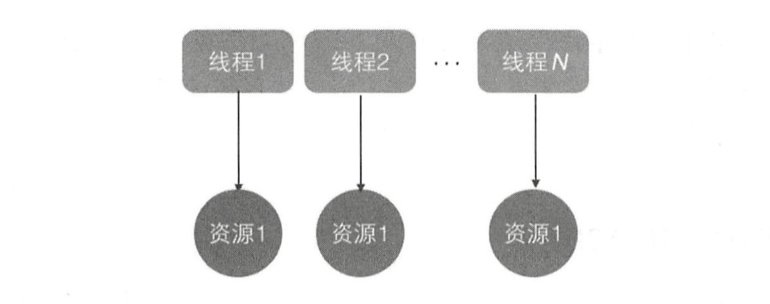
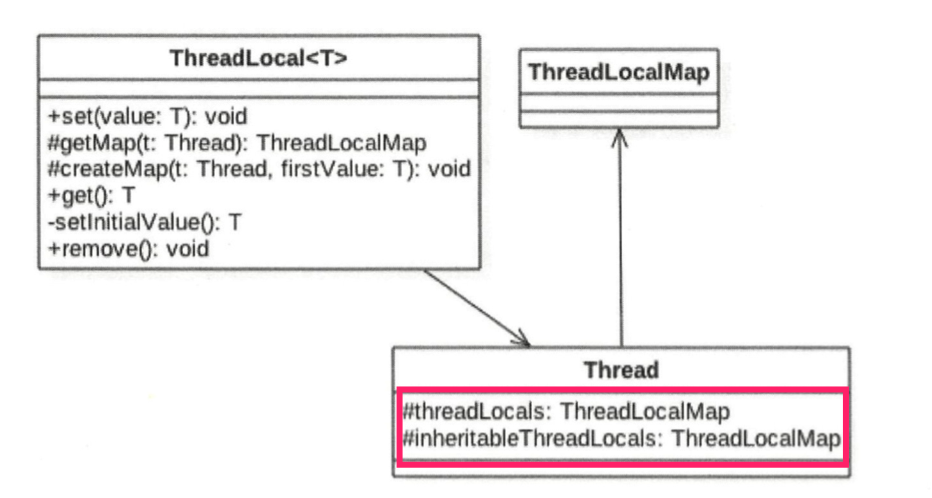
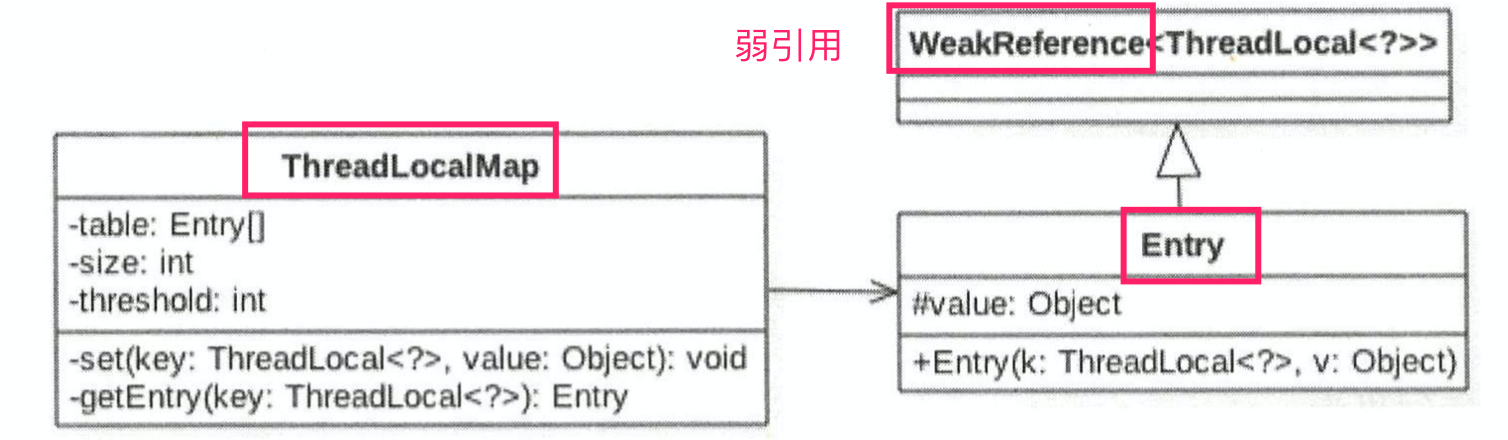
1. 线程局部变量(ThreadLocal)其实的功用非常简单，就是为每一个使用该变量的线程都提供一个变量值的副本，是Java中一种较为特殊的线程绑定机制，是每一个线程都可以独立地改变自己的副本，而不会和其它线程的副本冲突。
2. 通过ThreadLocal存取的数据，总是与当前线程相关，也就是说，JVM 为每个运行的线程，绑定了私有的本地实例存取空间，从而为多线程环境常出现的并发访问问题提供了一种隔离机制。如果你创建了一个 ThreadLocal 变量，那么访问这个变量的每个线程都会有这个变量的一个本地副本



