<https://blog.codingnow.com/2011/03/lua_gc_1.html>

<http://www.360doc.com/content/14/0104/08/9200790_342485388.shtml>

<https://www.cnblogs.com/zblade/p/8824376.html>

<https://blog.csdn.net/BigBrick/article/details/85317491#3Lua_17>（好文章）

**三色标记清除法**

主要缺点：效率不高，内存碎片

GC目前的问题是会暂停、阻碍代码运行，即stop the world，增量式GC处理的就是这个问题，将GC变得可分阶段进行。三色标记法是一个逻辑上的抽象，将对象分成：白-未搜索，灰-正搜索，黑-已搜索

Mark-sweep按增量来排，可以分成三个阶段：根查找、标记、清除

我们先从最常见的table分析，看看lua gc的启动，这里的核心方法luaC\_checkGC（lgc.h）这个宏，因为gc的启动一般来说就是通过这个宏开始的，在lapi.c中，每创建table和其他对象时候都会调用这个函数

#define luaC\_checkGC(L) { \

condhardstacktests(luaD\_reallocstack(L, L->stacksize - EXTRA\_STACK - 1)); \

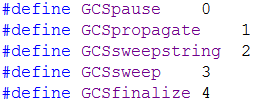
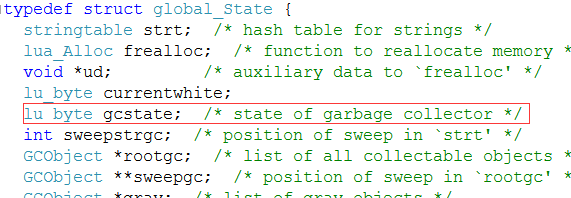
if (G(L)->totalbytes >= G(L)->GCthreshold) \

luaC\_step(L); }

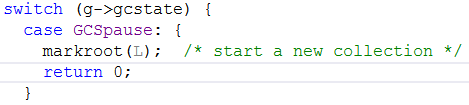
这里我们可以看到它会比较当前分配的字节数与GCthreshold进行比较，如果大于这个值才会进行step，而GCthreshold就是一个阈值

从lua 5.1开始，GC的实现改为分步的，虽然依旧是stop the world，但是每个步骤都可以分阶段执行，这样，每次停顿的时间较小，随之，这部分的代码也相对复杂了。分步执行最关键的问题是需要解决在GC的步骤之间，如果数据关联的状态发生了变化，如何保证GC的正确性，HC的分部执行对于于一次执行完，总的时间开销的差别并不是零代价的，只是在实现上要尽量让额外增加的代价较小。

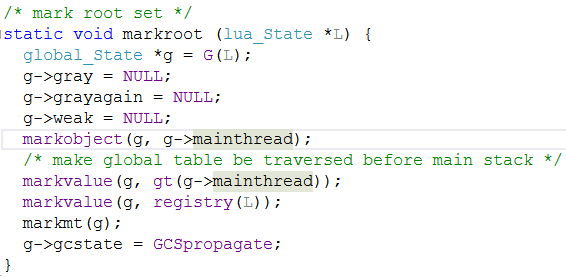
Lua中的GC分为五个大的阶段，GC处于哪个节点（代码中被称为状态），依据的是global\_State中的gcstate域，状态以宏的形式在lgc.h中定义：



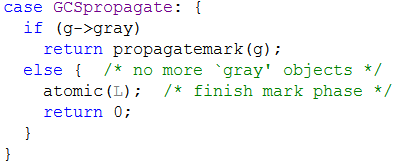
状态值的大小也暗示着它们的执行顺序，需要注意的是，GC的执行过程并非每个步骤都拥塞在一个状态上。GCPause阶段是每个GC流程的起始步骤，只是标记系统的根节点，键lgc.c



