leizhimin 的BLOG

写留言 去学院学习

发消息 加友情链接

进家园 加好友

原创 深入研究java.lang.ThreadLocal类

2007-11-23 11:50:51

标签：ThreadLocal java 类

深入研究java.lang.ThreadLocal类

一、概述

ThreadLocal是什么呢？其实ThreadLocal并非是一个线程的本地实现版本，它并不是一个Thread，而是threadlocalvariable(线程局部变量)。也许把它命名为ThreadLocalVar更加合适。线程局部变量(ThreadLocal)其实的功用非常简单，就是为每一个使用该变量的线程都提供一个变量值的副本，是Java中一种较为特殊的线程绑定机制，是每一个线程都可以独立地改变自己的副本，而不会和其它线程的副本冲突。

从线程的角度看，每个线程都保持一个对其线程局部变量副本的隐式引用，只要线程是活动的并且 ThreadLocal 实例是可访问的；在线程消失之后，其线程局部实例的所有副本都会被垃圾回收（除非存在对这些副本的其他引用）。

通过ThreadLocal存取的数据，总是与当前线程相关，也就是说，JVM 为每个运行的线程，绑定了私有的本地实例存取空间，从而为多线程环境常出现的并发访问问题提供了一种隔离机制。

ThreadLocal是如何做到为每一个线程维护变量的副本的呢？其实实现的思路很简单，在ThreadLocal类中有一个Map，用于存储每一个线程的变量的副本。

概括起来说，对于多线程资源共享的问题，同步机制采用了“以时间换空间”的方式，而ThreadLocal采用了“以空间换时间”的方式。前者仅提供一份变量，让不同的线程排队访问，而后者为每一个线程都提供了一份变量，因此可以同时访问而互不影响。

二、API说明

ThreadLocal()

创建一个线程本地变量。

T get()

返回此线程局部变量的当前线程副本中的值，如果这是线程第一次调用该方法，则创建并初始化此副本。

- Java线程：创建与启动
- Java线程：线程的同步与锁

protected T initialValue()

返回此线程局部变量的当前线程的初始值。最多在每次访问线程来获得每个线程局部变量时调用此方法一次，即线程第一次使用 get() 方法访问变量的时候。如果线程先于 get 方法调用 set(T) 方法，则不会在线程中再调用 initialValue 方法。

若该实现只返回 null；如果程序员希望将线程局部变量初始化为 null 以外的某个值，则必须为 ThreadLocal 创建子类，并重写此方法。通常，将使用匿名内部类。initialValue 的典型实现将调用一个适当的构造方法，并返回新构造的对象。

void remove()

移除此线程局部变量的值。这可能有助于减少线程局部变量的存储需求。如果再次访问此线程局部变量，那么在默认情况下它将拥有其 initialValue。

void set(T value)

将此线程局部变量的当前线程副本中的值设置为指定值。许多应用程序不需要这项功能，它们只依赖于 initialValue() 方法来设置线程局部变量的值。

在程序中一般都重写initialValue方法，以给定一个特定的初始值。

三、典型实例

1、Hiberante的Session 工具类HibernateUtil

这个类是Hibernate官方文档中HibernateUtil类，用于session管理。

```
public class HibernateUtil {
    private static Log log = LogFactory.getLog(HibernateUtil.class);
    private static final SessionFactory sessionFactory;     //定义SessionFactory

    static {
        try {
            // 通过默认配置文件hibernate.cfg.xml创建SessionFactory
            sessionFactory = new Configuration().configure().buildSessionFactory();
        } catch (Throwable ex) {
            log.error("初始化SessionFactory失败！", ex);
            throw new ExceptionInInitializerError(ex);
        }
    }

    //创建线程局部变量session，用来保存Hibernate的Session
    public static final ThreadLocal session = new ThreadLocal();

    /**
     * 获取当前线程中的Session
     * @return Session
     * @throws HibernateException
     */
    public static Session currentSession() throws HibernateException {
        Session s = (Session) session.get();
        // 如果Session还没有打开，则新开一个Session
        if (s == null) {
```

```
            s = sessionFactory.openSession();
            session.set(s);          //将新开的Session保存到线程局部变量中
        }
        return s;
    }

    public static void closeSession() throws HibernateException {
        //获取线程局部变量，并强制转换为Session类型
        Session s = (Session) session.get();
        session.set(null);
        if (s != null)
            s.close();
    }
}
```

在这个类中，由于没有重写ThreadLocal的initialValue()方法，则首次创建线程局部变量session其初始值为null，第一次调用currentSession()的时候，线程局部变量的get()方法也为null。因此，对session做了判断，如果为null，则新开一个Session，并保存到线程局部变量session中，这一步非常的关键，这也是“public static final ThreadLocal session = new ThreadLocal()”所创建对象session能强制转换为Hibernate Session对象的原因。

