Язык программирования Lua

Стандартная библиотека ввода-вывода, сопрограммы и функциональное программирование

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

Стандартная библиотека ввода-вывода

Стандартный модуль io предоставляет доступ к потокам ввода-вывода и к файловым объектам. Предопределены три стандартных потока:
`io.stdin, io.stdout и io.stderr`

Файл открывается функцией
`io.open(path, mode)`

При ошибке возвращается `nil` и сообщение об ошибке. Режимы «r», «w», «a», «r+», «w+», «a+» задают чтение, запись, добавление и режим обновления; суффикс «b» включает бинарный режим.

Стандартная библиотека ввода-вывода

Чтение выполняется методами файлового объекта: `f:read("*l")` возвращает строку без завершающего перевода строки, `"*L"` — со строкой перевода, `"*a"` — весь оставшийся поток, `"*n"` — очередное число.

Для построчного чтения без явного открытия используется итератор `io.lines(path)`, который по очереди выдаёт строки файла и сам закроет дескриптор по завершении итерации.

```
local f = assert(io.open("input.txt", "r")) -- открываем файл для чтения
local all = f:read("*a") -- записываем весь поток
f:close()

for line in io.lines("input.txt") do
  print(line)
end
```

Стандартная библиотека ввода-вывода. Буферизация

Файловые объекты буферизуются.
Метод `setvbuf(kind[, size])` управляет буфером: `kind` принимает значения "no" (без буфера), "line" (построчно) и "full" (полная буферизация).

Буфер можно принудительно сбросить методом `flush()`.

Потоки по умолчанию — это `io.input()` и `io.output()`; их можно переназначить на открытые файлы с помощью `io.input(file)` и `io.output(file)`. Для диагностических сообщений удобно писать в `io.stderr`.

Стандартная библиотека ввода-вывода. Позиционирование

Метод `seek([whence[, offset]])` перемещает и/или возвращает текущую позицию в файле в байтах. Аргумент `whence` принимает значения "set" (от начала), "cur" (от текущей позиции) и "end" (от конца).

```
local function filesize(path)
  local f = assert(io.open(path, "rb"))
  local sz = f:seek("end") -- смещаем указатель в конец и возвращаем текущую позицию
  f:seek("set") -- смещаем в начало
  f:close()
  return sz
end
```

Стандартная библиотека ввода-вывода. Временные файлы и внешние процессы

Есть функция `io.tmpfile()`, которая создаёт временный файл, как правило удаляемый при закрытии.

Для взаимодействия с внешними программами используется `io.popen(cmd, mode)`. В режиме "r" можно прочитать стандартный вывод процесса, а в режиме "w" — записывать в его стандартный ввод.

```
local p = assert(io.popen("ls -1", "r")) -- запускаем процесс
local out = p:read("*a") -- считываем вывод процесса
p:close()
print(out)
```

Стандартная библиотека ввода-вывода. Ошибки и безопасное закрытие

Почти все функции ввода-вывода в случае ошибки возвращают nil и диагностическое сообщение.

Быструю проверку удобно делать через `assert`, а контролируемую — через `pcall/xpcall`.

Важно явно закрывать файлы, а не полагаться на сборщик мусора.

Сопрограммы

Сопрограмма в Lua — это функция с собственным стеком и точками добровольной передачи управления. Код выполняется в одном потоке, а переключение задач происходит только там, где программист явно вызывает `coroutine.yield()`.

Сопрограммы

Значения, которые передаются в yield внутри сопрограммы, становятся результатами вызова resume снаружи; и наоборот — аргументы, переданные в resume после первого запуска, видит код внутри как возвращённые yield значения.

```
local co = coroutine.create(function(a, b)
  local sum = a + b
  local x = coroutine.yield(sum) -- вернули наружу, при следующем resume получим x
  return x * 2 -- финальный результат попадёт наружу из resume
end)

local ok, v1 = coroutine.resume(co, 2, 3) -- ok=true, v1=5
local ok2, v2 = coroutine.resume(co, 10)  -- ok2=true, v2=20; статус корутины теперь "dead"
```

Сопрограммы

Статус сопрограммы можно проверять через `coroutine.status(co)`: значения — "suspended", "running", "dead". Ошибки из тела сопрограммы не «падают» в основной поток, а возвращаются из resume как ok=false и текст ошибки.

```
local co = coroutine.create(function()
  error("что-то пошло не так")
end)

local ok, err = coroutine.resume(co)
if not ok then io.stderr:write("корутина завершилась с
ошибкой: ", err, "
") end
```

Функциональное программирование

Функциональное программирование поддерживается в Lua.

Это парадигма, которая рассматривает вычисления как оценку математических функций, избегает изменения состояния и изменяемых данных.

В Lua функции — значения первого класса: их можно присваивать переменным, передавать в качестве аргументов другим функциям и возвращать из других функций. Также есть замыкания и оптимизация хвостовых вызовов.

Источники

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