1. ThreadLocal实现原理

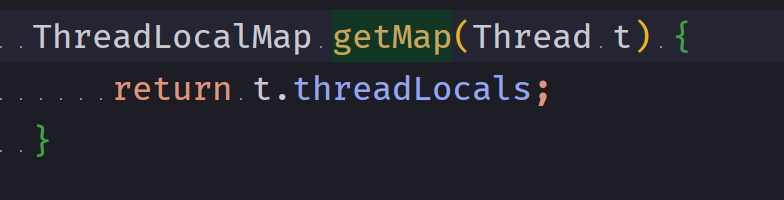
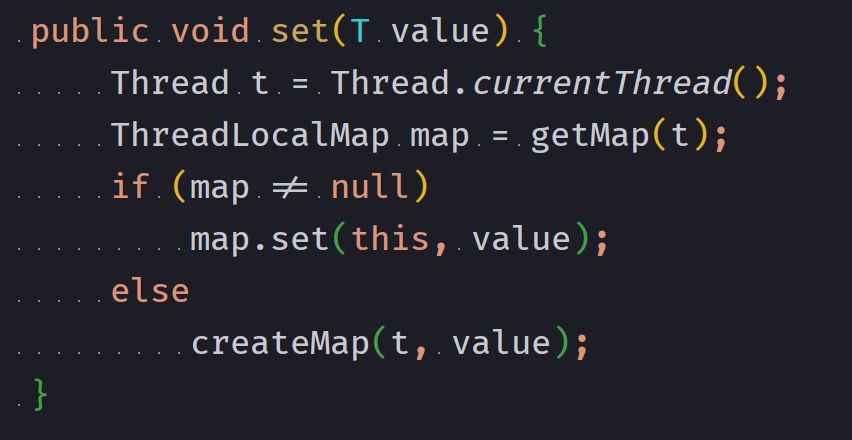


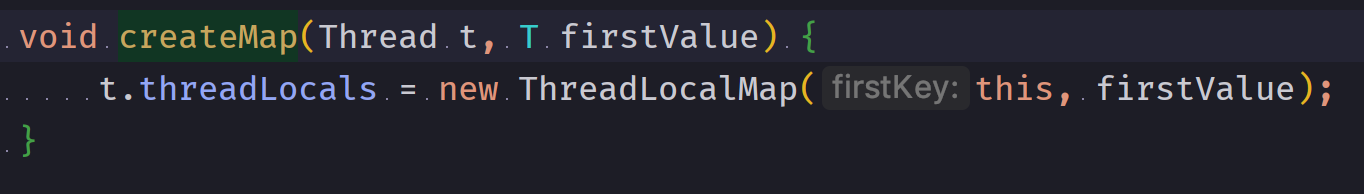
* 1. Thread类中有一个 threadLocals和一个inheritableThreadLocals， 它们都是 ThreadLocalMap 类型的变量，而 ThreadLocalMap 是一个定制化的 Hashmap
  2. 其实每个线程的本地变量不是存放在ThreadLocal实例里面， 而是存放在调用线程的 threadLocals变量里面。也就是说，ThreadLocal类型的本地变量存放在具体的线程内存空间. 如果当前线程一直不消亡，那么这些本地变量会一直存在，所以可能会造成内存溢出，因此使用完毕后要记得调用 ThreadLocal 的 remove方法删除对应线程的 threadLocals 中的本地变量



