链滴

JVM内存模型来解释多线程并发常见问题和volilate,final,ThreadLocal

作者：wanglei0622

原文链接：https://ld246.com/article/1461750223314

来源网站：链滴

多核CPU运行时，每个CPU都会有自己的Cache，同样jvm运行时，每个线程的私有栈在使用共数据时，都会现将共享数据拷贝进栈顶进行运算，这份数据其实是副本，因此也同样存在多个线程修一个内存单元的一致性问题。
JVM有自己的一套内存模型（Java memory model，JMM）。JDK1.2开始有，JDK1.5(JSR-133才逐渐成熟。JMM中将JVM内存分为“主存”和“工作内存”。 <r />

Main memory,主存，堆内存就是Java对象使用的区域，JMM中定义它为主存。多线程数据一致问题多发生在这里。
working memory ,工作内存,栈空间内部应当包含局部变量，对象的引用，操作数栈，当前方法常量池指针，当前方法的返回地址等信息，这块空间最接近CPU运算，也是每个线程的私有空间，当用方法时将给该私有栈分配空间，方法返回时回收栈空间，
工作内存和主存之间会采用read/write的方式进行通信，，当工作内存中的数据需要计算时，它发生load/store操作，load通常是将本地变量推至栈顶，用来给cpu调度，而store就是将栈顶数据写本地变量。
load操作发生在read之后，普通变量修改未必立即发生Store，但是发生store就会发生write操。

可见性问题，同一个对象，一个线程将该对象的属性修改后，另一个线程看不到该属性被修改的果，或者是未必能马上看到，这种可用volatile修饰属性解决。
指令重排序问题，Java编译器在编译Java代码时，对虚指令重排序，也可以是cpu对目标指令重序，它们的目的是为了高效

<pre class="brush: java">public boolean inited;
private int a;
public void init(){
    int a = 11;
    inited = true;//实际运行时可能被重排，inited的赋值在a之前.多线程判断时就肯能出问题，inite为true但是a还没有赋值。
}

</pre>

4字节赋值问题，jvm中允许对一个非volilate的64位（8字节）变量赋值时，分解成两个32位（4节）完成，并不是必须一次性完成。如果变量是long或double类型的数据，赋值给32位后，正好被一个线程读取，那么读出来的数就是不可预见的结果。
非安全的发布，对象的初始化需要时间，一个线程初始化a对象，另一个线程来使用时，可能a对还没有初始操作，拿到的就是空指针，另一种就是，a对象没有初始化完成，这个对象空间可能已经建， 但是内部属性的初始还要时间，这是由于JMM中并没有要求普通属性的赋值，必须发生在构造法return语句之前。然后程序访问了还没有准备好的内容，一个数据再给多线程使用时，要保证线程全很难，简单的数据可以用final或者用数据拷贝，或者返回不允许修改的代理如集合提供的unmodifiableList(List)。
volilate:誉为最轻量级的锁，volilate变量也会像普通变量那样从主存拷贝到各个线程中去操作，别在于它要求实现storeLoad指令屏障（load前先store）。

第一个作用保证多线程中共享变量始终可见，但引用对象内部的属性不能保证。
JDK1.5后对volilate增强，要求对volilate变量操作时，前后的指令在某种情况下不允许重排序，到指令级别的轻量级锁，还有就是之前的4字节问题，必须一次性赋值

final:JMM中要求final的初始化动作必须在构造方法return前完成。
栈封闭：也就是线程操作的数据都是私有的，不会与其他线程共享。web容器提供给我们的request和response对象就是私有的，因为它不会被其他线程占用，线程绝对安全。想要达到栈封闭的使用据，除了局部变量外，还可以使用ThreadLocal，它本身可以被多个线程共享使用，又可以达到线程

全的目的。
用Thread的 ThreadLocal.ThreadLocalMap全局属性，保存线程-数据的键值对。
 用完后一定要remove,因为如果是线程池，那么线程的生命周期不可预测。

<pre class="brush: java">/**
     * 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) {
        Thread t = Thread.currentThread();
        ThreadLocalMap map = getMap(t);
        if (map != null)
            map.set(this, value);
        else
            createMap(t, value);
    }
    /**
     * 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() {
        Thread t = Thread.currentThread();
        ThreadLocalMap map = getMap(t);
        if (map != null) {
            ThreadLocalMap.Entry e = map.getEntry(this);
            if (e != null)
                return (T)e.value;
        }
        return setInitialValue();
    }
/**
     * Get the map associated with a ThreadLocal. Overridden in
     * InheritableThreadLocal.
     *
     * @param t the current thread
     * @return the map
     */
    ThreadLocalMap getMap(Thread t) {
        return t.threadLocals;
    }</pre>