dreamcatcher-cx

why is more important than what.

谈谈Java中的ThreadLocal

ThreadLocal介绍&跳出误区

看看源码

线程独享变量?

ThreadLocal介绍&跳出误区

ThreadLocal一般称为**线程本地变量**，它是一种特殊的线程绑定机制，将变量与线程**绑定**在一起，为每一个线程维护一个独立的变量副本。通过ThreadLocal可以将对象的可见范围限制在同一个线程内。

跳出误区

需要重点强调的的是，不要拿ThreadLocal和synchronized做类比，因为这种比较压根就是无意义的！sysnchronized是一种互斥同步机制，是为了保证在多线程环境下对于共享资源的正确访问。而ThreadLocal从本质上讲，无非是提供了一个**“线程级”**的**变量作用域**，它是一种**线程封闭**（每个线程独享变量）技术，更直白点讲，ThreadLocal可以理解为将对象的作用范围限制在一个**线程上下文**中，使得变量的作用域为**“线程级”**。

没有ThreadLocal的时候，一个线程在其声明周期内，可能穿过多个层级，多个方法，如果有个对象需要在此线程周期内多次调用，且是跨层级的（线程内共享），通常的做法是通过参数进行传递；而ThreadLocal将变量绑定在线程上，在一个线程周期内，无论“你身处何地”，只需通过其提供的get方法就可轻松获取到对象。极大地提高了对于“线程级变量”的访问便利性。

来看个简单的例子

假设我们要为每个线程关联一个唯一的序号，在每个线程周期内，我们需要多次访问这个序号，这时我们就可以使用ThreadLocal了.（当然下面这个例子没有完全体现出跨层级跨方法的调用，理解就可以了）

```
package concurrent;

import java.util.concurrent.atomic.AtomicInteger;

/**
 * Created by chengxiao on 2016/12/12.
 */
public class ThreadLocalDemo {
    public static void main(String []args){
        for(int i=0;i<5;i++){
            final Thread t = new Thread(){
                @Override
                public void run(){
                    System.out.println("当前线程:"+Thread.currentThread().getName()+",已分配ID:"+ThreadId.get());
                }
            };
            t.start();
        }
    }
    static   class ThreadId{
```

```
        //一个递增的序列，使用AtomicInger原子变量保证线程安全
        private static final AtomicInteger nextId = new AtomicInteger(0);
        //线程本地变量，为每个线程关联一个唯一的序号
        private static final ThreadLocal<Integer> threadId =
                new ThreadLocal<Integer>() {
                    @Override
                    protected Integer initialValue() {
                        return nextId.getAndIncrement();//相当于nextId++,由于nextId++这种操作是个复合操作而非原子操作，会有线程安全问题(可能在初始化时就获取到相同的ID，所以使用原子变量
                    }
                };

       //返回当前线程的唯一的序列，如果第一次get，会先调用initialValue，后面看源码就了解了
        public static int get() {
            return threadId.get();
        }
    }
}
```

执行结果,可以看到每个线程都分配到了一个唯一的ID，同时在此线程范围内的"任何地点"，我们都可以通过ThreadId.get()这种方式直接获取。

```
当前线程:Thread-4,已分配ID:1
当前线程:Thread-0,已分配ID:0
当前线程:Thread-2,已分配ID:3
当前线程:Thread-1,已分配ID:4
当前线程:Thread-3,已分配ID:2
```

看看源码

set操作，为线程绑定变量

```
 public void set(T value) {
    Thread t = Thread.currentThread();//1.首先获取当前线程对象
        ThreadLocalMap map = getMap(t);//2.获取该线程对象的ThreadLocalMap
        if (map != null)
            map.set(this, value);//如果map不为空，执行set操作，以当前threadLocal对象为key，实际存储对象为value进行set操作
        else
            createMap(t, value);//如果map为空，则为该线程创建ThreadLocalMap
    }
```

可以看到，ThreadLocal不过是个入口，真正的变量是绑定在线程上的。

```
ThreadLocalMap getMap(Thread t) {
    return t.threadLocals;//线程对象持有ThreadLocalMap的引用
}
```

下面给是Thread类中的定义，每个线程对象都拥有一个ThreadLocalMap对象

```
    ThreadLocal.ThreadLocalMap threadLocals = null;
```

现在，我们能看出ThreadLocal的设计思想了：

1.ThreadLocal仅仅是个变量访问的入口；

2.每一个Thread对象都有一个ThreadLocalMap对象，这个ThreadLocalMap持有对象的引用；

3.ThreadLocalMap以当前的threadlocal对象为key，以真正的存储对象为value。get时通过threadlocal实例就可以找到绑定在当前线程上的对象。

乍看上去，这种设计确实有些绕。我们完全可以在设计成Map<Thread,T>这种形式，一个线程对应一个存储对象。ThreadLocal这样设计的目的主要有两个：

一是可以保证当前线程结束时相关对象能尽快被回收；二是ThreadLocalMap中的元素会大大减少，我们都知道map过大更容易造成哈希冲突而导致性能变差。

我们再来看看get方法

Oracle(4)
并发编程(8)
分布式系统
数据结构(2)
算法(5)

