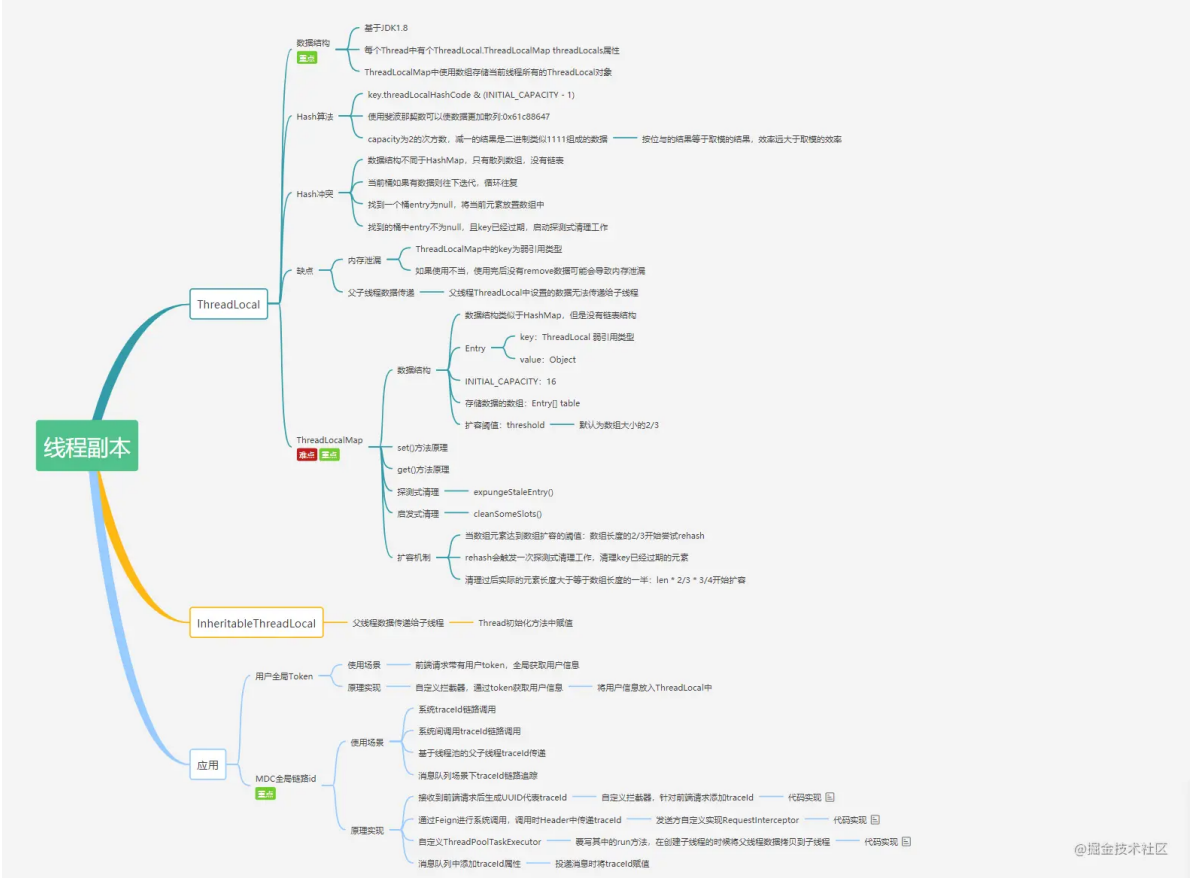
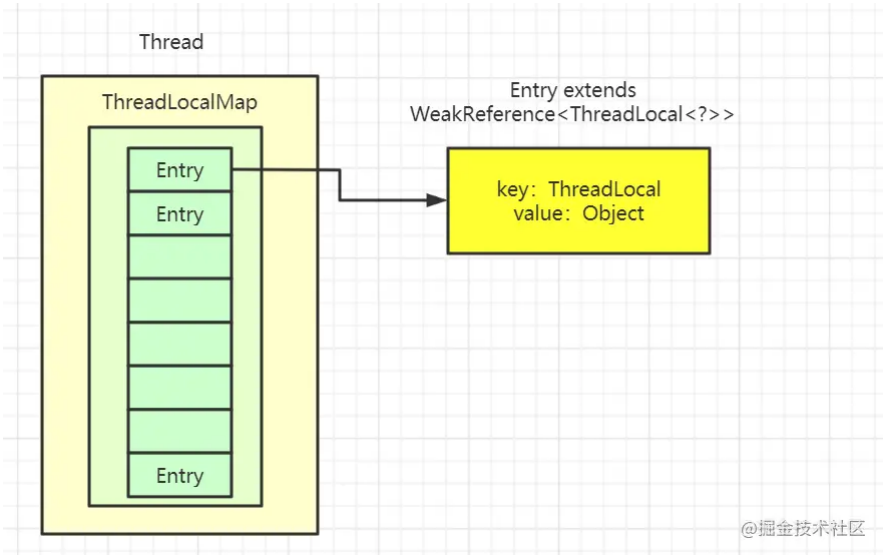
ThreadLocal(线程变量)



