在java多线程开发中ThreadLocal（线程本地变量）是经常用到的内容，ThreadLocal在每个线程中都创建了变量的副本，实现了多线程中变量的隔离，在多线程环境下保证了成员变量的安全。这次让我们一起研究下ThreadLocal吧。

1、ThreadLocal的简单应用

2、ThreadLocal源码简述

3、ThreadLocalMap的结构和存取

4、解决ThreadLocal内存泄漏

第1、2两部分讲解很简单，一定要仔细看第3部分内容，这是这篇文章的核心，也是ThreadLocal最需要掌握的部分。

1、ThreadLocal简单应用

让我们通过一个简单的例子看下ThreadLocal是怎么用的吧

@Slf4j  
public class MgDemo20200628 implements Runnable {  
  
 private static ThreadLocal<String> *local*=new ThreadLocal<>();  
 @Override  
 public void run() {  
 try {  
 *local*.set(Thread.*currentThread*().getName() +"---" + System.*currentTimeMillis*());  
 Thread.*sleep*(1000\*5);  
 *log*.info(*local*.get());  
 } catch (InterruptedException e) {  
 e.printStackTrace();  
 }  
 }  
 public static void main(String[] args) {  
 MgDemo20200628 demo =new MgDemo20200628();  
 for(int i=0;i<3;i++)  
 {  
 Thread task = new Thread(demo);  
 task.start();  
 }  
 }  
}

运行一下，控制台的输出结果

22:37:17.293 [Thread-1] INFO com.mg.empty.demo.thread.threadlocal.MgDemo20200628 - Thread-1---1593355032290

22:37:17.293 [Thread-2] INFO com.mg.empty.demo.thread.threadlocal.MgDemo20200628 - Thread-2---1593355032290

22:37:17.293 [Thread-0] INFO com.mg.empty.demo.thread.threadlocal.MgDemo20200628 - Thread-0---1593355032290

上边代码的执行结果可以看到，每个线程通过ThreadLocal获取到的String都是自己线程副本的内容，各线程间有很好的隔离。

注意：ThreadLocal通常都定义为static，因为ThreadLocal实际上没有存储功能，变量副本的真实存储位置是Thread对象的threadLocals这个ThreadLocal.ThreadLocalMap中，可以将ThreadLocal理解为一个工具类，用来保证线程本地变量的存储和存储碰撞。

让我们一块研究下ThreadLocal的具体实现，来更深入了解下。

二、ThreadLocal源码简述

ThreadLocal中常用的方法有get、set和remove，这几个函数的源码很简单

public T get() {  
 Thread t = Thread.*currentThread*();  
 ThreadLocalMap map = getMap(t);  
 if (map != null) {  
 ThreadLocalMap.Entry e = map.getEntry(this);  
 if (e != null) {  
 @SuppressWarnings("unchecked")  
 T result = (T)e.value;  
 return result;  
 }  
 }  
 return setInitialValue();  
}

public void set(T value) {  
 Thread t = Thread.*currentThread*();  
 ThreadLocalMap map = getMap(t);  
 if (map != null)  
 map.set(this, value);  
 else  
 createMap(t, value);  
}

public void remove() {  
 ThreadLocalMap m = getMap(Thread.*currentThread*());  
 if (m != null)  
 m.remove(this);  
}

get、set和remove内容都很简单，都是先调用getMap这个函数来获取ThreadLocalMap对象，然后操作这个map对象来实现变量的存取。看下getMap这个函数

ThreadLocalMap getMap(Thread t) {  
 return t.threadLocals;  
}

这个是获取Thread对象中的threadLocals变量，这个变量是ThreadLocal.ThreadLocalMap类型的。ThreadLocalMap的真实存储位置是在每个线程中，不是在ThreadLocal对象中。一定要记住ThreadLocalMap是存储在Thread中，这个是理解ThreadLocal原理很重要的基础。

让我们一块研究下ThreadLocalMap这个内部类，来一块揭晓一下ThreadLocal的真正神秘所在。

二、ThreadLocalMap的结构和存取

在研究ThreadLocal之前，我们先研究下ThreadLocalMap这个内部类，这是变量数据存储的地方。

在研究这个类之前，一定要弄明白 ThreadLocalMap的真实存储位置是在每个线程中，不是在ThreadLocal对象中，这是理解ThreadLocalMap设计思路的前提。如果是在不理解可以先看下第三章中getMap这个函数。