

## 头条二面：你确定ThreadLocal真的会造成内存泄露？

Original 丁威 中间件兴趣圈 6 days ago

收录于话题

#技术干货 32 #面试 8


点击上方“中间件兴趣圈”，选择“设为星标”

做积极的人，越努力越幸运！



### 精品专栏推荐

- 消息中间件 RocketMQ 源码分析与线上故障案例排查案例
- 消息中间件 Kafka 源码分析
- 定时调度框架 Elastic-Job 源码分析
- RPC 服务框架 Dubbo 源码分析
- ORM 持久化框架 MyBatis 源码分析
- java8 实战
- 分布式全文搜索框架 Elasticsearch 使用指南

 中间件兴趣圈

ThreadLocal，java面试过程中的“钉子户”，在网上也充斥着各种有关ThreadLocal内存泄露的问题，**本文换个角度，先思考ThreadLocal体系中的ThreadLocalMap为什么要设计成弱引用。**

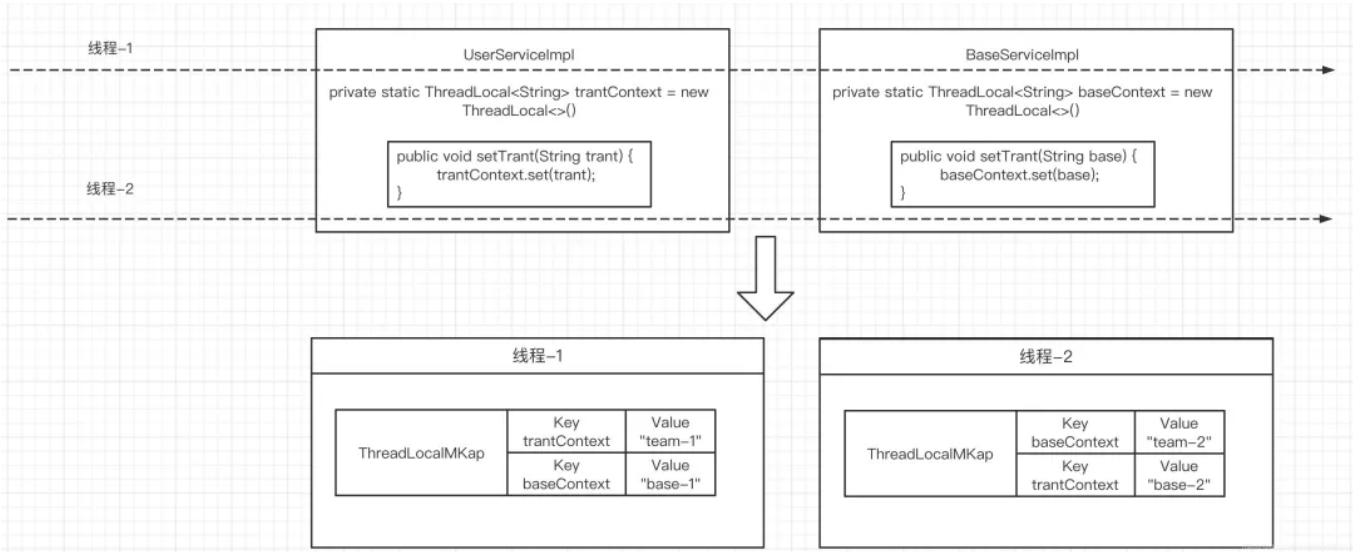
#### 1、ThreadLocal知识体系

本文还是不能免俗，在回答这个问题之前需要先和大家介绍一下ThreadLocal的知识，使大家对ThreadLocal有一个相对全面的认识。

ThreadLocal本地线程变量，主要用于解决数据访问的竞争，通常用于多租户、全链路压测、链路跟踪中保存线程上下文环境，在一个请求流转中非常方便的获取一些关键信息，例如当前的租户信息、压测标记。