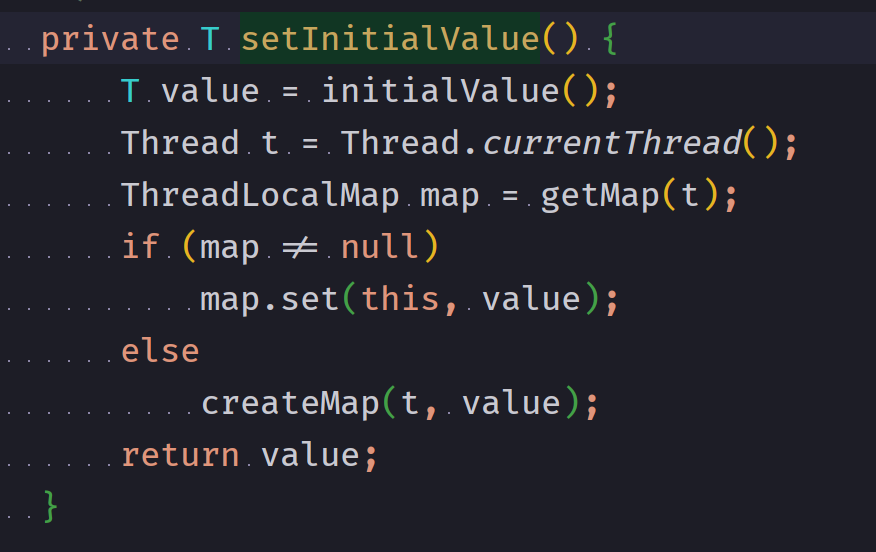
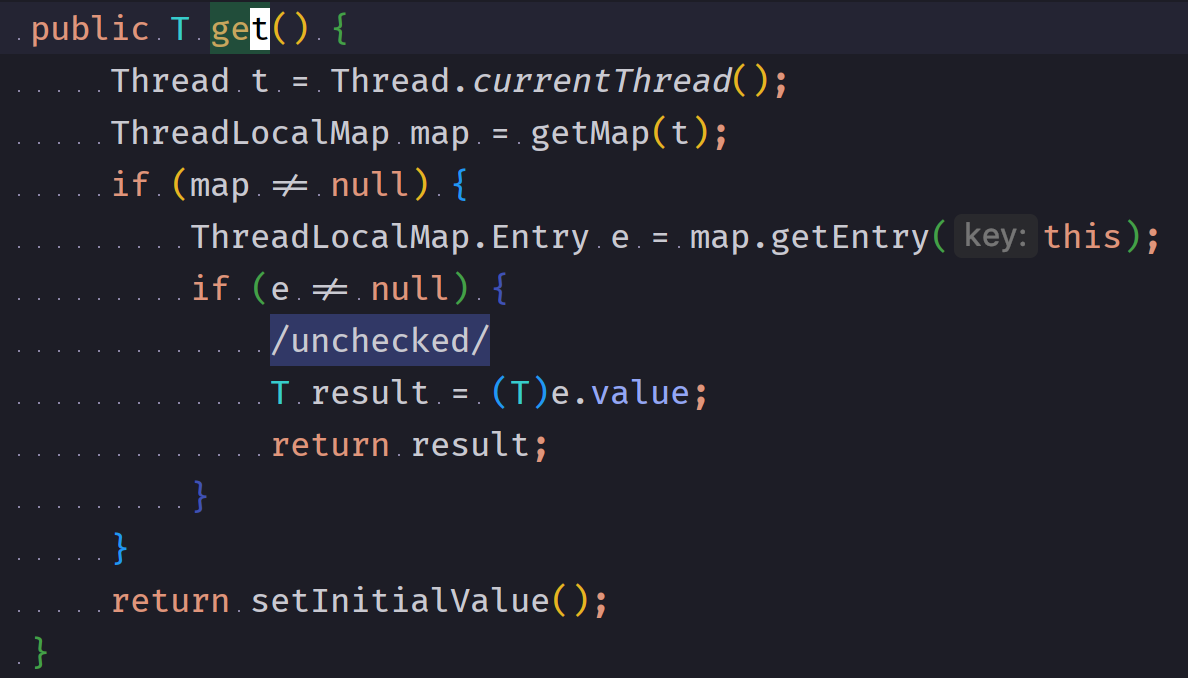
ThreadLocalMap的Entry 中的key使用的是ThreadLocal对象的弱引用，value为设置的值；如果在其它地方持有ThreadLocal对象的引用，则当前线程的ThreadLocalMap变量里面会存在对ThreadLocal变量的引用和对value对象的引用，不会释放，造成内存泄露；如果ThreaLocal对象没有其他强依赖，由于key是弱引用，所以在gc时候会被回收，但是对应的value不会别回收，这时候ThreadLocalMap里面会存在key为null但是value不为null的entry项，同样造成内存泄露。

