使用Lua编写可嵌入式脚本之四 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_144252.htm 函数 String() 接收一个字符串 string，将其封装到一个表（{value = s or }）中，并将元表 mt 赋值给这个表。函数 mt.__add() 是一个元方法，它将字符串 b 添加到在 a.value 中找到的字符串后面 b 次。这行代码 print((s There World!).value) 调用这个元方法两次。 __index 是另外一个事件。__index 的元方法每当表中不存在键值时就会被调用。下面是一个例子，它记住 (memoize) 函数的值：--code courtesy of Rici Lake, rici@ricilake.netfunction Memoize(func, t) return setmetatable(t or {}, {__index = function(t, k) local v = func(k). t[k] = v. return v. end })endCOLORS = {"red", "blue", "green", "yellow", "black"}color = Memoize(function(node) return COLORS[math.random(1, table.getn(COLORS))] end) 将这段代码放到 Lua 解释器中，然后输入 print(color[1], color[2], color[1])。您将会看到类似于 blue black blue 的内容。 这段代码接收一个键值 node，查找 node 指定的颜色。如果这种颜色不存在，代码就会给 node 赋一个新的随机选择的颜色。否则，就返回赋给 node 的颜色。在前一种情况中，__index 元方法被执行一次以分配一个颜色。后一种情况比较简单，所执行的是快速散列查找。 Lua 语言提供了很多其他功能强大的特性，所有这些特性都有很好的文档进行介绍。在碰到问题或希望与专家进行交谈时，请连接 Lua Users Chat Room IRC Channel（请参见 参考资料）获得非常热心的支持。 嵌入和扩展 除了语法简单并且具有功能强大的表结构之外，Lua 的

强大功能使其可以与宿主语言混合使用。由于 Lua 与宿主语言的关系非常密切，因此 Lua 脚本可以对宿主语言的功能进行扩充。但是这种融合是双赢的：宿主语言同时也可以对 Lua 进行扩充。举例来说，C 函数可以调用 Lua 函数，反之亦然。 Lua 与宿主语言之间的这种共生关系的核心是宿主语言是一个虚拟堆栈。虚拟堆栈与实际堆栈类似，是一种后进先出（LIFO）的数据结构，可以用来临时存储函数参数和函数结果。要从 Lua 中调用宿主语言的函数（反之亦然），调用者会将一些值压入堆栈中，并调用目标函数；被调用的函数会弹出这些参数（当然要对类型和每个参数的值进行验证），对数据进行处理，然后将结果放入堆栈中。当控制返回给调用程序时，调用程序就可以从堆栈中提取出返回值。 实际上在 Lua 中使用的所有的 C 应用程序编程接口（API）都是通过堆栈来进行操作的。堆栈可以保存 Lua 的值，不过值的类型必须是调用程序和被调用者都知道的，特别是向堆栈中压入的值和从堆栈中弹出的值更是如此（例如 lua_pushnil() 和 lua_pushnumber()。 清单 2 给出了一个简单的 C 程序（节选自 参考资料 中 Programming in Lua 一书的第 24 章），它实现了一个很小但却功能完善的 Lua 解释器。 清单 2. 一个简单的 Lua 解释器 1 #include 2 #include 3 #include 4 #include 5 6 int main (void) { 7 char buff[256]. 8 int error. 9 lua_State *L = lua_open(). /* opens Lua */10 luaopen_base(L). /* opens the basic library */11 luaopen_table(L). /* opens the table library */12 luaopen_io(L). /* opens the I/O library */13 luaopen_string(L). /* opens the string lib. */14 luaopen_math(L). /* opens the math lib. */1516 while (fgets(buff, sizeof(buff), stdin) != NULL) {17 error =

luaL_loadbuffer(L, buff, strlen(buff), "line") ||18 lua_pcall(L, 0, 0, 0).19 if (error) {20 fprintf(stderr, "%s", lua_tostring(L, -1)).21 lua_pop(L, 1). /* pop error message from the stack */22 }23 }2425 lua_close(L).26 return 0.27 } 第 2 行到第 4 行包括了 Lua 的标准函数，几个在所有 Lua 库中都会使用的方便函数以及用来打开库的函数。第 9 行创建了一个 Lua 状态。所有的状态最初都是空的；我们可以使用 luaopen_...() 将函数库添加到状态中，如第 10 行到第 14 行所示。 第 17 行和 luaL_loadbuffer() 会从 stdin 中以块的形式接收输入，并对其进行编译，然后将其放入虚拟堆栈中。第 18 行从堆栈中弹出数据并执行之。如果在执行时出现了错误，就向堆栈中压入一个 Lua 字符串。第 20 行访问栈顶（栈顶的索引为 -1）作为 Lua 字符串，打印消息，然后从堆栈中删除该值。 使用 C API，我们的应用程序也可以进入 Lua 状态来提取信息。下面的代码片段从 Lua 状态中提取两个全局变量：..if (luaL_loadfile(L, filename) || lua_pcall(L, 0, 0, 0)) error(L, "cannot run configuration file: %s", lua_tostring(L, -1)).lua_getglobal(L, "width").lua_getglobal(L, "height")...width = (int) lua_tonumber(L, -2).height = (int) lua_tonumber(L, -1)... 请再次注意传输是通过堆栈进行的。从 C 中调用任何 Lua 函数与这段代码类似：使用 lua_getglobal() 来获得函数，将参数压入堆栈，调用 lua_pcall()，然后处理结果。如果 Lua 函数返回 n 个值，那么第一个值的位置在堆栈的 -n 处，最后一个值在堆栈中的位置是 -1。 反之，在 Lua 中调用 C 函数也与之类似。如果您的操作系统支持动态加载，那么 Lua 可以根据需要来动态加载并调用函数。（在必须使用静态加载的操作系统中，可以对 Lua 引擎进行扩充，此时

调用 C 函数时需要重新编译 Lua。） 结束语 Lua 是一种学习起来容易得难以置信的语言，但是它简单的语法却掩饰不了其强大的功能：这种语言支持对象（这与 Perl 类似），元表使表类型具有相当程度的可伸展性，C API 允许我们在脚本和宿主语言之间进行更好的集成和扩充。Lua 可以在 C、C 、C#、Java&#8482. 和 Python 语言中使用。 在创建另外一个配置文件或资源格式（以及相应的处理程序）之前，请尝试一下 Lua。Lua 语言及其社区非常健壮，具有创新精神，随时准备好提供帮助。 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