2、另外一个实例

创建一个Bean，通过不同的线程对象设置Bean属性，保证各个线程Bean对象的独立性。

```
/**
 * Created by IntelliJ IDEA.
 * User: leizhimin
 * Date: 2007-11-23
 * Time: 10:45:02
 * 学生
 */
public class Student {
    private int age = 0;   //年龄

    public int getAge() {
        return this.age;
    }

    public void setAge(int age) {
        this.age = age;
    }
}
```

```
/**
 * Created by IntelliJ IDEA.
 * User: leizhimin
 * Date: 2007-11-23
 * Time: 10:53:33
 * 多线程下测试程序
 */
public class ThreadLocalDemo implements Runnable {
    //创建线程局部变量studentLocal，在后面你会发现用来保存Student对象
```

```
    private final static ThreadLocal studentLocal = new ThreadLocal();

    public static void main(String[] agrs) {
        ThreadLocalDemo td = new ThreadLocalDemo();
        Thread t1 = new Thread(td, "a");
        Thread t2 = new Thread(td, "b");
        t1.start();
        t2.start();
    }

    public void run() {
        accessStudent();
    }

    /**
     * 示例业务方法，用来测试
     */
    public void accessStudent() {
        //获取当前线程的名字
        String currentThreadName = Thread.currentThread().getName();
        System.out.println(currentThreadName + " is running!");
        //产生一个随机数并打印
        Random random = new Random();
        int age = random.nextInt(100);
        System.out.println("thread " + currentThreadName + " set age to:" + age);
        //获取一个Student对象，并将随机数年龄插入到对象属性中
        Student student = getStudent();
        student.setAge(age);
        System.out.println("thread " + currentThreadName + " first read age is:" + student.getAge());
        try {
            Thread.sleep(500);
        }
        catch (InterruptedException ex) {
            ex.printStackTrace();
        }
        System.out.println("thread " + currentThreadName + " second read age is:" +
student.getAge());
    }

    protected Student getStudent() {
        //获取本地线程变量并强制转换为Student类型
        Student student = (Student) studentLocal.get();
        //线程首次执行此方法的时候，studentLocal.get()肯定为null
        if (student == null) {
            //创建一个Student对象，并保存到本地线程变量studentLocal中
            student = new Student();
            studentLocal.set(student);
        }
        return student;
    }
}
```

运行结果：

```
a is running!
thread a set age to:76
b is running!
thread b set age to:27
thread a first read age is:76
thread b first read age is:27
thread a second read age is:76
thread b second read age is:27
```

可以看到a、b两个线程age在不同时刻打印的值是完全相同的。这个程序通过妙用ThreadLocal，既实现多线程并发，游兼顾数据的安全性。

四、总结

ThreadLocal使用场合主要解决多线程中数据数据因并发产生不一致问题。ThreadLocal为每个线程的中并发访问的数据提供一个副本，通过访问副本来运行业务，这样的结果是耗费了内存，单大大减少了线程同步所带来性能消耗，也减少了线程并发控制的复杂度。

ThreadLocal不能使用原子类型，只能使用Object类型。ThreadLocal的使用比synchronized要简单得多。

ThreadLocal和Synchonized都用于解决多线程并发访问。但是ThreadLocal与synchronized有本质的区别。synchronized是利用锁的机制，使变量或代码块在某一时该只能被一个线程访问。而ThreadLocal为每一个线程都提供了变量的副本，使得每个线程在某一时间访问到的并不是同一个对象，这样就隔离了多个线程对数据的数据共享。而Synchronized却正好相反，它用于在多个线程间通信时能够获得数据共享。

Synchronized用于线程间的数据共享，而ThreadLocal则用于线程间的数据隔离。

当然ThreadLocal并不能替代synchronized,它们处理不同的问题域。Synchronized用于实现同步机制，比ThreadLocal更加复杂。

五、ThreadLocal使用的一般步骤

1、在多线程的类（如ThreadDemo类）中，创建一个ThreadLocal对象threadXxx，用来保存线程间需要隔离处理的对象xxx。
2、在ThreadDemo类中，创建一个获取要隔离访问的数据的方法getXxx()，在方法中判断，若ThreadLocal对象为null时候，应该new()一个隔离访问类型的对象，并强制转换为要应用的类型。
3、在ThreadDemo类的run()方法中，通过getXxx()方法获取要操作的数据，这样可以保证每个线程对应一个数据对象，在任何时刻都操作的是这个对象。

参考文档：

JDK 官方文档
[url]http://www.java3z.com/cwbwebhome/article/article2a/271.html?id=319[/url]
[url]http://www.java3z.com/cwbwebhome/article/article2/2952.html?id=1648[/url]

上一篇 Hibernate+Spring彻底搞定Clob、Blob的存取 下一篇 Java中的break和continue关键字使用总结

相关文章

- 谈谈java的类与对象
- Java中的Object类
- java nio buffer
- Java类加载原理解析

职位推荐

- Java后台高级工程师
- Java开发工程师
- Java高级工程师
- java后端负责人
- Java资深服务器开发工程师

本文收录至博客专题：《深入研究 Java.lang 的类》

文章评论

<< 1 2 >> 页数 (1/2)