ThreadLocal正如其名，本地线程变量，即数据存储在在线程自己的局部变量中。

其整体架构如下图所示：



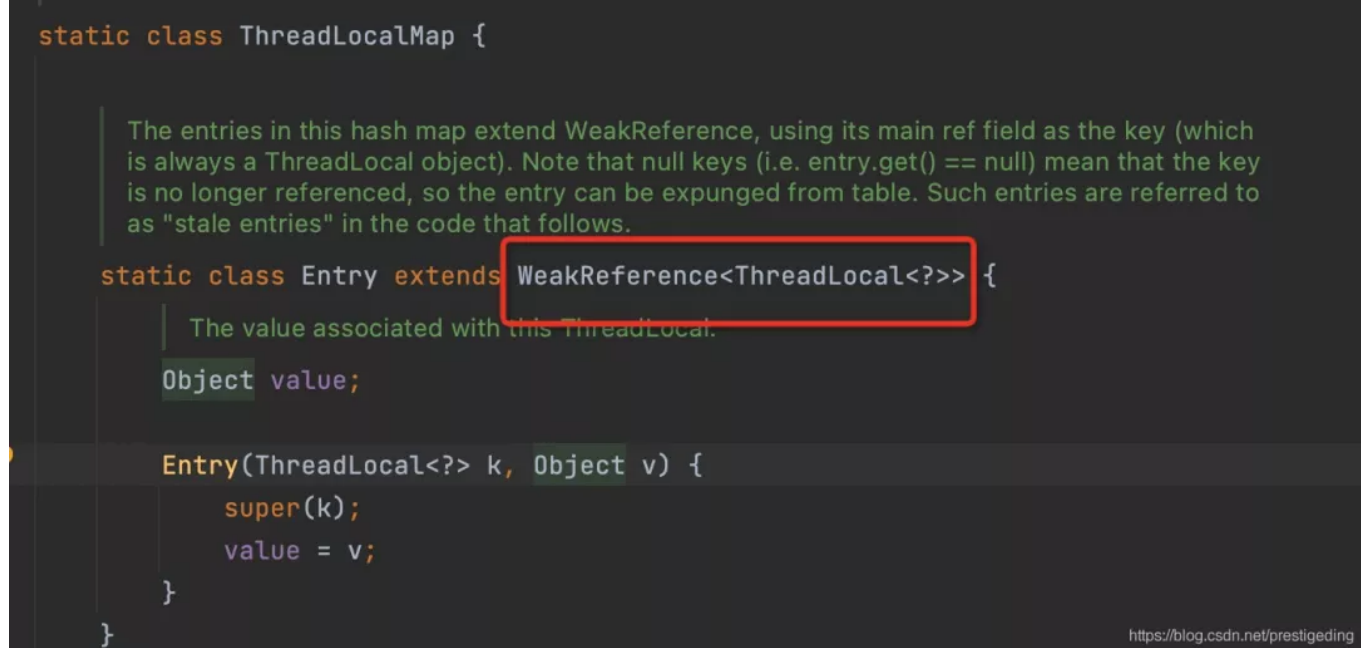
ThreadLocal的核心设计理念总结如下：

- 每一个线程对象会维护一个私有属性:`ThreadLocal.ThreadLocalMap threadLocals`。
- **ThreadLocalMap内部结构为Key-Value键值对，其Key为ThreadLocal对象，Value为调用ThreadLocal的set方法设置的值。**

一言以蔽之：**ThreadLocal是将线程需要访问的数据存储在线程对象自身中，从而避免多线程的竞争。**

## 2、为什么会被设计为弱引用呢？

接下来我们来看一下ThreadLocalMap的声明：



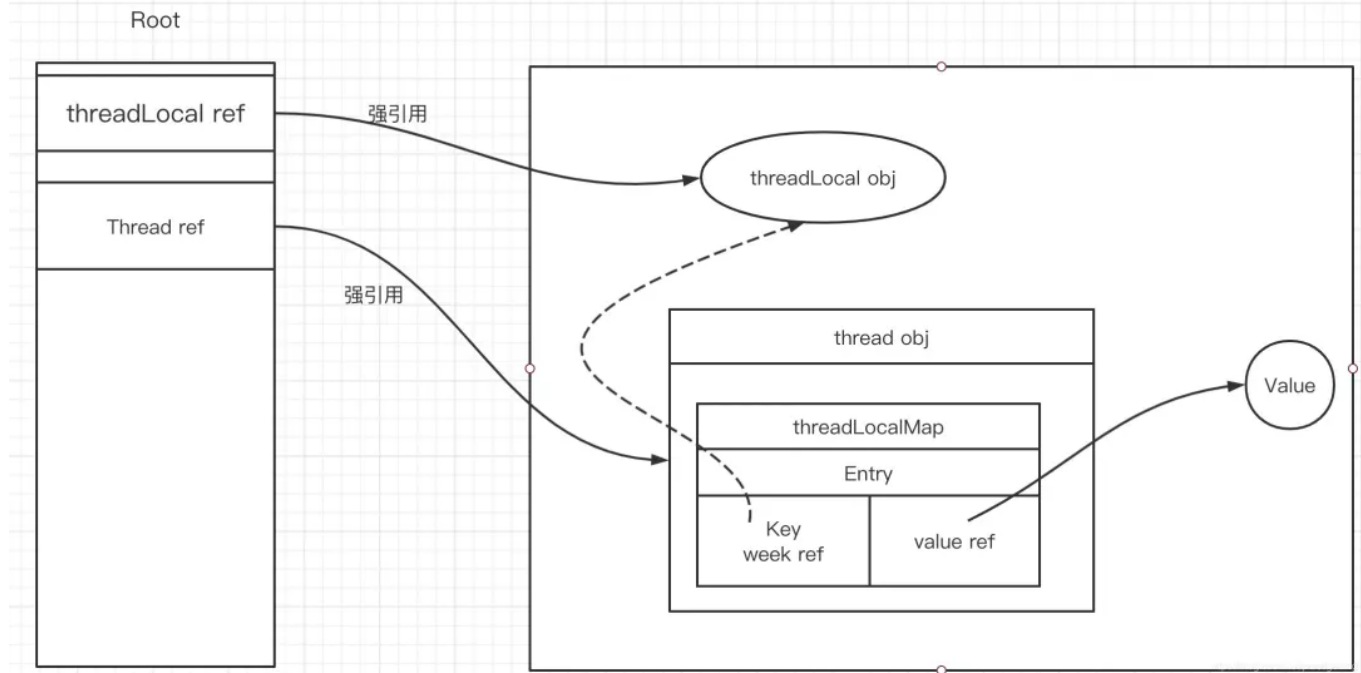
## 什么？Map中的用于存储键值对的Entry为什么要继承WeakReference？

