valor)
end

print ("Iterando diccionario con ipairs")

for clave, valor in ipairs(parametros) do
    print(clave, valor)
end

print ("Iterando array con ipairs")

for clave, valor in ipairs(paises) do
    print(clave, valor)
end

print ("Iterando array con pairs")

for clave, valor in pairs(paises) do
    print(clave, valor)
end
```

Cuando ejecutamos el ejemplo obtenemos la siguiente salida:

```
Iterando diccionario con pairs
user    postgres
port    5432
database        postgres
host    localhost
Iterando diccionario con ipairs
Iterando array con ipairs
1       Suiza
2       Espana
3       Colombia
4       Italia
5       Francia
Iterando array con pairs
```

```
1       Suiza
2       Espana
3       Colombia
4       Italia
5       Francia
```

Vamos a analizar el resultado de la ejecución del código anterior por bloques.

```
Iterando diccionario con pairs
user    postgres
port    5432
database        postgres
host    localhost
```

En este primer bloque, la función `pairs()` ha iterado el diccionario que le hemos pasado, pero observa que los elementos han sido extraidos en un orden diferente al que tienen dentro de la tabla. El primer elemento de la tabla era `{["host"] = "localhost"}`, sin embargo, en la salida por consola aparece el último. Este comportamiento no es predecible y a cada ejecución el orden puede variar.

```
Iterando diccionario con ipairs
```

El segundo bloque, solo devuelve el título. Como explicábamos anteriormente, `ipairs()` no es capaz de iterar tablas de tipo hash, con lo cual no devuelve nada.

```
Iterando array con ipairs
1       Suiza
2       Espana
3       Colombia
4       Italia
5       Francia
```

En este tercer bloque estamos iterando un array con la función `ipairs()`. El resultado es el esperado: extrae de manera secuencial cada uno de los elementos en el orden de indexación.

```
Iterando array con pairs
1       Suiza
2       Espana
3       Colombia
4       Italia
5       Francia
```

En este cuarto y último bloque, estamos iterando un array con la función `pairs()`. Como puedes observar se comporta igual que `ipairs()`: todos los elementos han sido iterados de forma secuencial en el orden de indexación. Para el caso de arrays, `pairs()` sí tiene un comportamiento predecible y, para cada ejecución, producirá el mismo resultado.

Iteradores no genéricos

Hasta ahora hemos visto los iteradores genéricos que están definidos en la librería estándar de Lua. Estos iteradores son útiles para casos comunes y pueden servir para casi cualquier situación.

Sin embargo en casos particulares, los iteradores genéricos pueden no ser lo suficientemente específicos, es por ello que Lua soporta los iteradores definidos por el desarrollador.

En Lua podemos definir dos tipos de iteradores no genéricos: sin estado y con estado.

Iteradores no genéricos sin estado

Los iteradores no genéricos sin estado, tal como su nombre indica, no guardan su estado por si mismos. Esto significa que podemos definir un iterador y usarlo en varios bucles sin que la ejecución de uno afecte al otro.

El iterador se define como una función que recibe dos variables: un valor fijo y un valor variable, esta última se conoce como variable de control. La función debe devolver dos valores si la iteración tuvo éxito: el valor variable adaptado y el valor devuelto, que correspone con el elmento. Si no existieran más elementos a iterar la función debe devolver `nil`.

Veamos un ejemplo de iterador no genérico sin estado para entender mejor como funciona.

```
function suma (valorFijo, valorVariable)
    if (valorVariable < valorFijo)
    then
        valorVariable = valorVariable + 1
        return valorVariable, valorFijo + valorFijo * valorVariable
    end
end

for clave, valor in suma,10,0 do
    print(clave, valor)
end
```

En el ejemplo definimos una función `suma( valorFijo, valorVariable)` que acepta dos argumentos: `valorFijo` y `valorVariable`. Dentro de la función se verifica que `valorVariable` sea menor que `valorFijo` y si es verdadero, entonces incrementamos en una unidad `valorVariable`, devolviendo primero el nuevo valor de `valorVariable` y el resultado de sumar `valorFijo` a la multiplicación de `valorFijo` y `valorVariable`. En el caso que `valorVariable` sea mayor que `valorFijo`, entonces no devolvemos nada, que se interpreta como un valor `nil`.

El bucle lo hemos definido con una referencia a la función `suma` a la cual le pasamos como `valorFijo` el valor `10` y como valor inicial para `valorVariable` el valor `0`.

Veamos el resultado de ejecutar este ejemplo.