随笔档案(20)
2017年7月 (2)
2017年6月 (1)
2017年5月 (2)
2017年4月 (1)
2017年3月 (1)
2017年2月 (1)
2017年1月 (1)
2016年12月 (3)
2016年11月 (4)
2016年10月 (2)
2016年9月 (2)

积分与排名
积分 - 45393
排名 - 8329

最新评论
1. Re:HashMap实现原
@云淡wy谢谢指正，改
--dr
2. Re:HashMap实现原
如果定位到的数组包含链
作，其时间复杂度依然为
的Entry会插入链表头部

```
public T get() {
    Thread t = Thread.currentThread();//1.首先获取当前线程
        ThreadLocalMap map = getMap(t);//2.获取线程的map对象
        if (map != null) {//3.如果map不为空，以threadlocal实例为key获取到对应Entry，然后从Entry中取出对象即可。
            ThreadLocalMap.Entry e = map.getEntry(this);
            if (e != null)
                return (T)e.value;
        }
        return setInitialValue();//如果map为空，也就是第一次没有调用set直接get（或者调用过set，又调用了remove）时，为其设定初始值
    }
```

setInitialValue

```
1  private T setInitialValue() {
2          T value = initialValue();//获取初始值
3          Thread t = Thread.currentThread();
4          ThreadLocalMap map = getMap(t);
5          if (map != null)
6              map.set(this, value);
7          else
8              createMap(t, value);
9          return value;
10     }
```

initialValue方法，默认是null，访问权限是protected，即允许重写。

```
1 protected T initialValue() {
2          return null;
3      }
```

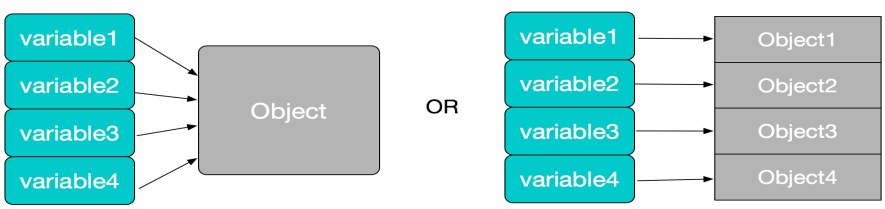
谈到这儿，我们应该已经对ThreadLocal的设计目的及设计思想有一定的了解了。

线程独享变量?

还有一个会引起疑惑的问题，我们说ThreadLocal为每一个线程维护一个独立的**变量副本**，那么是不是说各个线程之间真正的做到对于对象的**“完全自治”**而不对其他线程的对象产生影响呢？其实这已经不属于对于ThreadLocal的讨论，而是你出于何种目的去使用ThreadLocal。如果我们为一个线程关联的对象是“完全独享”的，也就是每个线程拥有一整套的新的 **栈中的对象引用+堆中的对象，**那么这种情况下是真正的彻底的“**线程独享变量**”，相当于一种深度拷贝，每个线程自己玩自己的，对该对象做任何的操作也不会对别的线程有任何影响。

另一种更普遍的情况，所谓的独享变量副本，其实也就是每个线程都拥有一个独立的对象引用，而堆中的对象还是**线程间共享**的，这种情况下，自然还是会涉及到对共享资源的访问操作，依然会有线程不安全的风险。所以说，ThreadLocal无法解决线程安全问题。

所以，需不需要完全独享变量，进行完全隔离，就取决于你的应用场景了。可以想象，对象过大的时候，如果每个线程都有这么一份“深拷贝”，并发又比较大，对于服务器的压力自然是很大的。像web开发中的servlet，servlet是线程不安全的，一请求一线程，多个线程共享一个servlet对象；而早期的CGI设计中，N个请求就对应N个对象，并发量大了之后性能自然就很差。



ThreadLocal在spring的事务管理，包括Hibernate的session管理等都有出现，在web开发中，有时会用来管理用户会话 HttpSession，web交互中这种典型的一请求一线程的场景似乎比较适合使用ThreadLocal，但是需要特别注意的是，

变引用链即可

这个是不是写错了,急是

3. Re:图解排序算法(五

三数取中法

楼主，文章写的很棒，if

以减少一次交换吧。if (i

wap(arr, i, right - 1);

4. Re:HashMap实现原

@大脸喵感谢支持哈...

--dr

5. Re:HashMap实现原

博主，你好，在分析tab

是2的N次方的时，个人

-->hashcode------>h

x，整体上这是一个多对

----->h.....

阅读排行榜

1. 图解排序算法(一)之3

择，冒泡，直接插入)(7

2. HashMap实现原理及

3)

3. 图解排序算法(三)之堆

4. 图解排序算法(二)之希

5. 图解排序算法(四)之归

评论排行榜

1. HashMap实现原理及

2. 图解排序算法(一)之3

择，冒泡，直接插入)(1

3. 图解排序算法(四)之归

4. 图解排序算法(三)之堆

5. 图解排序算法(二)之希