正确理解ThreadLocal

锁定老帖子 主题：正确理解ThreadLocal
该帖已经被评为良好帖

作者

- lujh99
- 等级:
- 文章: 74
- 积分: 165
- 来自: ...
- 我现在离线

正文
发表时间：2007-07-21
相关推荐:

- ThreadLocal-分析-总结
- 深入浅出ThreadLocal
- ThreadLocal and synchronized 补充
- 正确理解 ThreadLocal
- 二年级上册数学期中考试试卷.doc
- 完美版资料集1关于软件开发过程模型及CMM相关题与实际结合重点.doc
- 8X8LED驱动（仿真文件+设计报告+PCB+视频讲解+原理图）.zip
- 优秀资料（2021-2022年收藏）师德师风建设网络培训试卷.doc

推荐群组: Hibernate
更多相关推荐
设计模式
首先，ThreadLocal 不是用来解决共享对象的多线程访问问题的，一般情况下，通过ThreadLocal.set() 到线程中的对象是该线程自己使用的对象，其他线程是不需要访问的，也访问不到的。各个线程中访问的是不同的对象。

另外，说ThreadLocal使得各线程能够保持各自独立的一个对象，并不是通过ThreadLocal.set()来实现的，而是通过每个线程中的new 对象 的操作来创建的对象，每个线程创建一个，不是什么对象的拷贝或副本。通过ThreadLocal.set()将这个新创建的对象的引用保存到各线程的自己的一个map中，每个线程都有这样一个map，执行ThreadLocal.get()时，各线程从自己的map中取出放进去的对象，因此取出来的是各自自己线程中的对象，ThreadLocal实例是作为map的key来使用的。

如果ThreadLocal.set()进去的东西本来就是多个线程共享的同一个对象，那么多个线程的ThreadLocal.get()取得的还是这个共享对象本身，还是有并发访问问题。

下面来看一个hibernate中典型的ThreadLocal的应用：
Java代码

```
1. private static final ThreadLocal threadSession = new ThreadLocal();
2.
3. public static Session getSession() throws InfrastructureException {
4.     Session s = (Session) threadSession.get();
5.     try {
6.         if (s == null) {
7.             s = getSessionFactory().openSession();
8.             threadSession.set(s);
9.         }
10.    } catch (HibernateException ex) {
11.        throw new InfrastructureException(ex);
12.    }
13.    return s;
14. }
```

可以看到在getSession()方法中，首先判断当前线程中有没有放进去session，如果还没有，那么通过sessionFactory().openSession()来创建一个session，再将session set到线程中，实际是放到当前线程的ThreadLocalMap这个map中，这时，对于这个session的唯一引用就是当前线程中的那个ThreadLocalMap（下面会讲到），而threadSession作为这个值的key，要取得这个session可以通过threadSession.get()来得到，里面执行的操作实际是先取得当前线程中的ThreadLocalMap，然后将threadSession作为key将对应的值取出。这个session相当于线程的私有变量，而不是public的。
显然，其他线程中是取不到这个session的，他们也只能取到自己的ThreadLocalMap中的东西。要是session是多个线程共享使用的，那还不乱套了。
试想如果不用ThreadLocal怎么来实现呢？可能就要在action中创建session，然后把session一个个传到service和dao中，这可够麻烦的。或者可以自己定义一个静态的map，将当前thread作为key，创建的session作为值，put到map中，应该也行，这也是一般人的想法，但事实上，ThreadLocal的实现刚好相反，它是在每个线程中有一个

锁定老帖子 主题：正确理解ThreadLocal
该帖已经被评为良好帖

作者 正文

map，而将ThreadLocal实例作为key，这样每个map中的项数很少，而且当线程销毁时相应的东西也一起销毁了，不知道除了这些还有什么其他的好处。

总之，ThreadLocal不是用来解决对象共享访问问题的，而主要是提供了保持对象的方法和避免参数传递的方便的对象访问方式。归纳了两点：
1。每个线程中都有一个自己的ThreadLocalMap类对象，可以将线程自己的对象保持到其中，各管各的，线程可以正确的访问到自己的对象。
2。将一个共用的ThreadLocal静态实例作为key，将不同对象的引用保存到不同线程的ThreadLocalMap中，然后在线程执行的各处通过这个静态ThreadLocal实例的get()方法取得自己线程保存的那个对象，避免了将这个对象作为参数传递的麻烦。