ThreadLocal类型是为了解决线程独享变量并且高效调用(避免形参传递)场景而催生的一种类型，为了解决多个独享变量场景，ThreadLocal有个内部类ThreadLocalMap实现了ThreadLocal专用的类hashmap结构； 实现了get/set/remove等方法能够访问和修改该thredlocal变量(key)在当前线程中的对应数据(value)．每个线程thread中有个ThreadLocalMap类型的成员变量threadlocals，对于同一个threadlocal变量会在不同的线程各自的threadlocals中操作对应的value，因此就实现了独享变量在不同线程间的隔离．

ThreadLocalMap



ThreadLocalMap底层用Entry数组实现没有链表结构，因此利用开放定址法(线性探测再hash)来解决冲突问题，Entry的key是ThreadLocal变量的**弱引用**，value是Object类型；由于key是弱引用，会存在key为null的情况，额外引入了过期key清理的一些策略．

- key弱引用
  - key设置为弱引用原因

    Thread是一个长生命周期对象，ThreadLocal是一个短生命周期对象，Thread通过threadlocals间接持有ThreadLocal，会造成长生命周期对象持有短生命周期对象导致短生命周期对象无法被释放的问题，因此将ThreadLocal用弱引用修饰，当其只被threadlocals持有弱引用时，就会被GC回收.
  - GC时key情况

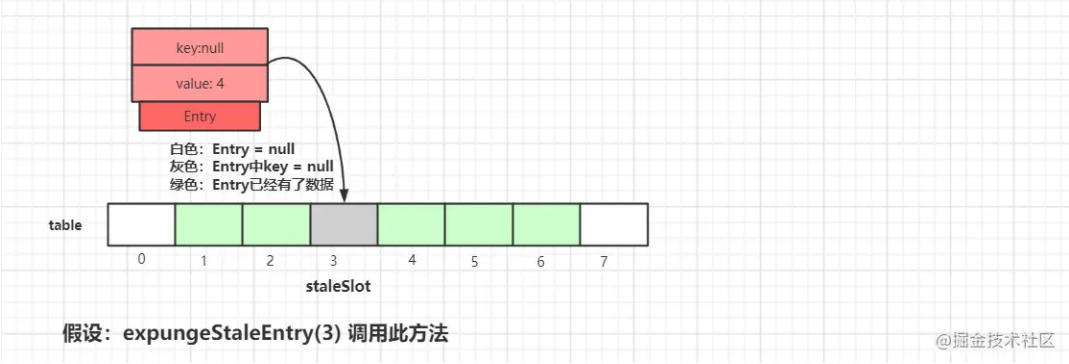
    如果GC时threadlocal的没有强引用(例如匿名实例，new ThreadLocal<>())，则此时被回key=null，否则key不为null.
  - 内存泄漏

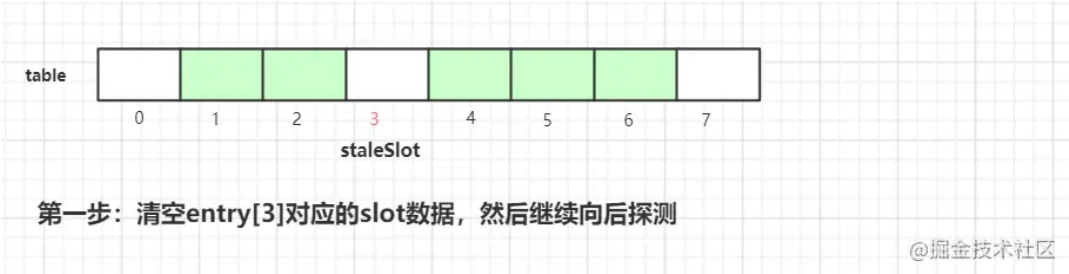
    key被回收置空，而value未被回收，造成value泄漏；预防策略每次使用完threadlocal后调用remove方法清除.
- Hash算法
  - ThreadLocal变量hashcode

