链滴

ThreadLocal 源码分析

作者：xumiao

原文链接：https://ld246.com/article/1587952527795

来源网站：链滴

ThreadLocal是什么?

顾名思义，ThreadLocal可以为每个线程独立存储不同的值（线程本地变量）。其意义在于高并发场下变量被多个线程访问互不影响，有效避免线程安全问题和同步带来的性能开销。当然它也存在一定陷，由于每个线程都会创建ThreadLocal变量，就会带来一定的内存消耗；它的思想就是“以空间换间”。

特殊情况：InhertiableThreadLocal并不是只存储当前线程的值，它默认会集成父类中的值。

ThreadLocal类方法定义

```
public class ThreadLocal<T> {
    public T get();
    private T setInitialValue();
    public void set(T value);
    public void remove();
}
```

- **get**：用于获取当前线程私有的ThreadLocal变量；
- **setInitialValue**：可以进行重写设置当前ThreadLocal对应的数据，用于第一次调用get时懒加载取；
- **set**：设置当前Thread的ThreadLocal值；
- **remove**：删除ThreadLocal中的数据；

---

get方法具体实现

```
/**
 * Returns the value in the current thread's copy of this
 * thread-local variable.  If the variable has no value for the
 * current thread, it is first initialized to the value returned
 * by an invocation of the {@link #initialValue} method.
 * @return the current thread's value of this thread-local
 */
public T get() {
    //获取当前线程
    Thread t = Thread.currentThread();
    //获取Thread类中定义的ThreadLocal.ThreadLocalMap变量
    ThreadLocalMap map = getMap(t);
    //判断map是否为null
    if (map != null) {
        //获取当前ThreadLocal对应的Entry
        ThreadLocalMap.Entry e = map.getEntry(this);
        if (e != null) {
            //获取set的值
            @SuppressWarnings("unchecked")
            T result = (T) e.value;
            return result;
        }
```

```
        }
        //当map为null或未设置值时调用
        return setInitialValue();
    }
```

其中ThreadLocalMap是ThreadLocal中的静态内部类，内部维护了一个Entry数组的table，可以看是一个kv的map，只不过key是当前ThreadLocal生成的Hash值；

```
static class ThreadLocalMap {

    private Entry[] table;

    static class Entry extends WeakReference<ThreadLocal<?>> {
            /** The value associated with this ThreadLocal. */
            Object value;

            Entry(ThreadLocal<?> k, Object v) {
                super(k);
                value = v;
            }
        }
}
```

setInitialValue具体实现

当get方法中map为null或未设置值时会调用setInitialValue方法，接下来看看它的实现：

```
    /**
     * Variant of set() to establish initialValue. Used instead
     * of set() in case user has overridden the set() method.
     * @return the initial value
     */
    private T setInitialValue() {
        //value等于重写initialValue后返回的值，否则默认返回null
        T value = initialValue();
        //获取当前线程
        Thread t = Thread.currentThread();
        //获取当前Thread类中定义的ThreadLocal.ThreadLocalMap变量
        ThreadLocalMap map = getMap(t);
        //map为null直接添加到table中，否则新建map
        if (map != null)
            map.set(this, value);
        else
            createMap(t, value);
        return value;
    }
```

值得注意的是其中initialValue默认是返回null的，当然你也可以选择重写来懒加载数据，当调用get法的时候才获取，就像这样：

```
ThreadLocal<Integer> threadLocal = new ThreadLocal<Integer>() {
    int i = 1;

    @Override
```

```
    protected Integer initialValue() {
        return i;
    }
};
```

其中当map为null是会调用createMap方法，将创建的ThreadLocalMap赋值到当前Thread的threadocals变量，完成关联！

```
/**
     * Create the map associated with a ThreadLocal. Overridden in
     * InheritableThreadLocal.
     * @param t
     *        the current thread
     * @param firstValue
     *        value for the initial entry of the map
     */
    void createMap(Thread t, T firstValue) {
        t.threadLocals = new ThreadLocalMap(this, firstValue);
    }

public
class Thread implements Runnable {
    /* ThreadLocal values pertaining to this thread. This map is maintained
     * by the ThreadLocal class. */
    ThreadLocal.ThreadLocalMap threadLocals = null;
}
```

set方法具体实现

和上面setInitialValue中实现几乎一样，这里就不赘述了...

```
    /**
     * Sets the current thread's copy of this thread-local variable
     * to the specified value.  Most subclasses will have no need to
     * override this method, relying solely on the {@link #initialValue}
     * method to set the values of thread-locals.
     * @param value
     *        the value to be stored in the current thread's copy of
     *        this thread-local.
     */
    public void set(T value) {
        //获取当前Thread
        Thread t = Thread.currentThread();
        //获取当前Thread类中定义的ThreadLocal.ThreadLocalMap变量
        ThreadLocalMap map = getMap(t);
        //map为null直接添加到table中，否则新建map
        if (map != null)
            map.set(this, value);
        else
            createMap(t, value);
    }
```

remove方法具体实现

获取当前线程中的ThreadLocal.ThreadLocalMap，清除table数组中以当前ThreadLocal Hash为ke的数据。

```
/**
 * Removes the current thread's value for this thread-local
 * variable.  If this thread-local variable is subsequently
 * {@linkplain #get read} by the current thread, its value will be
 * reinitialized by invoking its {@link #initialValue} method,
 * unless its value is {@linkplain #set set} by the current thread
 * in the interim.  This may result in multiple invocations of the
 * {@code initialValue} method in the current thread.
 * @since 1.5
 */
public void remove() {
    //获取当前Thread类中定义的ThreadLocal.ThreadLocalMap变量
    ThreadLocalMap m = getMap(Thread.currentThread());
    if (m != null)
        //删除以当前ThreadLocal为key设置的数据
        m.remove(this);
}
```

关于内存泄露的问题

抛出问题

- 首先ThreadLocal实例被线程的ThreadLocalMap实例持有，也可以看成被线程持有。
- 如果应用使用了线程池，那么之前的线程实例处理完之后出于复用的目的依然存活

所以，ThreadLocal设定的值被持有，导致内存泄露。

结论

首先Entry是继承WeakReference<ThreadLocal<?>>的，也就是弱引用，ThreadLocalMap的key用的是ThreadLocal的弱引用，所以并不会导致内存泄露，关于java中的四种引用我会在后面的文章记录。

最后

第一次写源码分析，有不好的地方欢迎指正。

参考资料

https://blog.appoptics.com/introduction-to-java-threadlocal-storage/

https://github.com/decaywood/decaywood.github.io/blob/master/_posts/2016/3/2016-03-15 ThreadLocal-intro.markdown

https://droidyue.com/blog/2016/03/13/learning-threadlocal-in-java/

https://fangjian0423.github.io/2014/11/22/java_threadlocal/