当然如果要把本来线程共享的对象通过ThreadLocal.set()放到线程中也可以，可以实现避免参数传递的访问方式，但是要注意get()到的是那同一个共享对象，并发访问问题要靠其他手段来解决。但一般来说线程共享的对象通过设置为某类的静态变量就可以实现方便的访问了，似乎没必要放到线程中。

ThreadLocal的应用场合，我觉得最适合的是按线程多实例（每个线程对应一个实例）的对象的访问，并且这个对象很多地方都要用到。

下面来看看ThreadLocal的实现原理（jdk1.5源码）
Java代码

```
1. public class ThreadLocal<T> {
2.     /**
3.      * ThreadLocals rely on per-thread hash maps attached to each thread
4.      * (Thread.threadLocals and inheritableThreadLocals).  The ThreadLocal
5.      * objects act as keys, searched via threadLocalHashCode.  This is a
6.      * custom hash code (useful only within ThreadLocalMaps) that eliminates
7.      * collisions in the common case where consecutively constructed
8.      * ThreadLocals are used by the same threads, while remaining well-behaved
9.      * in less common cases.
10.     */
11.    private final int threadLocalHashCode = nextHashCode();
12.
13.    /**
14.     * The next hash code to be given out. Accessed only by like-named method.
15.     */
16.    private static int nextHashCode = 0;
17.
18.    /**
19.     * The difference between successively generated hash codes - turns
20.     * implicit sequential thread-local IDs into near-optimally spread
21.     * multiplicative hash values for power-of-two-sized tables.
22.     */
23.    private static final int HASH_INCREMENT = 0x61c88647;
24.
25.    /**
26.     * Compute the next hash code. The static synchronization used here
27.     * should not be a performance bottleneck. When ThreadLocals are
28.     * generated in different threads at a fast enough rate to regularly
29.     * contend on this lock, memory contention is by far a more serious
30.     * problem than lock contention.
31.     */
32.    private static synchronized int nextHashCode() {
33.        int h = nextHashCode;
34.        nextHashCode = h + HASH_INCREMENT;
35.        return h;
36.    }
37.
38.    /**
39.     * Creates a thread local variable.
40.     */
41.    public ThreadLocal() {
42.    }
43.
44.    /**
45.     * Returns the value in the current thread's copy of this thread-local
```

锁定老帖子 主题：正确理解ThreadLocal
该帖已经被评为良好帖

作者 正文

```
46.     * variable.  Creates and initializes the copy if this is the first time
47.     * the thread has called this method.
48.     *
49.     * @return the current thread's value of this thread-local
50.     */
51.    public T get() {
52.        Thread t = Thread.currentThread();
53.        ThreadLocalMap map = getMap(t);
54.        if (map != null)
55.            return (T)map.get(this);
56.
57.        // Maps are constructed lazily.  if the map for this thread
58.        // doesn't exist, create it, with this ThreadLocal and its
59.        // initial value as its only entry.
60.        T value = initialValue();
61.        createMap(t, value);
62.        return value;
63.    }
64.
65.    /**
66.     * Sets the current thread's copy of this thread-local variable
67.     * to the specified value.  Many applications will have no need for
68.     * this functionality, relying solely on the {@link #initialValue}
69.     * method to set the values of thread-locals.
70.     *
71.     * @param value the value to be stored in the current threads' copy of
72.     *        this thread-local.
73.     */
74.    public void set(T value) {
75.        Thread t = Thread.currentThread();
76.        ThreadLocalMap map = getMap(t);
77.        if (map != null)
78.            map.set(this, value);
79.        else
80.            createMap(t, value);
81.    }
82.
83.    /**
84.     * Get the map associated with a ThreadLocal. Overridden in
85.     * InheritableThreadLocal.
86.     *
87.     * @param  t the current thread
88.     * @return the map
89.     */
90.    ThreadLocalMap getMap(Thread t) {
91.        return t.threadLocals;
92.    }
93.
94.    /**
95.     * Create the map associated with a ThreadLocal. Overridden in
96.     * InheritableThreadLocal.
97.     *
98.     * @param t the current thread
99.     * @param firstValue value for the initial entry of the map
100.    * @param map the map to store.
101.    */
102.   void createMap(Thread t, T firstValue) {
103.       t.threadLocals = new ThreadLocalMap(this, firstValue);
104.   }
105.
106.   .......
107.
108.   /**
109.    * ThreadLocalMap is a customized hash map suitable only for
110.    * maintaining thread local values. No operations are exported
```

