使用Lua编写可嵌入式脚本之三 PDF转换可能丢失图片或格式，建议阅读原文 https://www.100test.com/kao_ti2020/144/2021_2022__E4_BD_BF_E7_94_A8Lua_E7_c103_144248.htm 清单 1. 体验 Lua 表$ lua> -- create an empty table and add some elements> t1 = {}> t1[1] = "moustache"> t1[2] = 3> t1["brothers"] = true> -- more commonly, create the table and define elements> all at once> t2 = {[1] = "groucho", [3] = "chico", [5] = "harpo"}> t3 = {[t1[1]] = t2[1], accent = t2[3], horn = t2[5]}> t4 = {}> t4[t3] = "the marx brothers"> t5 = {characters = t2, marks = t3}> t6 = {["a night at the opera"] = "classic"}> -- make a reference and a string> i = t3> s = "a night at the opera"> -- indices can be any Lua value> print(t1[1], t4[t3], t6[s])moustache the marx brothers classic> -- the phrase table.string is the same as table["string"]> print(t3.horn, t3["horn"])harpo harpo> -- indices can also be "multi-dimensional"> print (t5["marks"]["horn"], t5.marks.horn)harpo harpo> -- i points to the same table as t3> = t4[i]the marx brothers> -- non-existent indices return nil values> print(t1[2], t2[2], t5.films)nil nil nil> -- even a function can be a key > t = {}> function t.add(i,j)>> return(i j)>> end> print(t.add(1,2))3> print(t[add](1,2))3> -- and another variation of a function as a key > t = {}> function v(x)>> print(x)>> end> t[v] = "The Big Store"> for key,value in t do key(value) endThe Big Store 正如我们可能期望的一样，Lua 还提供了很多迭代器函数来对表进行处理。全局变量 table 提供了这些函数（是的，Lua 包就是表）。有些函数，例如 table.foreachi()，会期望一个从

1（数字 1）开始的连续整数范围：> table.foreachi(t1, print)1 moustache2 3 另外一些函数，例如 table.foreach()，会对整个表进行迭代：> table.foreach(t2,print)1 groucho3 chico5 harpo> table.foreach(t1,print)1 moustache2 3brothers true 尽管有些迭代器对整数索引进行了优化，但是所有迭代器都只简单地处理 (key, value) 对。 现在我们可以创建一个表 t，其元素是 {2, 4, 6, language="Lua", version="5", 8, 10, 12, web="www.lua.org"}，然后运行 table.foreach(t, print) 和 table.foreachi(t, print)。 用户数据 由于 Lua 是为了嵌入到使用另外一种语言（例如 C 或 C ）编写的宿主应用程序中，并与宿主应用程序协同工作，因此数据可以在 C 环境和 Lua 之间进行共享。正如 Lua 5.0 Reference Manual 所说，userdata 类型允许我们在 Lua 变量中保存任意的 C 数据。我们可以认为 userdata 就是一个字节数组 字节可以表示指针、结构或宿主应用程序中的文件。 用户数据的内容源自于 C，因此在 Lua 中不能对其进行修改。当然，由于用户数据源自于 C，因此在 Lua 中也没有对用户数据预定义操作。不过我们可以使用另外一种 Lua 机制来创建对 userdata 进行处理的操作，这种机制称为元表（metatable）。 元表 由于表和用户数据都非常灵活，因此 Lua 允许我们重载这两种类型的数据的操作（不能重载其他 6 种类型）。元表 是一个（普通的）Lua 表，它将标准操作映射成我们提供的函数。元表的键值称为事件；值（换而言之就是函数）称为元方法。函数 setmetatable() 和 getmetatable() 分别对对象的元表进行修改和查询。每个表和 userdada 对象都可以具有自己的元表。例如，添加操作对应的事件是 __add。我们可以推断这段代码所做的事情么？-- Overload the add operation-- to do string

concatenation--mt = {}function String(string) return setmetatable({value = string or }, mt)end-- The first operand is a String table-- The second operand is a string-- .. is the Lua concatenate operator--function mt.__add(a, b) return String(a.value..b)ends = String(Hello)print((s There World!).value)
这段代码会产生下面的文本： Hello There World! 100Test 下载频道开通，各类考试题目直接下载。详细请访问 www.100test.com