思考这个问题之前先和大家普及一下Java的4种引用类型，主要是在垃圾回收时java虚拟机会根据不同的引用类型采取不同的措施。

- 强引用：java默认的引用类型，例如 `Object a = new Object();` 其中 `a` 为强引用，`new Object()` 为一个具体的对象。一个对象从根路径能找到强引用指向它，jvm虚拟机就不会回收。
- 软引用(SoftReference)：进行**年轻代的垃圾回收**不会触发SoftReference所指向对象的回收；但如果触发Full GC，那SoftReference所指向的对象将被回收。**备注：是除了软引用之外没有其他强引用引用的情况下。**
- 弱引用(WeakReference)：如果对象除了有弱引用指向它后没有其他强引用关联它，**当进行年轻代垃圾回收时，该引用指向的对象就会被垃圾回收器回收。**
- 虚引用(PhantomReference) 该引用指向的对象，无法对垃圾收集器收集对象时产生任何影响，但在执行垃圾回收后垃圾收集器会通过注册在PhantomReference上的队列来通知应用程序对象被回收。

从四种弱引用的实际作用来说，主要是与垃圾回收器配合，决策什么时候可以将被引用的对象回收。

理论看起来有点晦涩难懂，接下来笔者将以图解的方式，争取将该问题阐述清楚。



根据第一部分，声明了一个ThreadLocal对象，并且一个线程通过调用threadLocal对象的set(Object value)存储了一个对象，其引用如上图所示。

ThreadLocal的设计比较晦涩难懂，究其原因是我们通过threadLocal对象的set方法进行存储值，但数据并不是存储在ThreadLocal对象中，而是存储在当前调用该方法的线程对象中。但从应用者的角度来看，我们操作的对象是ThreadLocal，从设计上来说就应该为它考虑。

试问一个问题：如果应用程序觉得ThreadLocal对象的使命完成，将threadLocal ref 设置为null，如果Entry中引用ThreadLocal对象的引用类型设置为强引用的话，会发生什么问题？

答案是：ThreadLocal对象会无法被垃圾回收器回收，因为从thread对象出发，有强引用指向threadlocal obj。此时会违背用户的初衷，造成所谓的内存泄露。

由于ThreadLocalMap中的key是指向ThreadLocal，故从设计角度来看，设计为弱引用，将不会干扰用户的释放ThreadLocal意图。

### 3、大量Entry造成的内存溢出问题探讨

亮出了自己的观点，接下来我们再延伸一下，想再来谈谈网络上关于ThreadLocalMap中存储大量Entry对象导致的内存“泄露”问题。

温馨提示：本节仅代表我当前的观点，希望各位读者朋友们带着批判与辩证的思维来一起看待问题，而不是人云亦云。

我的观点：当然能成对使用当然更好，但在实际情况中，其实不调用remove方法也不太容易造成内存溢出，因为从存储结构来看，除非创建海量线程，并且这些线程都不释放，导致大量线程内部持有的ThreadLocalMap中对象一直不会释放，但一个线程所持有的Entry对象个数不多，取决于关联的ThreadLocal对象个数，**故我们需要的关注点而不是remove方法，而是防止线程资源泄露。**

## 全链路跟踪(压测)必备基础组件之线程上下文“三剑客”

掌握一到两门java主流中间件，是敲开BAT等大厂必备的技能，送给大家一个Java中间件学习路线，助力大家实现职场的蜕变。

最后分享笔者一个硬核的RocketMQ电子书，您将获得千亿级消息流转的运维经验。

 **中间件兴趣圈**  
《RocketMQ技术内幕》作者维护，主打成体系剖析JAVA主流中间件架构与设计原理，为构建完备的互联...  
172篇原创内容

获取方式：关注公众号，回复RMQPDF即可免费获取。



---

## 走进作者

---

10年IT老兵给职场新人的一些建议  
“我”被阿里巴巴宠幸了  
程序员如何提高影响力  
优秀程序员必备技能之如何高效阅读源码  
我的另一种参与 RocketMQ 开源社区的方式

点击查看“[阅读原文](#)”，可直接进入[\[中间件兴趣圈\]](#)文章合集。

收录于话题 [#技术干货](#)·32个

[上一篇](#) · [java并发高频面试题:Sempahore的使用场景与常见误区](#)

Read more

People who liked this content also liked

[java并发高频面试题:Sempahore的使用场景与常见误区](#)  
[中间件兴趣圈](#)

---

[Java内存攻击技术漫谈](#)  
[SilverNeedleLab](#)

---

[手把手教你Locust压测实践](#)  
[测试开发栈](#)