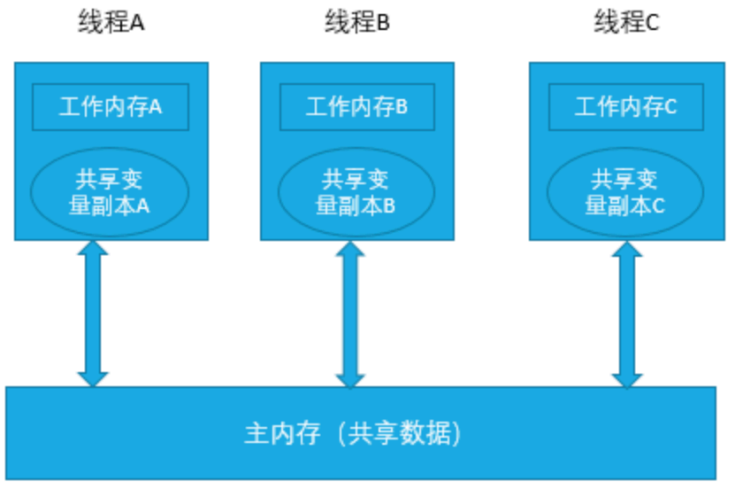
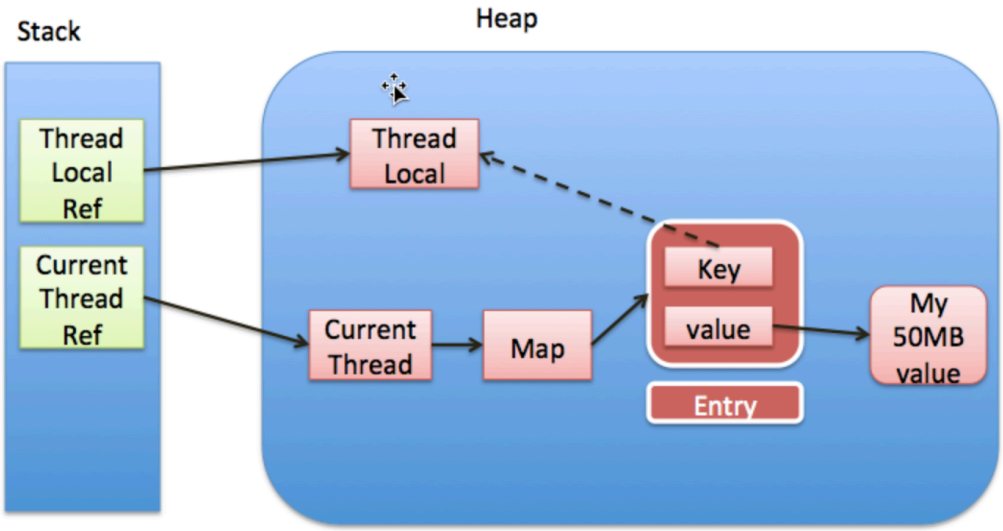
https://www.cnblogs.com/aspirant/p/8991010.html

ThreadLocal与多**线程**共享数据**安全**有关吗?
没有关系；
Java的ThreadLocal不是设计用来解决多线程安全问题的，事实证明也解决不了，共享变量a还是会被随意更改。ThreadLocal无能为力。所以，一般用ThreadLocal都不会将一个共享变量放到线程的ThreadLocal中。一般来讲，存放到ThreadLocal中的变量都是当前线程
本身就独一无二的一个变量。其他线程本身就不能访问，存到ThreadLocal中只是为了方便在程序中同一个线程之间传递这个变量。
所以，ThreadLocal和解决线程安全没有关系。



在threadlocal的生命周期中,都存在这些引用. 看下图: 实线代表强引用,虚线代表弱引用.



小马似乎讲错了，我认为以下是正确的：

考虑这样的情况：

C c = new C(b);

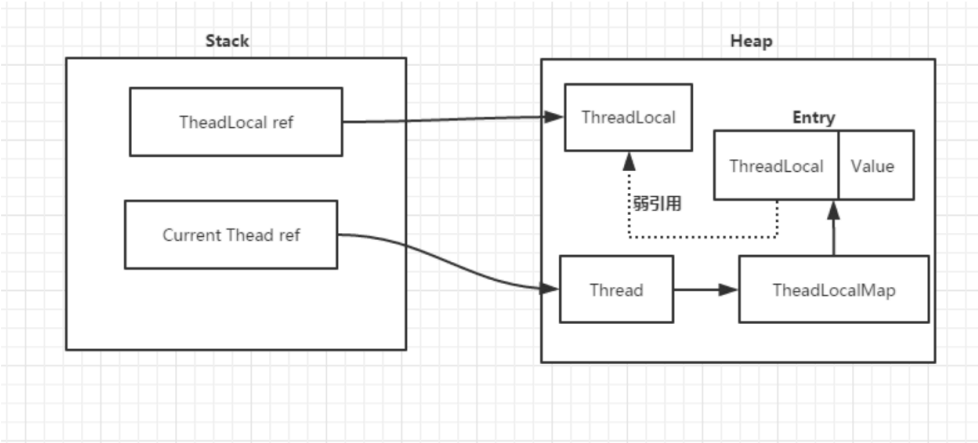
b = null;

考虑下GC的情况。要知道b被置为null，那么是否意味着一段时间后GC工作可以回收b所分配的内存空间呢？答案是否定的，因为即便b被置为null，但是c仍然持有对b的引用，而且还是强引用，所以GC不会回收b原先所分配的空间！既不能回收利用，又不能使用，这就造成了**内存泄露**。

那么如何处理呢?

可以c = null;也可以使用弱引用！（WeakReference w = new WeakReference(b);）

分析到这里，我们可以得到：



内存结构图

这里我们思考一个问题：ThreadLocal使用到了弱引用，是否意味着不会存在内存泄露呢?

首先来说，如果把ThreadLocal置为null，那么意味着Heap中的ThreadLocal实例不在有强引用指向，只有弱引用存在，因此GC是可以回收这部分空间的，也就是key是可以回收的。但是value却存在一条从Current Thread过来的强引用链。因此只有当Current Thread销毁时，value才能得到释放。

因此，只要这个线程对象被gc回收，就不会出现内存泄露，但在threadLocal设为null和线程结束这段时间内不会被回收的，就发生了我们认为的内存泄露。最要命的是线程对象不被回收的情况，比如使用线程池的时候，线程结束是不会销毁的，再次使用的，就可能出现内存泄露。

那么如何有效的避免呢?

事实上，在ThreadLocalMap中的set/getEntry方法中，会对key为null（也即是ThreadLocal为null）进行判断，如果为null的话，那么是会对value置为null的。我们也可以通过调用ThreadLocal的remove方法进行释放！