Lua 调试(Debug) Lua 面向对象

Lua 垃圾回收

Lua 采用了自动内存管理。 这意味着你不用操心新创建的对象需要的内存如何分配出来， 也不用考虑在对象不再被使用后怎样释放它们所占用的内存。

Lua 运行了一个**垃圾收集器**来收集所有**死对象**（即在 Lua 中不可能再访问到的对象）来完成自动内存管理的工作。 Lua 中所有用到的内存，如：字符串、表、用户数据、函数、线程、 内部结构等，都服从自动管理。

Lua 实现了一个增量标记-扫描收集器。 它使用这两个数字来控制垃圾收集循环： 垃圾收集器间歇率和垃圾收集器步进倍率。 这两个数字都使用百分数为单位（例如：值 100 在内部表示 1 ）。

垃圾收集器间歇率控制着收集器需要在开启新的循环前要等待多久。 增大这个值会减少收集器的积极性。 当这个值比 100 小的时候，收集器在开启新的循环前不会有等待。 设置这个值为 200 就会让收集器等到总内存使用量达到 之前的两倍时才开始新的循环。

垃圾收集器步进倍率控制着收集器运作速度相对于内存分配速度的倍率。 增大这个值不仅会让收集器更加积极，还会增加每个增量步骤的长度。 不要把这个值设得小于 100 ， 那样的话收集器就工作的太慢了以至于永远都干不完一个循环。 默认值是 200 ，这表示收集器以内存分配的"两倍"速工作。

如果你把步进倍率设为一个非常大的数字（比你的程序可能用到的字节数还大 10% ）， 收集器的行为就像一个 stop-the-world 收集器。 接着你若把间歇率设为 200 ， 收集器的行为就和过去的 Lua 版本一样了： 每次 Lua 使用的内存翻倍时，就做一次完整的收集。

垃圾回收器函数

Lua 提供了以下函数**collectgarbage ([opt [, arg]])**用来控制自动内存管理:

- **collectgarbage("collect"):** 做一次完整的垃圾收集循环。通过参数 opt 它提供了一组不同的功能：
- **collectgarbage("count"):** 以 K 字节数为单位返回 Lua 使用的总内存数。 这个值有小数部分，所以只需要乘上 1024 就能得到 Lua 使用的准确字节数（除非溢出）。
- **collectgarbage("restart"):** 重启垃圾收集器的自动运行。
- **collectgarbage("setpause"):** 将 arg 设为收集器的 间歇率 （参见 §2.5）。 返回 间歇率 的前一个值。
- **collectgarbage("setstepmul"):** 返回 步进倍率 的前一个值。
- **collectgarbage("step"):** 单步运行垃圾收集器。 步长"大小"由 arg 控制。 传入 0 时，收集器步进（不可分割的）一步。 传入非 0 值， 收集器收集相当于 Lua 分配这些多（K 字节）内存的工作。 如果收集器结束一个循环将返回 true 。
- **collectgarbage("stop"):** 停止垃圾收集器的运行。 在调用重启前，收集器只会因显式的调用运行。

以下演示了一个简单的垃圾回收实例:

```
mytable = {"apple", "orange", "banana"}

print(collectgarbage("count"))

mytable = nil

print(collectgarbage("count"))

print(collectgarbage("collect"))

print(collectgarbage("count"))
```

执行以上程序，输出结果如下(注意内存使用的变化)：

```
20.9560546875
20.9853515625
0
19.4111328125
```

Lua 调试(Debug) Lua 面向对象