Lua 错误处理

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+bend
add(10)
```

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个后者的参数，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)33true

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

> function f() return false,2 end> if f() then print '1' else print '0' end0
```

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。**即 pcall 以保护模式调用函数，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 会在调用栈展看（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。**即 xpcall 为 pcall 的加强版，允许调用栈展开前调用错误处理函数。**

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/nilend

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
```