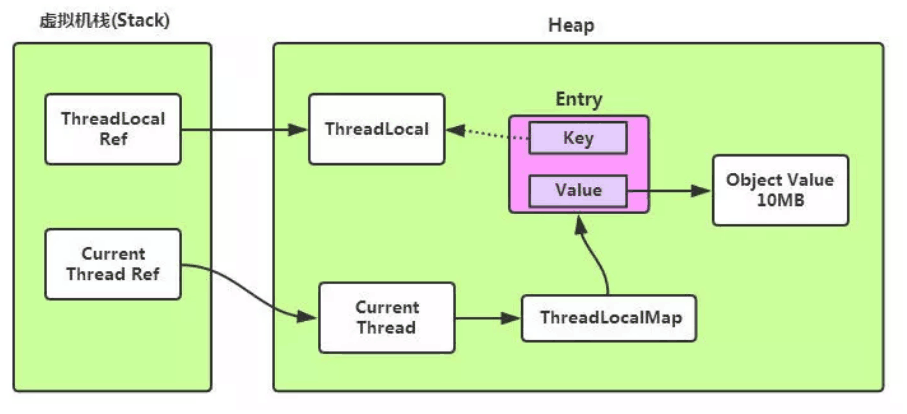
同类论述：

ThreadLocal使用不当导致内存泄漏的三个案例：

http://ifeve.com/%E4%BD%BF%E7%94%A8threadlocal%E4%B8%8D%E5%BD%93%E5%8F%AF%E8%83%BD%E4%BC%9A%E5%AF%BC%E8%87%B4%E5%86%85%E5%AD%98%E6%B3%84%E9%9C%B2/

首发于**[Java修仙道路](https://www.zhihu.com/column/c_1087293936418197504)**

写文章



# **ThreadLocal与内存泄漏**

[](https://www.zhihu.com/people/lu-lu-ya-15)

**[噜噜呀](https://www.zhihu.com/people/lu-lu-ya-15" \t "https://zhuanlan.zhihu.com/p/_blank)**

**[​](https://www.zhihu.com/question/48510028" \t "https://zhuanlan.zhihu.com/p/_blank)**

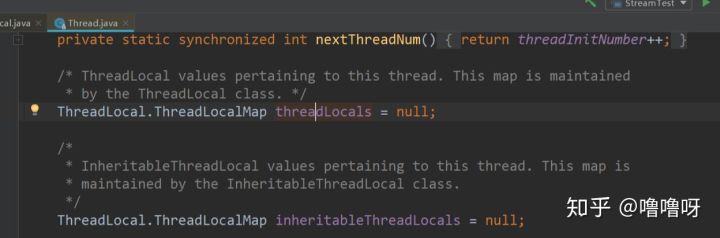
阿里巴巴 Java研发工程师

18 人赞同了该文章

阅读本文前，需要先了解ThreadLocal的原理。  
我们将从下面这些问题开始，寻找ThreadLocal中内存泄漏的罪魁祸首。

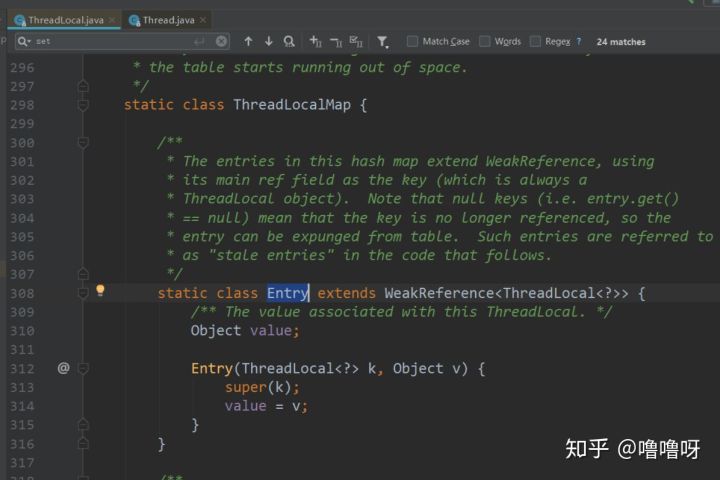
## **一、ThreadLocal，Thread，ThreadLocalMap，Entry<k,v>之间的关系？**