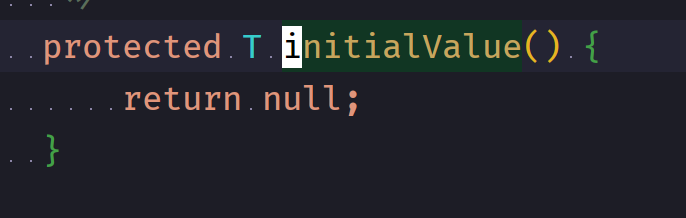
* 1. ThreadLocal就是一个工具壳，它通过 set方法把 value值放入调用线程的 threadLocals里面并存放起来，当调用线程调用它的 get方法时，再从当前线程的 threadLocals变量里面将其拿出来使用
     1. java.lang.ThreadLocal#set



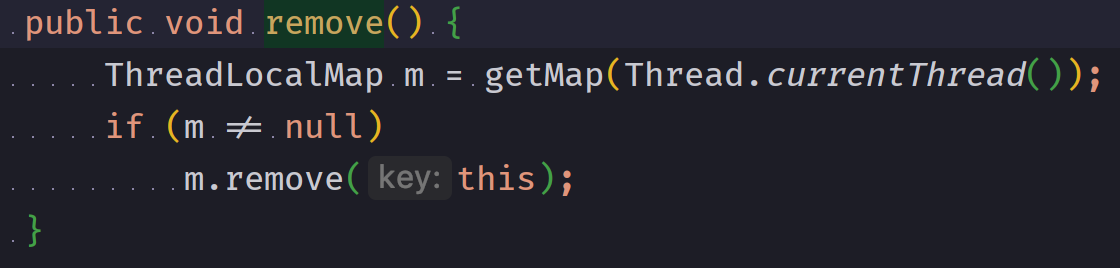


* + 1. java.lang.ThreadLocal#get

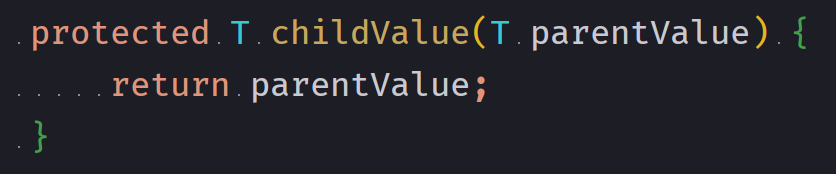