锁定老帖子 主题：正确理解ThreadLocal
该帖已经被评为良好帖

作者 正文

```
111.     * outside of the ThreadLocal class. The class is package private to
112.     * allow declaration of fields in class Thread.  To help deal with
113.     * very large and long-lived usages, the hash table entries use
114.     * WeakReferences for keys. However, since reference queues are not
115.     * used, stale entries are guaranteed to be removed only when
116.     * the table starts running out of space.
117.     */
118.    static class ThreadLocalMap {
119.
120.    ........
121.
122.    }
123.
124. }
```

可以看到ThreadLocal类中的变量只有这3个int型：
Java代码

```
1. private final int threadLocalHashCode = nextHashCode();
2. private static int nextHashCode = 0;
3. private static final int HASH_INCREMENT = 0x61c88647;
```

而作为ThreadLocal实例的变量只有 threadLocalHashCode 这一个，nextHashCode 和HASH_INCREMENT 是ThreadLocal类的静态变量，实际上HASH_INCREMENT是一个常量，表示了连续分配的两个ThreadLocal实例的threadLocalHashCode值的增量，而nextHashCode 的表示了即将分配的下一个ThreadLocal实例的threadLocalHashCode 的值。

可以来看一下创建一个ThreadLocal实例即new ThreadLocal()时做了哪些操作，从上面看到构造函数ThreadLocal()里什么操作都没有，唯一的操作是这句：
Java代码

```
1. private final int threadLocalHashCode = nextHashCode();
```

那么nextHashCode()做了什么呢：
Java代码

```
1. private static synchronized int nextHashCode() {
2.     int h = nextHashCode;
3.     nextHashCode = h + HASH_INCREMENT;
4.     return h;
5. }
```

就是将ThreadLocal类的下一个hashCode值即nextHashCode的值赋给实例的threadLocalHashCode，然后nextHashCode的值增加HASH_INCREMENT这个值。

因此ThreadLocal实例的变量只有这个threadLocalHashCode，而且是final的，用来区分不同的ThreadLocal实例，ThreadLocal类主要是作为工具类来使用，那么ThreadLocal.set()进去的对象是放在哪儿的呢？

看一下上面的set()方法，两句合并一下成为
Java代码

```
1. ThreadLocalMap map = Thread.currentThread().threadLocals;
```

这个ThreadLocalMap 类是ThreadLocal中定义的内部类，但是它的实例却用在Thread类中：
Java代码

```
1. public class Thread implements Runnable {
2.     ......
3.
4.     /* ThreadLocal values pertaining to this thread. This map is maintained
5.      * by the ThreadLocal class. */
6.     ThreadLocal.ThreadLocalMap threadLocals = null;
7.     ......
```

锁定老帖子 主题：正确理解ThreadLocal
该帖已经被评为良好帖

作者 正文

```
8. }
```

再看这句：
Java代码

```
1. if (map != null)
2.     map.set(this, value);
```

也就是将该ThreadLocal实例作为key，要保持的对象作为值，设置到当前线程的ThreadLocalMap 中，get()方法同样大家看了代码也就明白了，ThreadLocalMap 类的代码太多了，我就不帖了，自己去看源码吧。

写了这么多，也不知讲明白了没有，有什么不当的地方还请大家指出来。

推荐链接

返回顶楼

- liangguanhui
- 等级:
- 
- 性别:
- 文章: 242
- 积分: 308
- 我现在离线

发表时间：2007-07-23
引用
或者可以自己定义一个静态的map，将当前thread作为key，创建的session作为值，put到map中，应该也行，这也是一般人的想法，但事实上，ThreadLocal的实现刚好相反，它是在每个线程中有一个map，而将ThreadLocal实例作为key，这样每个map中的项数很少，而且当线程销毁时相应的东西也一起销毁了，不知道除了这些还有什么其他的好处。