先来看看Thread源码



可以发现，每一个Thread中维护了一个ThreadLocalMap成员变量（也称threadLocals）。

再来看看ThreadLocal中的源码

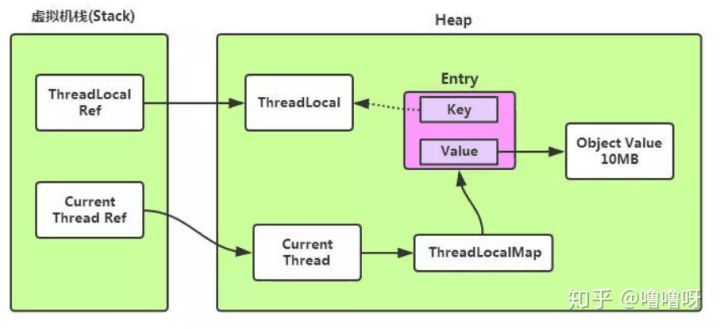


不难看出ThreadLocalMap类是ThreadLocal类的静态内部类，而Entry是ThreadLocalMap的静态内部类，key是ThreadLocal（声明为弱引用），value是Object，也就是我们要存的值。

## **二、ThreadLocal在预防内存泄漏方面，做了哪些努力？**

Thread中维护了ThreadLocalMap，所以ThreadLocalMap的生命周期和Thread（当前线程）一样长。使用不当就可能会导致内存泄漏问题。但是，在ThreadLocal中，进行get，set操作的时候会清除Map里所有key为null的value。

## **三、ThreadLocal的实现原理？**



ThreadLocal自身并不储存值，而是作为一个key来让线程从ThreadLocal获取value。Entry是中的key是弱引用，所以jvm在垃圾回收时如果外部没有强引用来引用它，ThreadLocal必然会被回收。但是，作为ThreadLocalMap的key，ThreadLocal被回收后，ThreadLocalMap就会存在null，但value不为null的Entry。若当前线程一直不结束，可能是作为线程池中的一员，线程结束后不被销毁，或者分配（当前线程又创建了ThreadLocal对象）使用了又不再调用get/set方法，就可能引发内存泄漏。其次，就算线程结束了，操作系统在回收线程或进程的时候不是一定杀死线程或进程的，在繁忙的时候，只会清除线程或进程数据的操作，重复使用线程或进程（线程id可能不变导致内存泄漏）。因此，**key弱引用并不是导致内存泄漏的原因，而是因为ThreadLocalMap的生命周期与当前线程一样长，并且没有手动删除对应key。**

那么，为什么要将Entry中的key设为弱引用？相反，设置为弱引用的key能预防大多数内存泄漏的情况。如果key 使用强引用，引用的ThreadLocal的对象被回收了，但是ThreadLocalMap还持有ThreadLocal的强引用，如果没有手动删除，ThreadLocal不会被回收，导致Entry内存泄漏。如果key为弱引用，引用的ThreadLocal的对象被回收了，由于ThreadLocalMap持有ThreadLocal的弱引用，即使没有手动删除，ThreadLocal也会被回收。value在下一次ThreadLocalMap调用set,get，remove的时候会被清除。

## **四、如何避免上述弱引用引发的内存泄漏？**

在使用完ThreadLocal时，及时调用它的的remove方法清除数据。

总而言之，如果开发者希望将类的某个静态变量与线程状态关联，可以考虑使用ThreadLocal。ThreadLocal的设计本身就是为了能够在当前线程中有属于自己的变量，并不是为了解决并发或者共享变量的问题。

博主常年在线，如有错误，欢迎评论指出。  
如果喜欢我的文章欢迎关注我的专栏~

编辑于 2019-05-10