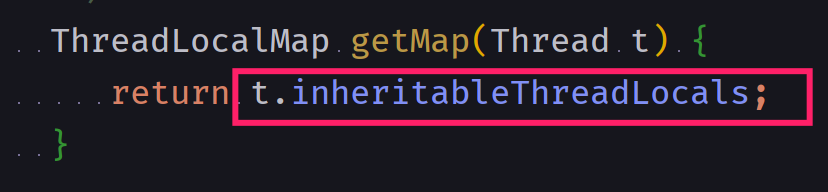


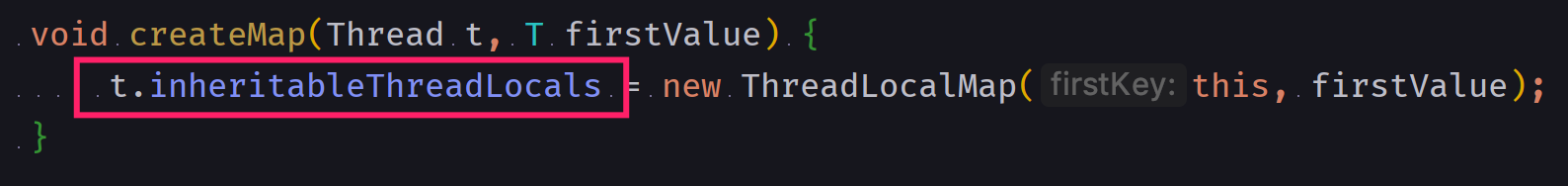
* + 1. java.lang.ThreadLocal#remove



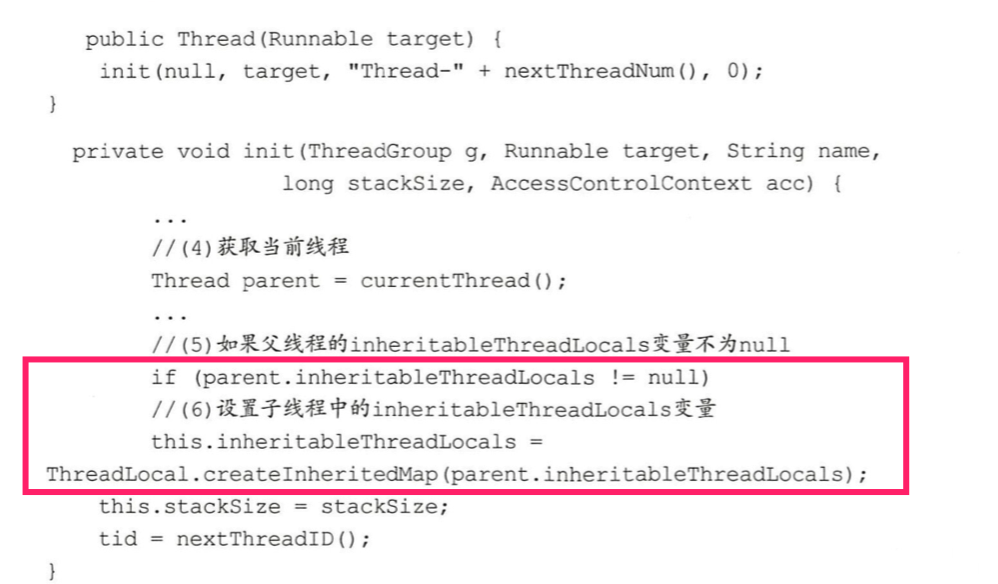
1. ThreadLocal不支持继承，即子线程获取不到父线程中的ThreadLocal值
2. InheritableThreadLocal类，子线程可以访问在父线程中设置的本地变量
   1. InheritableThreadLocal继承自ThreadLocal，相比于ThreadLocal，新加了一个特性，就是让子线程可以访问父线程中设置的本地变量。重写了以下三个方法

