第7章 输入输出

7.1 简单I/O模型

对于文件操作来说，IO库提供了两种不同的模型。

简单模型虚拟了一个当前输入流和一个当前输出流，其I/O操作是通过这些流实现的。

IO库把当前输入流初始化为进程的标准输入（C语言中的stdin），将当前输出流初始化为进程的标准输出(C语言中的stdout)。

执行类似于io.read()可以从标准输入读取一行。

函数io.input和函数io.output可以用于改变当前输入输出流。

调用io.input(filename)会以制度模式打开指定文件，并将文件设置为当前输入流。

io.write可以读取任意数量的字符串（或者数字）并将其写入当前输出流。

由于该函数可以使用多个参数，因此应该避免使用io.write(a..b..c)，应该调用io.write(a,b,c)

作为原则，应该只在“用后即弃”的代码或调试代码中使用函数print，当需要完全控制输出时，应该使用io.write。

io.read可以从当前输入流中读取字符串，其参数决定了要读取的数据：

a 读取整个文件

l 读取下一行（丢弃换行符）

L 读取下一行（保留换行符）

n 读取一个数值

num 以字符串读取num个字符

如果要逐行迭代一个文件，那么使用io.lines迭代器更简单。

调用函数read时可以指定多个选项，函数会根据每个参数返回相应的结果。

local n1, n2, n3 = io.read( "n", "n", "n")

7.2 完整I/O模型

简单IO模型对于诸如同时读写多个文件等更高级的文件操作来说就不够了，需要用到完整IO模型。

可以使用io.open来打开一个文件，该函数有两个参数，一个参数是待打开的文件名，另一个参数是一个模式字符串。

r表示只读，w表示只写，a表示追加，b表示打开二进制文件。

io.open返回对应文件的流，当发生错误时返回nil同时返回一套错误信息及一个系统相关的错误码。

检查错误的一种典型方法是使用函数assert:local f = assert(io.open(filename,mode))

打开文件后，可以使用方法read和write从流中读取和向流中写入。需要使用冒号运算符将它们当作流对象的方法来调用。f:read("a")

函数io.input和io.output允许混用完整IO模型和简单IO模型。调用无参数的io.input()可以获得当前输入流，调用io.input(handle)可以设置当前输入流。

io.read(args)实际上是io.input:read(args)的简写。

7.3 其他文件操作

函数flush将所有的缓冲数据写入文件。

函数setvbuf用于设置流的缓冲模式。

函数seek用来获取和设置文件的当前位置。

file:seek()会返回当前位置而不改变当前位置，file:seek("set")会将位置重置到文件开头并返回0，file:seek("end")会将当前位置重置到文件结尾并返回文件的大小。

os.rename用于文件重命名，os.remove用于移除文件。

7.4 其他系统调用

os.exit用于终止程序的执行。

os.getenv用于获取某个环境变量。

7.4.1 运行系统命令

os.execute用于运行系统命令，它等价于C语言中的System。

例子：os.execute("mkdir " .. dirname)

io.popen同函数os.execute一样，该函数运行一条系统命令，但该函数还可以重定向命令的输入/输出。

如果要使用操作系统的其他扩展功能，最好的选择是使用第三方库，比如用于基本目录操作和文件属性操作的LuaFileSystem。