其实在jdk1.4之前的ThreadLocal的实现就是类似第一种情况的实现，jdk1.4就改成后面那种实现。
至于好处，除了可以自动释放外，还有一个很重要的好处：速度快了很多。

返回顶楼 回帖地址
0 0 请登录后投票

发表时间：2007-07-23
liangguanhui 写道
引用
或者可以自己定义一个静态的map，将当前thread作为key，创建的session作为值，put到map中，应该也行，这也是一般人的想法，但事实上，ThreadLocal的实现刚好相反，它是在每个线程中有一个map，而将ThreadLocal实例作为key，这样每个map中的项数很少，而且当线程销毁时相应的东西也一起销毁了，不知道除了这些还有什么其他的好处。

其实在jdk1.4之前的ThreadLocal的实现就是类似第一种情况的实现，jdk1.4就改成后面那种实现。
至于好处，除了可以自动释放外，还有一个很重要的好处：速度快了很多。

- lujh99
- 等级:
- 文章: 74
- 积分: 165
- 来自: ...
- 我现在离线

不是啊，我看了一下jdk1.3的源码，这一点上和1.4、1.5是一样的，map都是放在每个线程中的，以threadLocal为key，所不同的是，在1.3中这个map是个普通的HashMap，而1.4和1.5中是个ThreadLocalMap类，一个明显的特征是其中的Entry用到了弱引用WeakReference 类，但我感觉这个WeakReference的用法在这里并不能起到应有的作用。

另外想到一个把map放到各自线程中带来的好处是 因为各线程访问的map是各自不同的map，所以不需要同步，速度会快些；而如果把所有线程要用的对象都放到一个静态map中的话 多线程并发访问需要进行同步。

返回顶楼 回帖地址
1 0 请登录后投票

锁定老帖子 主题：正确理解ThreadLocal
该帖已经被评为良好帖

作者 正文

- liangguanhui
- 等级: ☆☆☆

发表时间：2007-08-21
可能是我搞错了，是1.3的时候改写的

- 性别: ♂
- 文章: 242
- 积分: 308
- 我现在离线

返回顶楼 回帖地址
0 0 请登录后投票

- hax
- 等级: 💎💎💎💎

发表时间：2007-08-21
lujh99 写道
另外想到一个把map放到各自线程中带来的好处是 因为各线程访问的map是各自不同的map，所以不需要同步，速度会快些；而如果把所有线程要用的对象都放到一个静态map中的话 多线程并发访问需要进行同步。

- 性别: ♂
- 文章: 1543
- 积分: 1795
- 来自: 上海
- 我现在离线

不是快一点两点的，jdk 1.3之前的实现是非常之慢的!

返回顶楼 回帖地址
0 0 请登录后投票

- SINCE1978
- 等级: 初级会员

发表时间：2007-11-16
我想知道如果多次new ThreadLocal并且调用其set方法的话、是否就和普通hashmap一样后set进去的会覆盖先set进去的？这样的话ThreadLocal只能植入一个资源喽？这肯定不对，否则还用ThreadLocalMap这个自定义哈希表干什么，那么如何区分一个线程当中不同方法或不同类set进去的资源？并正确set和get？？

- 性别: ♂
- 文章: 55
- 积分: 40
- 来自: 济南
- 我现在离线

返回顶楼 回帖地址
0 2 请登录后投票

- jieyuan_cg
- 等级: 初级会员

发表时间：2008-04-02
SINCE1978 写道
我想知道如果多次new ThreadLocal并且调用其set方法的话、是否就和普通hashmap一样后set进去的会覆盖先set进去的？这样的话ThreadLocal只能植入一个资源喽？这肯定不对，否则还用ThreadLocalMap这个自定义哈希表干什么，那么如何区分一个线程当中不同方法或不同类set进去的资源？并正确set和get？？

