Язык программирования Lua

Функции и конструкции управления

гр. 5030102/20201
Смирнова А. П.
Грушин А. Д.

Управляющие конструкции. Блоки

Блок — это список выражений, которые выполняются последовательно:

```
block ::= {stat}
```

Lua допускает *пустые* выражения, что позволяет разделять выражения с помощью точки с запятой, начинать блок с точки с запятой или писать две точки с запятой подряд:

```
stat ::= ';'
```

Также блок может быть явно выделен для создания единственного выражения:

```
stat ::= do block end
```

Явные блоки полезны для контроля за областью видимости переменных, ведь они имеют собственную область видимости.

Управляющие конструкции. Блоки

Вызовы функций и присваивания могут начинаться с открывающейся скобки. Эта возможность ведет к неоднозначности в грамматике Lua. Рассмотрим следующий фрагмент:

```
a = b + c
(print or io.write)('done')
```

Грамматика может рассматривать это двумя путями:

```
a = b + c(print or io.write)('done')
a = b + c; (print or io.write)('done')
```

Текущий парсер всегда рассматривает такие конструкции первым путем, интерпретируя открывающуюся скобку, как начало аргументов для вызова.
Для избежания этой неоднозначности, лучше всегда ставить точку с запятой перед выражениями начинающимися со скобок:

```
;(print or io.write)('done')
```

Управляющие конструкции. Условные конструкции. if/elseif/else

Управляющая конструкция *if* имеет обычное значение и знакомый синтаксис:

```
stat ::= if exp then block {elseif exp
then block} [else block] end
```

Условное выражение может возвращать любое значение. *False* и *nil* рассматриваются как *false*. Все значения, отличные от *nil* и *false*, рассматриваются как *true*.

Управляющие конструкции. Циклы. while и repeat

Управляющие конструкции *while* и *repeat*:

```
stat ::= while exp do block end
stat ::= repeat block until exp
```

В цикле *while* условие *exp* проверяется перед выполнением блока. Выполнение продолжается, пока выражение истинно. Если изначально условие ложно, тело цикла не выполнится ни разу.

В цикле *repeat–until*, в отличие от *while*, тело цикла выполняется как минимум один раз, так как проверка условия идёт после блока. В *until exp* условие может ссылаться на локальные переменные, объявленные внутри цикла.

Управляющие конструкции. Циклы. for

Конструкция *for* имеет две формы: **цифровую** и **общую**.

Цифровой цикл for повторяет блок кода, пока управляющая переменная изменяется в арифметической прогрессии. Он имеет следующий синтаксис:

```
stat ::= for Name ‘=’ exp ‘,’ exp [‘,’ exp] do
block end
```

block повторяется для *name* начиная с первого значения *exp*, пока не достигнет значения второго *exp*, шагами равными третьему *exp*. Если значение шага отсутствует, тогда используется шаг 1. Для выхода из цикла for можно использовать *break* и *goto*.

Управляющие конструкции. Циклы. for

Общий for работает через функции, называемые *итераторами*. На каждой итерации, функция-итератор вызывается чтобы выдать новое значение, остановка происходит, когда новое значение равно *nil*. Общий цикл *for* имеет следующий синтаксис:

```
stat ::= for namelist in explist do block
end
namelist ::= Name {‘,’ Name}
```

Управляющие конструкции. goto/break/return

Оператор goto передает управление на метку. По синтаксическим причинам, метки в Lua тоже считаются выражениями:

```
stat ::= goto Name
stat ::= label
label ::= ‘::’ Name ‘::’
```

Выражение break завершает исполнение цикла while, repeat или for, пропуская оставшиеся команды цикла:

```
stat ::= break
```

break завершает самый внутренний цикл.

Выражение return используется для возврата значений из функции. Функции могут возвращать несколько значений, поэтому синтаксис для выражения return следующий:

```
stat ::= return [explist] [‘;’]
```

Функции. Определение и вызов

В Lua функции — это значения первого класса.

Синтаксис для определения функции:

```
functiondef ::= function funcbody
funcbody ::= ‘(’ [parlist] ‘)’ block end
```

Вызов функции в Lua имеет следующий синтаксис:

```
functioncall ::= prefixexp args
```

При вызове функции, сперва вычисляются prefixexp и args. Если значение prefixexp имеет тип function, то вызывается эта функция с данными аргументами.

Форма

```
functioncall ::= prefixexp ‘:’ Name args
```

может быть использована для вызова "методов".

Функции.
Входные данные

Передача аргументов всегда происходит **по значению**:

```
function addOne(x)
    return x + 1
end

local a = 10
print(addOne(a)) -- 11
```

Но таблицы передаются **как ссылки**:

```
function change(t)
    t.value = 42
end
local obj = { value = 0 }
change(obj)
print(obj.value) -- 42
```

Функции.
Выходные данные

Функции могут возвращать любое количество значений:

```
function divide(a, b)
  return math.floor(a/b), a % b
end
q, r = divide(10, 3) -- q = 3, r = 1
```

Если возвращается меньше значений, недостающие будут nil.

Функции. Рекурсия

Lua поддерживает как прямую, так и взаимную рекурсию (функции вызывают друг друга).

```
function factorial(n)
  if n == 0 then return 1 end
  return n * factorial(n - 1)
end

function even(n)
  if n == 0 then return true end
  return odd(n - 1)
end

function odd(n)
  if n == 0 then return false end
  return even(n - 1)
end
```

Функции. Замыкание

В Lua функции могут захватывать переменные из внешней области видимости. Когда функция возвращается наружу, эти переменные не исчезают, а продолжают жить внутри функции. Такой объект и называется замыкание.

```
function makeCounter()
  local count = 0           -- локальная переменная
(замыкается)
  return function()         -- внутренняя функция
    count = count + 1
    return count
  end
end

local c1 = makeCounter()
print(c1()) -- 1
print(c1()) -- 2
print(c1()) -- 3
```

Источники

При создании этой презентации использовалась информация из документации с официального сайта языка Lua (https://www.lua.org).