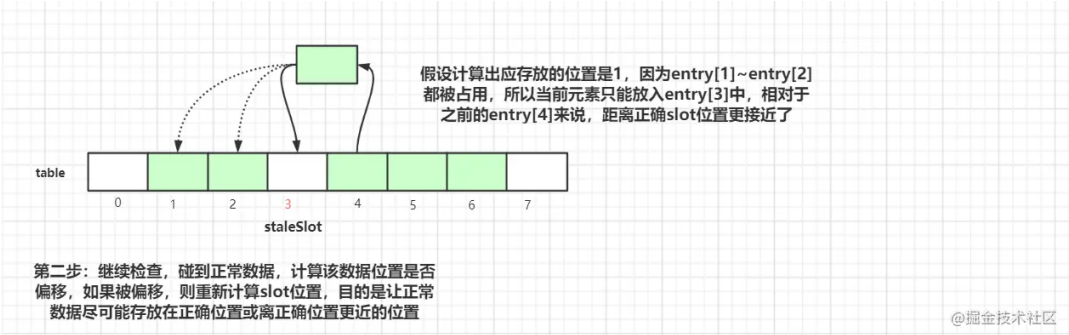
    ThreadLocal类中有个静态的AtomicInteger类型的变量nextHashCode，初始化为0，每次生成新的ThreadLocal实例时，nextHashCode会加上一个HASH_INCREMENT(斐波那契数，这个数使得散列分布均匀)作为该threadlocal变量的threadLocalHashCode.
  - ThreadLocal变量hash寻址

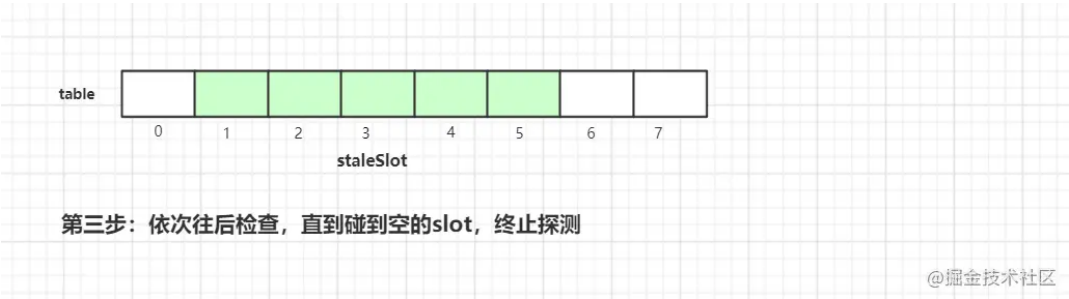
    类似于hashmap的策略，用ThreadLocal中的threadLocalHashCode & (len -1)来寻址.
  - Hash冲突解决策略

    忽略细节来说，如果当前位置有ThreadLocal实例且key不为自己(冲突)，则向后线性搜索，直到找到首个null/key为自己/key==null(过期)的位置进行放置/更新.
- 过期key清理策略
  - 探测式清理