- 性别: ♂
- 文章: 111
- 积分: 50
- 来自: 北京
- 我现在离线

每个ThreadLocal当然只能放一个对象。要是需要放其他的对象，就再new 一个新的ThreadLocal出来，这个新的ThreadLocal作为key,需要放的对象作为value，放在ThreadLocalMap中。。。。

锁定老帖子 主题：正确理解ThreadLocal
该帖已经被评为良好帖

作者 正文

返回顶楼 回帖地址
0 0 请登录后投票

- jieyuan_cg
- 等级: 初级会员
- 
- 性别:
- 文章: 111
- 积分: 50
- 来自: 北京
- 我现在离线

发表时间：2008-04-02
LZ的这篇对ThreadLocal的解释真到位。。。多谢！
坛子里面有几篇对ThreadLocal的解释好像都有些误解。。。看得晕晕乎乎。

返回顶楼 回帖地址
0 0 请登录后投票

- elice
- 等级: 初级会员
- 
- 性别:
- 文章: 21
- 积分: 0
- 来自: 武汉
- 我现在离线

发表时间：2008-05-22
主要是提供了保持对象的方法和避免参数传递的方便的对象访问方式

返回顶楼 回帖地址
0 0 请登录后投票

- csrcom
- 等级:
-
- 性别:
- 文章: 75
- 积分: 610
- 来自: 杭州
- 我现在离线

发表时间：2008-05-22
其实可以在thread local存放更丰富的对象比如Map，那么就不用实例化那么thread local了

Java代码

```
1. package com.meidusa.amoeba.util;
2.
3. import java.util.HashMap;
4. import java.util.Map;
5.
6. import org.apache.log4j.Logger;
7. /**
8.  * ThreadLocal Context
9.  * @author <a href=mailto:piratebase@sina.com>Struct chen</a>
10.  * @version $Id: ThreadLocalContext.java 3597 2006-11-23 08:11:58Z struct $
11.  */
12. public class ThreadLocalMap{
13.     private static Logger logger = Logger.getLogger(ThreadLocalMap.class);
14.
15.     protected final static ThreadLocal<Map<String,Object>> threadContext = new MapThreadLocal();
16.
17.     private ThreadLocalMap(){};
18.
19.     public static void put(String key,Object value){
20.         getContextMap().put(key,value);
21.     }
22.
23.     public static Object remove(String key){
24.         return getContextMap().remove(key);
```

锁定老帖子 主题：正确理解ThreadLocal
该帖已经被评为良好帖

作者 正文

```
25.  }
26.
27.  public static Object get(String key){
28.      return getContextMap().get(key);
29.  }
30.
31.  private static class MapThreadLocal extends ThreadLocal<Map<String,Object>> {
32.      protected Map<String,Object> initialValue() {
33.          return new HashMap<String,Object>() {
34.
35.              private static final long serialVersionUID = 3637958959138295593L;
36.
37.              public Object put(String key, Object value) {
38.                  if (logger.isDebugEnabled()) {
39.                      if (containsKey(key)) {
40.                          logger.debug("Overwritten attribute to thread context: " + key
41.                                  + " = " + value);
42.                      } else {
43.                          logger.debug("Added attribute to thread context: " + key + " = "
44.                                  + value);
45.                      }
46.                  }
47.
48.                  return super.put(key, value);
49.              }
50.          };
51.      }
52.  }
53.
54.  /**
55.   * 取得thread context Map的实例。
56.   *
57.   * @return thread context Map的实例
58.   */
59.  protected static Map<String,Object> getContextMap() {
60.      return (Map<String,Object>) threadContext.get();
61.  }
62.
63.
64.  /**
65.   * 清理线程所有被hold住的对象。以便重用!
66.   */
67.
68.  public static void reset(){
69.      getContextMap().clear();
70.  }
71. }
```

返回顶楼 回帖地址
1 0 请登录后投票

« 上一页 1 2 下一页 »
论坛首页 → Java企业应用版

跳转论坛: Java企业应用

- ITeye首页
- 资讯
- 精华
- 论坛
- 问答
- 博客
- 专栏
- 群组
- 下载

java 搜索

- 广告服务