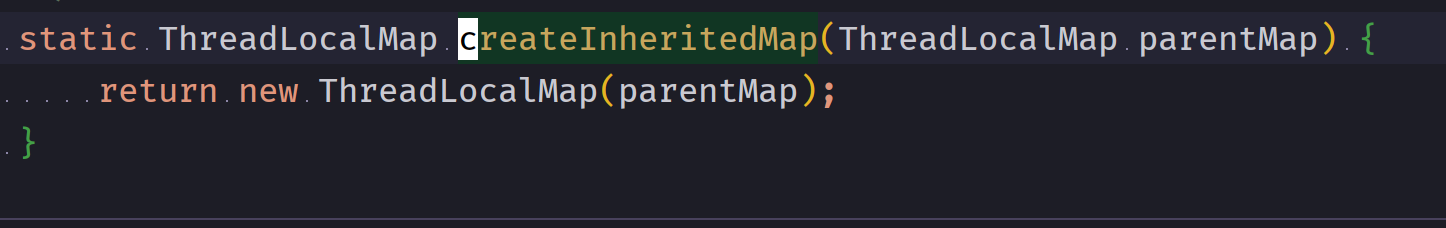


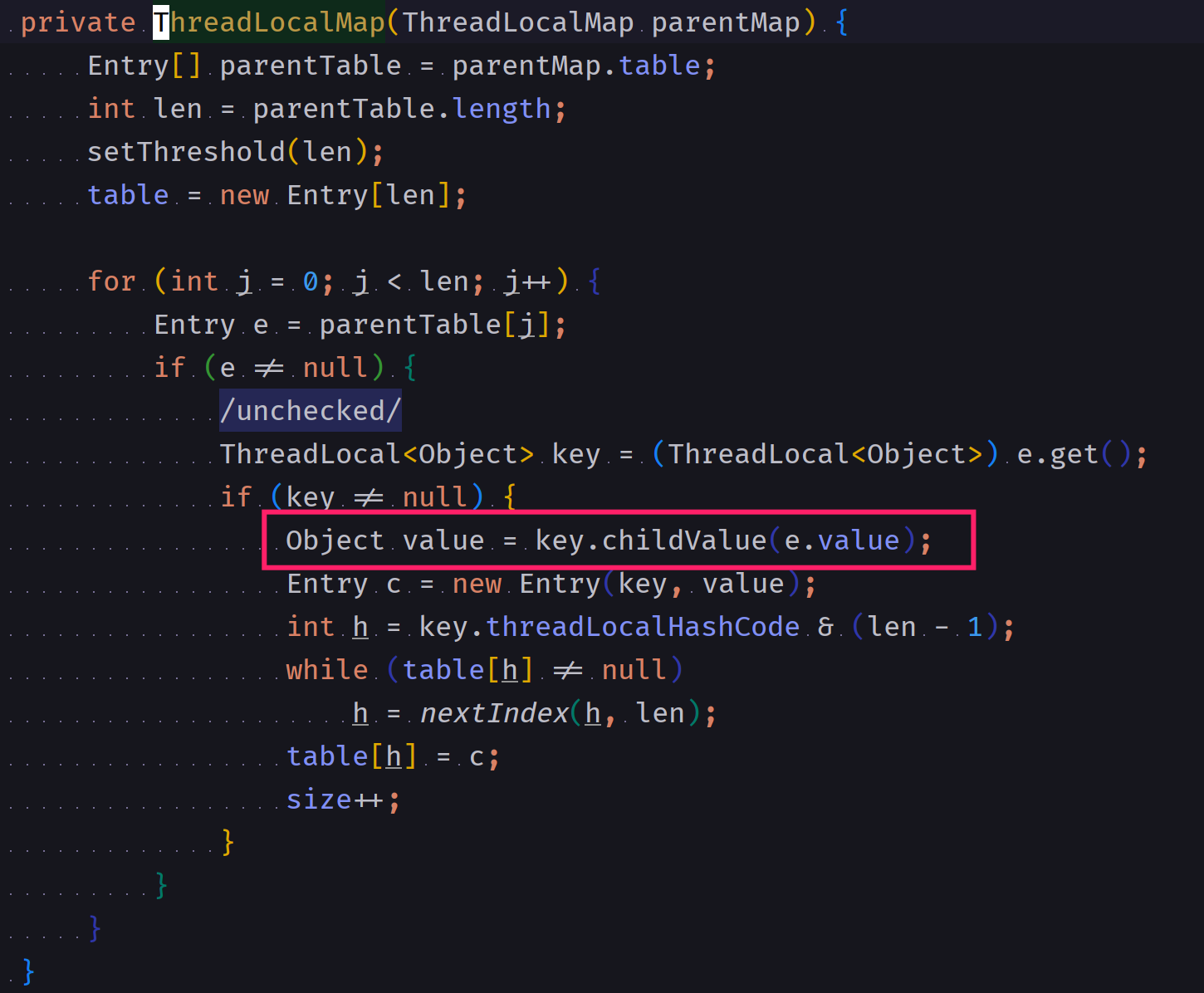




通过重写getMap和createMap方法，变量inheritableThreadLocals 替代了threadLocals。Thread类的构造方法

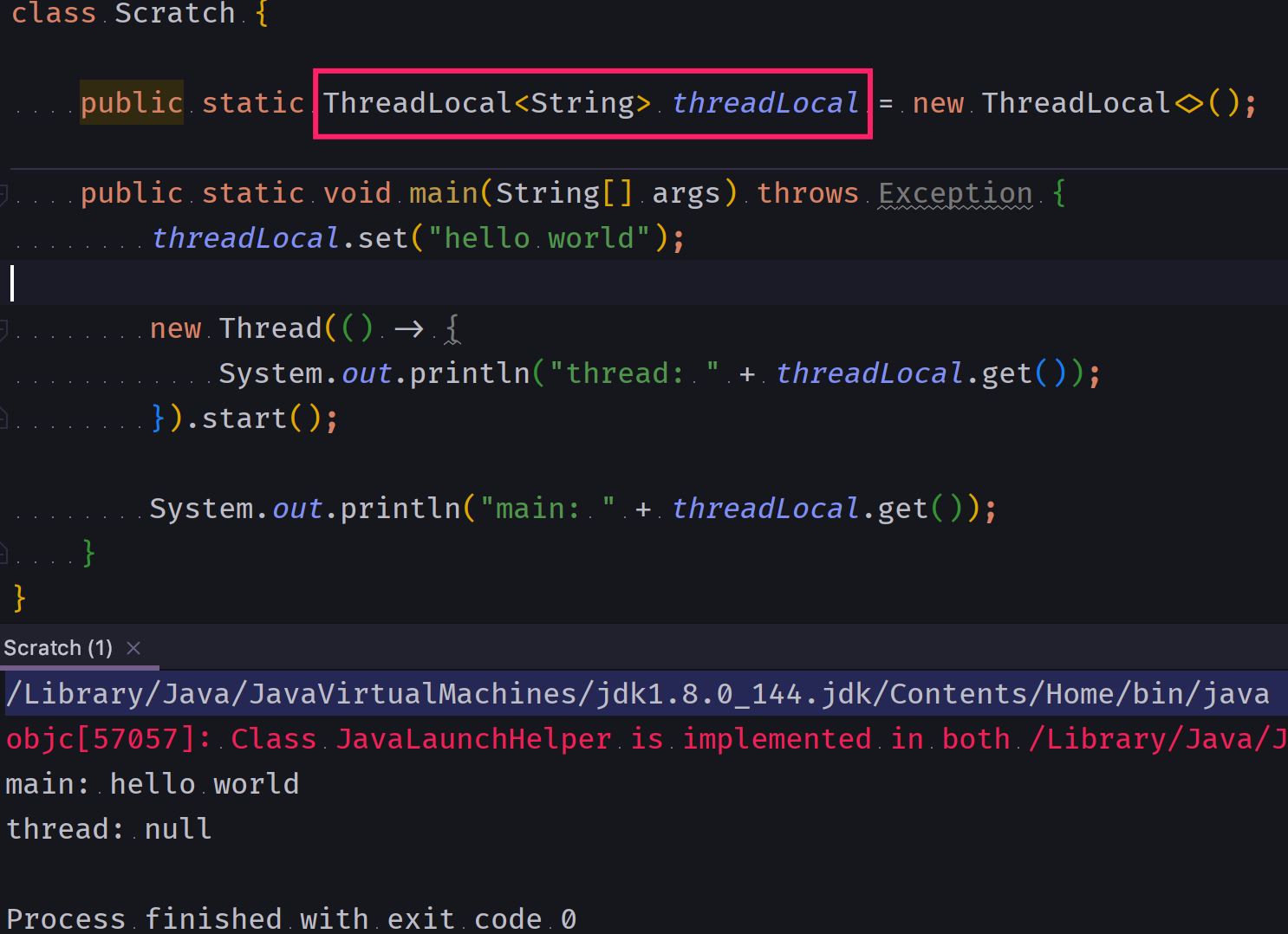






最终通过childValue方法获取父线程的值

1. 概括起来说，对于多线程资源共享的问题，同步机制采用了“以时间换空间”的方式，而ThreadLocal采用了“以空间换时间”的方式。前者仅提供一份变量，让不同的线程排队访问，而后者为每一个线程都提供了一份变量，因此可以同时访问而互不影响
2. 例
   1. ThreadLocal



* 1. InheritableThreadLocal