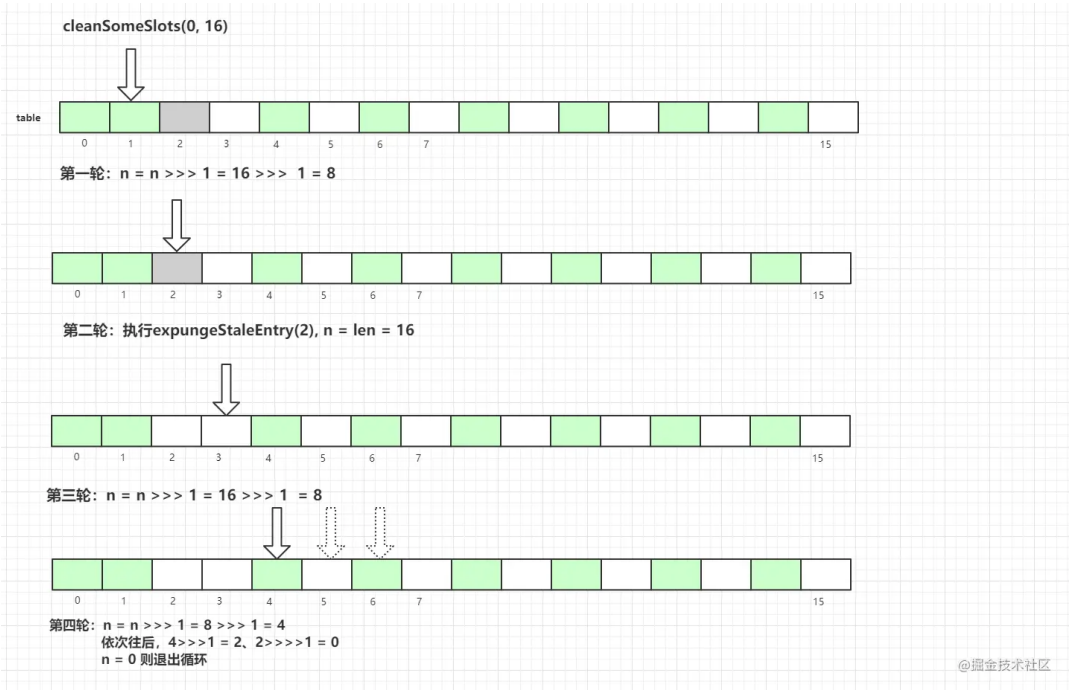






探测式清理的策略如下：首先将当前传入位置staleSlot的Entry[staleSlot]清空，然后向后遍历，碰到key==null的staleEntry就清空Entry[cur]；碰到有效的key，就计算它的hash地址h，从h开始向后探测空的位置，将当前有效Entry[cur]移到距离h最近的位置(重新调整位置，为了get时的效率更高)；碰到空的位置就结束清理.

- 启发式清理



启发式清理策略如下：输入检测开始位置i和次数控制n，从(i+1)%len开始，总共做logn + 1次探测，如果当次元素Entry[cur]为staleSlot则从cur开始执行一次探测式清理，否则进入下一轮的当前元素
检测Entry[(cur+1)%len].

- set过程

  计算h = threadlocal.threadLocalHashCode & (len -1)

  - case 1 & 2

    Entry[h]为空或者Entry[h].key是当前threadlocal则直接存储/更新value.

  - case 3

Entry[h]有效且不为当前threadlocal，向后遍历的过程中，先发现case1/2，直接存储/更新value.

- case 4

  Entry[h]有效且不为当前threadlocal，向后遍历的过程中，先发现了key=null的Entry[slateSlot]过期元素，则需要需要找到包含当前Entry[slateSlot]的一个最小run(null-xxx-null，头尾为null的一个连续序列)，清理run中的所有过期元素，然后将threadlocal和value插入到当前slateSlot位置(从slateSlot开始找run时，向前找首个null元素时，记录最外侧的过期元素的位置slotToExpunge；向后找首个null元素时，如果碰到key相等的情况也停止，当前位置视为null).

- 扩容机制
  - rehash

    set函数最后会执行一轮启发式清理，如果这个过程中没有清除的任何过期元素并且当前的**元素数量**
    size>=threshold，则进入rehash函数，rehash函数从数组首部开始遍历，碰到一个过期元素则进
    行一次探测式清理.

  - resize

    在rehash函数执行完清理过程后，如果当前有效元素的数量**size>=3/4 * threshold**则进入resize，resize会将创建一个2*oldlenth的新数组(**两倍扩容**)，然后遍历原数组，将元素放到h位置或者距离h
    最近的无冲突位置.

  - threshold

    threshold = newlenth * 2/3

- get过程

  计算h = threadlocal.threadLocalHashCode & (len -1)

  - case 1

    Entry[h]的key与当前threadlocal相同，则直接返回value.

  - case 2

    Entry[h]的key与当前threadlocal 不相同，向后遍历搜索key，每碰到key=null的过期元素，则调用探测式清理，如果结束都未找到key=threadlocal的元素，则返回null.

- **参考资料**
  - 面试官：小伙子，听说你看过ThreadLocal源码？（万字图文深度解析ThreadLocal） - 掘金 (juejin.cn)
  - Java ThreadLocal你之前了解的可能有误_小鱼人爱编程的博客-CSDN博客