

Lua 教程
Lua 教程
Lua 环境安装
Lua 基本语法
Lua 数据类型
Lua 变量
Lua 循环
Lua 流程控制
Lua 函数
Lua 运算符
Lua 字符串
Lua 数组
Lua 迭代器
Lua table(表)
Lua 模块与包
Lua 元表(Metatable)
Lua 协同程序(coroutine)
Lua 文件 I/O
Lua 错误处理
Lua 调试(Debug)
Lua 垃圾回收
Lua 面向对象
Lua 数据库访问
Lua 5.3 参考手册

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：

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：

实例

```
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:
    [C]: in function 'add'
    test2.lua:2: 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：

实例

```
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]: ?
```

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：

```
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..
```

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，希望获得更多的调试信息，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 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：

```
实例
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
```

分类导航

HTML / CSS

JavaScript

服务端

数据库

移动端

XML 教程

ASP.NET

Web Service

开发工具

网站建设

Advertisement

