链滴

ThreadLocal 笔记

作者：Hassan

原文链接：https://ld246.com/article/1471923020256

来源网站：链滴

<h2 id="toc_h2_0">作用</h2>
ThreadLocal不是用来解决共享对象访问的多线程访问问题，而是用于解决不同线程保持各自独的一个对象。 典型的问题就是：当一个单例A持有某个属性对象a.b时，如果a.b在多个方法里面使用就有可能造成线程不安全，如果把b定义成 ThreadLocal<B> b 就可以避免以上问题。
<h2 id="toc_h2_1">实现</h2>
<h4 id="toc_h4_2">一、ThreadLocal</h4>
实例方法：

<pre class="brush: java">public void set(T value) {
    Thread t = Thread.currentThread();
    ThreadLocalMap map = getMap(t);
    if (map != null)
        map.set(this, value);
    else
        createMap(t, value);
}
ThreadLocalMap getMap(Thread t) {
return t.threadLocals;
}
void createMap(Thread t, T firstValue) {
t.threadLocals = new ThreadLocalMap(this, firstValue);
}
public T get() {
Thread t = Thread.currentThread();
ThreadLocalMap map = getMap(t);
if (map != null) {
ThreadLocalMap.Entry e = map.getEntry(this);
if (e != null)
return (T)e.value;
}
return setInitialValue();
} </pre>

set()方法逻辑： 获取当前线程对应的ThreadLocalMap map; 如果存在直接map.set; 否则new ThreadLocalMap给线程使用。
由此可知，每个线程都有一个自己的ThreadLocal.ThreadLocalMap对象。
get()方法逻辑： 获取当前线程对应的ThreadLocalMap map; 如果存在，且当前ThreadLocal例对应的值不为空，返回map拿到的值； 否则，设置前ThreadLocal实例默认值并返回。
<h4 id="toc_h4_3">二、ThreadLocalMap</h4>
ThreadLocalMap构造函数：

<pre class="brush: java">private static final int INITIAL_CAPACITY = 16;
ThreadLocalMap(ThreadLocal firstKey, Object firstValue) {
table = new Entry[INITIAL_CAPACITY];
int i = firstKey.threadLocalHashCode & (INITIAL_CAPACITY - 1);
table[i] = new Entry(firstKey, firstValue);
size = 1;
setThreshold(INITIAL_CAPACITY);
} </pre>

第一点：INITIAL_CAPACITY必须是2的N次幂，默认值为16。
为什么是2的N次幂值？
*ThreadLocalMap类保存着一个table *每一个ThreadLocal有自己的threadLocalHashCode值

从ThreadLocalMap table里存/取值的时候会通过threadLocalHashCode值计算出一个i，再通过able[i]得到ThreadLocal的值。 如构造函数代码所示：

<pre>int i = firstKey.threadLocalHashCode
span>& (INITIAL_CAPACITY - 1</span
);</pre>

这是计算方法，这个过程实际上是一个取模的过程。
举个例子
<pre><code>十进制取模：26 % 16 = 10
二进制取模：
  00011010
& 00001111
= 00001010
= 10
</code></pre>
所以，十进制的取模对于二进制，只需要使用公式 M & (C-1)即可，这种与操作对于CPU算效率很高。当然，一个大前提就是C-1的值转换为二进制时，低位部分要求全是1才行。所以要求C须是2的N次幂。
第二点：threadLocalHashCode
看一下ThreadLocal这部分的代码：

<pre class="brush: java">/**
 * ThreadLocals rely on per-thread linear-probe hash maps attached
 * to each thread (Thread.threadLocals and
 * inheritableThreadLocals). The ThreadLocal objects act as keys,
 * searched via threadLocalHashCode. This is a custom hash code
 * (useful only within ThreadLocalMaps) that eliminates collisions
 * in the common case where consecutively constructed ThreadLocals
 * are used by the same threads, while remaining well-behaved in
 * less common cases.
 */
private final int threadLocalHashCode = nextHashCode();
/**

The next hash code to be given out. Updated atomically. Starts at
zero.
*/
private static AtomicInteger nextHashCode = new AtomicInteger();

/**

The difference between successively generated hash codes - turns
implicit sequential thread-local IDs into near-optimally spread
multiplicative hash values for power-of-two-sized tables.
*/
private static final int HASH_INCREMENT = 0x61c88647;

/**

Returns the next hash code.
*/
private static int nextHashCode() {
return nextHashCode.getAndAdd(HASH_INCREMENT);

}
</pre>

可以看出来，ThreadLocal第一次set值的时候，threadLocalHashCode得到的是0，之后每次到的数都是加了0x61c88647。这算一个16进制表示的数，转换成十进制是：1640531527。 为什么这个数？ 本屌查过一些资料，貌似都是数学问题，还涉及到黄金比例，看的不是很懂，欢迎留言指教
简单的总结一下ThreadLocal：
1、每一个ThreadLocal实例有一个自己的threadLocalHashCode；
2、每一个Thread有一个自己的ThreadLocalMap threadLocals， threadLocals的key是ThreadLocal实例，value是ThreadLocal真是的实际保存的对象实例。
3、ThreadLocalMap使用table数组保存每一个Entry（key-value）。
4、ThreadLocalMap计算ThreadLocal对应table[i]的i使用threadLocalHashCode取模获得。