**11-Lua**

**垃圾收集原理概述**

**Lua** 使用的是**标记-清除（Mark-and-Sweep）算法**进行垃圾收集。这个过程分为两个阶段：

**标记（Mark）阶段**：Lua 遍历所有活动对象（即那些仍然可以从根集合直接或间接访问的对象），并将它们标记为活动的。

**清除（Sweep）阶段**：Lua 移除所有未被标记的对象，释放它们占用的内存。

**内存分配**

Lua 使用 malloc 和 free（C语言标准库函数）进行内存分配和释放。

**内存泄漏**

尽管 Lua 提供了自动垃圾收集，但内存泄露仍然可能发生，尤其是在使用复杂的数据结构和循环引用时。程序员需要注意正确管理对象的生命周期，使用弱引用表来帮助打破潜在的循环引用。

**弱引用表**

弱引用表，简称弱表，是一种特殊类型的表，其键值对中的键（key）和/或值（value）可以是弱引用。这意味着，如果一个对象只被弱表所引用，那么它不会被视为活跃对象，因此可以被垃圾收集器回收。

弱表的行为通过设置其元表（metatable）中的 \_\_mode 字段来控制。\_\_mode 字段可以有以下设置：

"k"：如果设置为 "k"，则表中的键是弱引用。这意味着，如果一个对象只作为键存在于表中，它可以被回收。

"v"：如果设置为 "v"，则表中的值是弱引用。这意味着，如果一个对象只作为值存在于表中，它可以被回收。

"kv" 或 "vk"：在这种情况下，键和值都是弱引用。

弱表的使用示例

如果一个对象只被弱表引用，一旦程序的其他部分不再引用该对象，它就会成为垃圾收集的候选对象。

这种特性使弱表成为实现自动缓存机制的理想选择。在缓存场景中，您可能希望暂时存储一些数据以提高效率，但如果这些数据不再被需要，它们应该自动释放，以避免不必要地占用内存。

假设您正在开发一个应用程序，需要频繁地对某些对象进行昂贵的计算。为了提高效率，您决定缓存这些计算结果。但是，您不希望缓存永久占用内存，特别是当原始对象不再需要时。