```
1       20
2       30
3       40
4       50
5       60
6       70
7       80
8       90
9       100
10      110
```

Como puedes ver el iterador a ido incrementado los valores internamente, pero no mantiene su valor (no tiene estado). A cada iteración el bucle nos envía el último valor que le devolvimos para `valorVariable`. Observa la tabla de valores a continuación para entender mejor este concepto.

Iteración	Valores Iniciales		Valor pasado por bucle		Valor interno función			Valor retorno función	
	valorFijo	valorVariable	valorFijo	valorVariable	valorFijo	valorVariable	Cálculo interno	elemento	valorVariable
1	10	0	10	0	10	1	10 + 10 * 1	20	1
2	10	0	10	1	10	2	10 + 10 * 2	30	2
3	10	0	10	2	10	3	10 + 10 * 3	40	3
4	10	0	10	3	10	4	10 + 10 * 4	50	4
5	10	0	10	4	10	5	10 + 10 * 5	60	5
6	10	0	10	5	10	6	10 + 10 * 6	70	6
7	10	0	10	6	10	7	10 + 10 * 7	80	7
8	10	0	10	7	10	8	10 + 10 * 8	90	8
9	10	0	10	8	10	9	10 + 10 * 9	100	9
10	10	0	10	9	10	10	10 + 10 * 10	110	10

De este modo podemos concluir que en los iteradores sin estado, el que mantiene el estado es el propio bucle, ya que, envía de vuelta a la función el valor variable que recibió de ésta en la última llamada, por lo tanto, `valorVariable` constituye la variable de control de estado.

La función `ipairs()` es un ejemplo de iterador sin estado.

Iteradores no genéricos con estado complejo

En algunos casos, la variable de control no es suficiente para mantener el control de la iteración. En estos casos se puede usar una tabla donde se almacenarán los valores necesarios para el control de la iteración. La tabla se pasa como valor fijo a la función, ya que la tabla en si misma no cambia durante la ejecución, pero sí lo hace el contenido de la misma. Esta técnica permite mantener un número arbitrario de parámetros y valores para el control de la iteración.

Dado que todos los valores necesarios para el iterador están contenidos en la tabla, el uso de la variable de control resulta innecesario, con lo cual se puede omitir. Vamos a ver un ejemplo que nos permita entender mejor el funcionamiento de este tipo de iteradores. Vamos a reescribir el ejemplo del apartado anterior usando una tabla como método de control.

```
tablaEstado = {}

tablaEstado.valorFijo = 10
tablaEstado.valorVariable = 0
tablaEstado.elemento = 0
tablaEstado.indice = 0

function suma (tblEstado)
    if (tblEstado.valorVariable < tblEstado.valorFijo)
    then
        tblEstado.valorVariable = tblEstado.valorVariable + 1
        tblEstado.elemento = tblEstado.valorFijo + tblEstado.valorFijo *
tblEstado.valorVariable
        tblEstado.indice = tblEstado.valorVariable
        return tblEstado
    end
end

for valor in suma,tablaEstado do
    print(valor.indice, valor.elemento)
end
```

Como puedes ver en el ejemplo, hemos definido una tabla hash (diccionario), con cuatro claves. Estas cuatro claves corresponden con las cuatro variables que habíamos usado en el ejemplo anterior. Dentro de la función suma hemos sustituido las variables originales por sus homólogas en la tabla. De este modo hemos hecho una refactorización del ejemplo anterior usando una tabla. En

la función ahora, solamente devolvemos un valor, la tabla, que contiene todo lo necesario. En el bucle observa que solamente pasamos la referencia a la tabla, es decir pasamos la variable `tablaEstado` solamente, ya que como hemos dicho, ella contiene toda la información necesaria para el control. Al ejecutar el código anterior obtenemos el siguiente resultado.

```
1       20
2       30
3       40
4       50
5       60
6       70
7       80
8       90
9       100
10      110
```

Como puedes ver, es exactamente el mismo resultado que en el ejemplo anterior. Pero aun queda un detalle, en el bucle, observa que hemos eliminado la variable `clave`, esto es, porque como estamos devolviendo un solo valor, ya no tenemos el valor del índice, como en el caso anterior. Ahora lo que hacemos es definirlo esplícitamente dentro de la tabla.

Revisión #1
Creado 21 febrero 2024 10:30:45 por Guillermo
Actualizado 5 abril 2024 06:53:25 por Guillermo