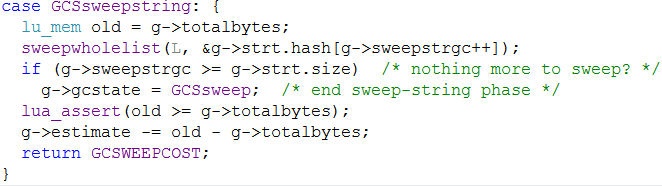
Markroot这个函数所做的事，就是标记主线程对象，标记主线程的全局表、注册表以及为全局类型注册的元表



GCPause阶段执行完，立刻就将状态切换到GCSpropagate，这是一个标记流程，这个流程会分步完成。当检测到尚有对象待标记时，迭代标记（反复调用propagatemark），最终会有一个标记过程不可被打断，这些操作放在一个叫做atomic的函数中执行。



这里需要理解的是gray域，它指GCObject中的灰色节点链，即处于白色与黑色之间的状态。我们知道，string在lua中是单独管理的，所以也需要单独清除，GCWsweepstring阶段干的就是这个事情，GCWsweepstring中，每个步骤（step）清理hash表的一列



这里可以看到estimate和totalbytes两个域，从名字可以知道它们分别表示了lua vm占用的内存字节数以及实际分配的字节数。

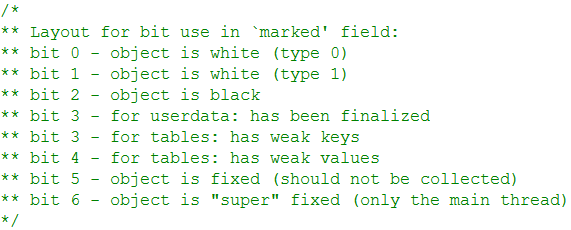
接下来就是对所有未标记的其他GCObject做清理工作了，即GCSsweep阶段，它和上面的GCSsweepstring类似

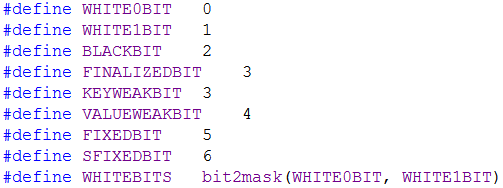
最后是GCSfinalize阶段，如果在前面的阶段发现了需要低啊用gc原方法的userdata对象，将在这个阶段逐个调用，做这件事情的函数是GCTM，前面已经谈到过，所有拥有gc元方法的userdata对象以及其关联的数据，实际上都不会再之前的清除阶段被清除（由于单独做过标记）所有的元方法调用都是安全的，而它们的实际清除，则需要等到下一次GC流程，或是在lua\_close被调用时清除。

Lua\_close并不做完整的gc工作，只是简单的处理所有userdata的gc方法，以及释放所有用到的内存，它是相对廉价的。

接下来看一些概念：

简单的说，lua认为每个GCObject（需要被GC收集器管理的对象）都有一个颜色，一开始，所有节点都是白色的，新创建出来的节点也被默认设置为白色。在标记节点，可见的节点，逐个被这只给黑色，有些节点比较复杂，它会关联别的节点，在没有处理完所有关联节点前，lua认为它是灰色的。节点的颜色被储存在GCObject的CommonHeader里，放在marked域中，为了节约内存，是以位形式存放，marked是一个单字节量，总共可以储存8个标记，而lua 5.1.4用到了7个标记为，在lgc.h中





白色和黑色是分别标记的，当一个对象非白非黑时，就认为它是灰色的。为什么会有两个白色标记位？这是lua采用的一个小技巧，在GC的标记流程结束，但清除流程尚未做完前。一旦对象间的关系发生变化，比如新增加了一个对象，这些对象的生命周期是不可预料的，最安全的方法是把它们标记为不可清除，但我们又不能直接把对象设置为黑色，因为清理过程结束后，需要把所有对象设置回白色，方便下次清理，lua实际上是单遍扫描，处理完一个节点就重置一个节点的颜色，简单把新创建出来的对象设置为黑色，有可能导致它在GC流程结束后再也没有机会变为白色。