[1楼] [匿名]51CTO游客 回复

2007-11-23 22:55:11

已被深深的吸引

[2楼] [匿名]51CTO游客 回复

2009-03-30 14:30:34

我测试了下 如果不调用getStudent（） 结果也没有区别么

[3楼] **danni505** 回复

2009-09-23 23:54:29

写的不错！
我也写了一遍，请指点http://danni505.blog.51cto.com/15547/204633

[4楼] **danni505** 回复

2009-09-23 23:57:33

利用ThreadLocal来做多线程环境下的管理器真是太适合了！

[5楼] [匿名]wavefly 回复

2009-09-28 11:48:06

貌似转载javaeye中的

[6楼] [匿名]... 回复

2009-11-09 10:46:29

是好资料 不过方法得注释翻译得真烂

[7楼] [匿名]51CTO游客 回复

2010-03-08 16:43:06

```
Student student = getStudent();
```
改成
```
Student student = new Student();
```

好像是没有什么区别，也能区分每个线程的值

[8楼] [匿名]阿哥 回复

2010-05-15 15:02:03

LZ根本就没有理解ThreadLocal

[9楼] [匿名]zdjray@163.com 回复

2010-10-29 17:00:02

8楼不要过早下结论
我毫不犹豫支持lz
lz抓住了其他我看过的绝大多数文章没有提到的步骤中的2.
2中的new其实是ThreadLocal的核心.
不得不说终于有明白人了.

[10楼] [匿名]andylau0927 回复

2010-12-16 19:55:31

学习了
例子写的真不错

[11楼] bigtiger 回复

2011-05-19 14:21:50

LZ或许明白了，但是还没说到点子上，或者说讲的不够明确。

[12楼] [匿名]123 回复

2011-08-17 11:46:10

其实LZ student只是方法级别的变量，还不是线程级别的变量，例子不好

[13楼] xiaoxi0324 回复

2011-10-13 13:35:48

12 楼正解
student 应该是线程级别的变量
楼主的方法内student不能诠释threadLocal 的作用
会误导别人

[14楼] [匿名]51CTO游客 回复

2012-05-23 15:16:10

其实你那个学生的例子是有问题的

[15楼] [匿名]51CTO游客 回复

2012-05-23 15:27:06

我给你改进一下吧　你的例子是有问题的其他都不变就是修改业务那个方法

```
/**
  * 示例业务方法，用来测试
  */
  public void accessStudent() {
    //获取当前线程的名字
    String currentThreadName = Thread.currentThread().getName();
    System.out.println(currentThreadName + " is running!");
    //产生一个随机数并打印
    Random random = new Random();
    int age = random.nextInt(100);
    System.out.println("thread " + currentThreadName + " set age to:" + age);
    //获取一个Student对象，并将随机数年龄插入到对象属性中
    getStudent().setAge(age);
    System.out.println("thread " + currentThreadName + " first read age is:" + getStudent().getAge());
    try {
        Thread.sleep(1000);
    }
    catch (InterruptedException ex) {
        ex.printStackTrace();
    }
    System.out.println("thread " + currentThreadName + " second read age is:" + getStudent().getAge());
  }
```

你的student的是一个局部变量 取局部变量的话 永远在方法中都不会改变的

[16楼] [匿名]51CTO游客 回复

2012-05-23 15:27:49

不信的话你将例子改成

```
protected Student getStudent() {

    return new Student();
  }
```

[17楼] zsjin 回复

2013-01-23 10:40:49

很好，写的不够详细！

[18楼] zhong_anjing 回复

2013-06-20 15:39:14

设置一个私有变量

```
private Student student;
......
protected Student getStudent() {

    //Student
    if (useLocal) {
                student = (Student) studentLocal.get();
                //studentLocal.get()null
                if (student == null) {
                      //StudentstudentLocal
                      student = new Student();
                      studentLocal.set(student);
                }
          } else {
                if(student == null)
                      student = new Student();
          }
          return student;
  }
```

[19楼] wenlong1001 回复

2013-09-20 21:46:23

ThreadLocal在1.6版本中不是用一个全局的Map来存各个线程的变量副本，而是在Thread类中有一个ThreadLocalMap的变量，然后用Thread.currentThread().threadLocals.get(this)来引用的各线程变量副本，这样避免了去同步全局的Map

[20楼] javaeye2012 回复

2014-07-14 22:01:40

转载保存到我的博客中了，houzhiqingjava.blog.163.com 如有冒犯请联系，以标明出处和作者。

<< 1 2 >> 页数 (1/2)

发表评论

昵 称: 登录 快速注册

验证码: 请点击后输入验证码 博客过2级，无需填写验证码

内 容:

发表评论