Lua 文件 I/O Lua 调试(Debug)

Lua 错误处理

程序运行中错误处理是必要的，在我们进行文件操作，数据转移及web service 调用过程中都会出现不可预期的错误。如果不注重错误信息的处理，就会造成信息泄露，程序无法运行等情况。

任何程序语言中，都需要错误处理。错误类型有：

- 语法错误
- 运行错误

语法错误

语法错误通常是由于对程序的组件（如运算符、表达式）使用不当引起的。一个简单的实例如下：

```
-- test.lua 文件
a == 2
```

以上代码执行结果为：

```
lua: test.lua:2: syntax error near '=='
```

正如你所看到的，以上出现了语法错误，一个 "=" 号跟两个 "=" 号是有区别的。一个 "=" 是赋值表达式两个 "=" 是比较运算。

另外一个实例:

```
for a= 1,10
   print(a)
end
```

执行以上程序会出现如下错误：

```
lua: test2.lua:2: 'do' expected near 'print'
```

语法错误比程序运行错误更简单，运行错误无法定位具体错误，而语法错误我们可以很快的解决，如以上实例我们只要在for语句下添加 do 即可：

```
for a= 1,10
do
   print(a)
end
```

运行错误

运行错误是程序可以正常执行，但是会输出报错信息。如下实例由于参数输入错误，程序执行时报错：

```
function add(a,b)
   return a+b
end

add(10)
```

当我们编译运行以下代码时，编译是可以成功的，但在运行的时候会产生如下错误：

```
lua: test2.lua:2: attempt to perform arithmetic on local 'b' (a nil value)
stack traceback:
    test2.lua:2: in function 'add'
    test2.lua:5: in main chunk
    [C]: ?
```

以下报错信息是由于程序缺少 b 参数引起的。

错误处理

我们可以使用两个函数：assert 和 error 来处理错误。实例如下：

```
local function add(a,b)
   assert(type(a) == "number", "a 不是一个数字")
   assert(type(b) == "number", "b 不是一个数字")
   return a+b
end
add(10)
```

执行以上程序会出现如下错误：

```
lua: test.lua:3: b 不是一个数字
stack traceback:
    [C]: in function 'assert'
    test.lua:3: in local 'add'
    test.lua:6: in main chunk
    [C]: in ?
```

实例中assert首先检查第一个参数，若没问题，assert不做任何事情；否则，assert以第二个参数作为错误信息抛出。

error函数

语法格式：

```
error (message [, level])
```

功能：终止正在执行的函数，并返回message的内容作为错误信息(error函数永远都不会返回)

通常情况下，error会附加一些错误位置的信息到message头部。

Level参数指示获得错误的位置:

- Level=1[默认]：为调用error位置(文件+行号)
- Level=2：指出哪个调用error的函数的函数
- Level=0:不添加错误位置信息

---

pcall 和 xpcall、debug

Lua中处理错误，可以使用函数pcall（protected call）来包装需要执行的代码。

pcall接收一个函数和要传递给后者的参数，并执行，执行结果：有错误、无错误；返回值true或者或false, errorinfo。

语法格式如下

```
if pcall(function_name, ….) then
-- 没有错误
else
-- 一些错误
end
```

简单实例：

```
> =pcall(function(i) print(i) end, 33)
33
true

> =pcall(function(i) print(i) error('error..') end, 33)
33
false        stdin:1: error..
```

```
> function f() return false,2 end
> if f() then print '1' else print '0' end
0
```

pcall以一种"保护模式"来调用第一个参数，因此pcall可以捕获函数执行中的任何错误。

通常在错误发生时，希望落得更多的调试信息，而不只是发生错误的位置。但pcall返回时，它已经销毁了调用栈的部分内容。

Lua提供了xpcall函数，xpcall接收第二个参数——一个错误处理函数，当错误发生时，Lua会在调用栈展开（unwind）前调用错误处理函数，于是就可以在这个函数中使用debug库来获取关于错误的额外信息了。

debug库提供了两个通用的错误处理函数:

- debug.debug：提供一个Lua提示符，让用户来检查错误的原因
- debug.traceback：根据调用栈来构建一个扩展的错误消息

```
>=xpcall(function(i) print(i) error('error..') end, function() print(debug.traceback()) end, 33)
33
stack traceback:
stdin:1: in function <stdin:1>
[C]: in function 'error'
stdin:1: in function <stdin:1>
[C]: in function 'xpcall'
stdin:1: in main chunk
[C]: in ?
false        nil
```

xpcall 使用实例 2:

```
function myfunction ()
   n = n/nil
end

function myerrorhandler( err )
   print( "ERROR:", err )
end

status = xpcall( myfunction, myerrorhandler )
print( status)
```

执行以上程序会出现如下错误：

```
ERROR:    test2.lua:2: attempt to perform arithmetic on global 'n' (a nil value)
false
```

← Lua 文件 I/O

Lua 调试(Debug) →

点我分享笔记