第16章 编译，执行和错误

Lua语言总是在运行代码前先预编译源码为中间代码。

解释型语言的区分并不在于源码是否被编译，而在于是否又能（且轻易地）执行动态生成的代码。

16.1 编译

dofile函数是运行Lua代码段的主要方式之一。

dofile是一个辅助函数，函数loadfile才完成了真正的核心工作。

与函数dofile类似，函数loadfile也是从文件中加载Lua代码段，但它不会运行代码，而只是编译代码，然后将编译后的代码段作为一个函数返回。

loadfile只返回错误码而不抛出异常。

可以认为dofile就是

function dofile(filename) local f = assert(loadfile(filename)) return f() end

load与loadfile类似，不同之处在于该函数从一个字符串或函数中读取代码段，而不是从文件中读取。

例如：f = load("i = i + 1") f()

load函数应该谨慎地使用，该函数开销较大并且可能会引起诡异的问题。找不到更简单的解决方式后再使用该函数。

load在编译时不涉及词法定界。

load最典型的用法是执行外部代码或动态生成的代码。

如果需要对表达式求值，那么可以在表达式前添加return。

load和loadfile从来不引发错误，当有错误发生时，它们会返回nil及错误信息。

16.2 预编译的代码

生成预编译文件（也称为二进制文件）的最简单的方式是，使用标准发行版中附带的luac程序。

luac -o prog.lc prog.lua

lua prog.lc

Lua中能够使用源码的地方都可以使用预编译代码。loadfile和load都可以接受预编译代码。

string.dump函数的参数是一个Lua函数，返回值是传入函数对应的字符串形式的预编译代码。

16.3 错误

由于Lua语言是一种经常被嵌入在应用程序中的扩展语言，所以当错误发生时并不能简单地崩溃或退出。

我们可以显示地通过调用函数error并传入一个错误信息作为参数来引发一个错误。

Lua语言提供了一个内建的函数assert来完成这类工作。

assert检查第一个参数是否为真，如果为真返回该参数，如果为假则引发一个错误，第二个参数是一个可选的错误信息。

当一个函数发现某种意外的情况发生时，在进行异常处理时可以采取两种基本方式：一种是返回错误代码，另一种是通过调用函数error引发一个错误。

容易避免的异常应该引发错误，否则应该返回错误码。

16.4 错误处理和异常

如果要在Lua代码中处理错误，那么应该使用函数pcall来封装代码。

pcall会以一种保护模式来调用它的第一个参数，以便捕获该函数执行中的错误。

如果发生错误，pcall返回false及错误信息。

16.5 错误信息和栈回溯