参考：

[大白话聊聊Java并发面试问题之volatile到底是什么？【石杉的架构笔记】]

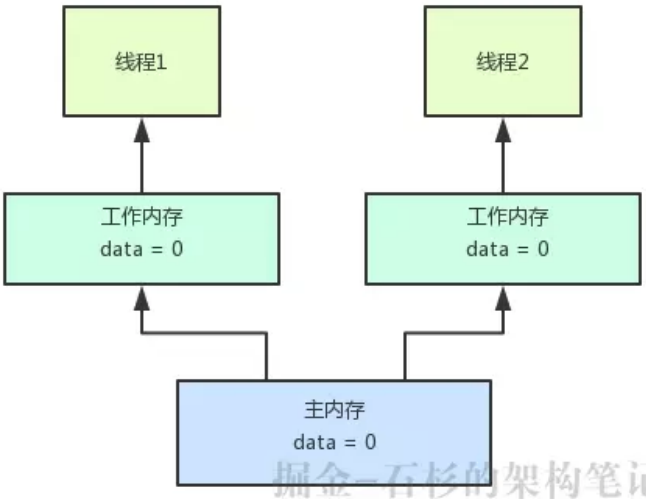
[java volatile关键字解惑（占小狼）]

前言

在之前研究CAS，AQS源码经常会见到核心的变量用这么个关键字修饰下，就是我们这里要讲解的volitile。这个玩意可能很多人都很少关注过，但它实实在在的在并发中发挥着重要的作用，解决了并发中的可见性问题。

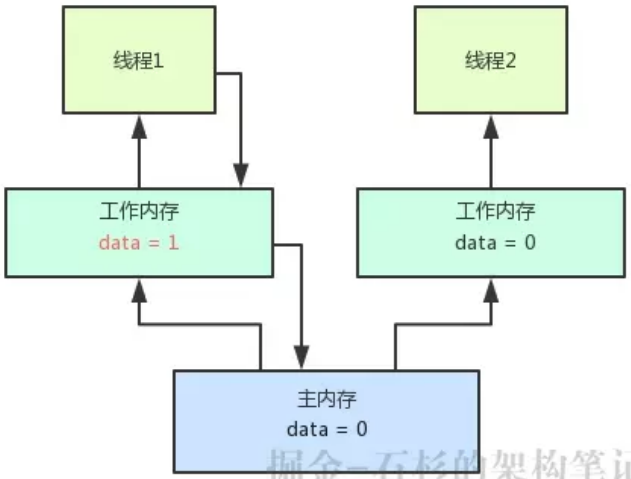
问题引入

我们先上一张图描述一下多线程操作一个数据的场景。



如图，这里涉及到个工作内存的概念，每个线程都有一个工作内存，在处理之前都会从主内存中获取数据写入工作内存。那么，问题就会出现了，比如我线程1把工作内存的值已经修改成1了，然而，线程2

已经在线程1将新值刷回主内存前就从主内存读取了0，那么就是这个状态了。



这样会有什么问题呢，这样问题大了，线程1将1刷回主内存，线程2是不知道的，线程2只会去操作0这样数据就错乱了，这就是可见性问题。

volatile作用简述

那么volatile修饰了变量后会咋样呢，其实原理上就是解决上述问题嘛。第一，一旦某线程修改工作内存后，立刻强制刷回主内存；第二，紧接着强制让其它线程的工作内存的值强制过期，不允许再读取和使用了；第三，其它工作内存再强制从主内存中读取最新值。

volatile使用场景

1.状态标记量

在高并发的场景，通过一个boolean类型的变量控制代码的逻辑走向。例如，我现在想要走true路线了，我们极端假设现在这个地方是有一万个线程在跑，哇，要保持同步的把变量值改成true，你能想到什么

最合适的方式，自然就是volatile了，我只需后台调用一下修改变量为true的方法，因为volatile的功效，一万个线程都将获取变量最新值true，这样我们既不需要重新跑一万个线程，也不需要啥复杂的操作，就让一万个线程的变化同步了。

2.单例模式的double check

该场景具体待应用后再来下结论。

volatile的局限性

其实不算是volatile的局限性，volatile的设计初衷就是解决可见性问题，但有很多人却会误认为这也是一种有效的解决高并发数据安全性的手段，但其实它并不回去管数据的原子性问题。举个例子，高并发的场景下变量a需要进行a++操作，这时候volatile不回去保证啥。再举个例子，变量包含在了具有其它变量的不变式中，不懂上段代码就懂了。

```
public class NumberRange {
    private volatile int lower = 0;
     private volatile int upper = 10;

    public int getLower() { return lower; }
    public int getUpper() { return upper; }

    public void setLower(int value) {
        if (value > upper)
            throw new IllegalArgumentException(...);
        lower = value;
    }

    public void setUpper(int value) {
        if (value < lower)
            throw new IllegalArgumentException(...);
        upper = value;
    }
}
```

这段代码设置了下限和上限，如果两个线程同时操作了setLower和setUpper方法，结果却得到（8，5）这样的错误范围，显然就说明了volatile的乏力。

所以volatile不是去解决高并发各种问题的万金油，这个时候我们就要引出CAS操作这个概念了，这个概念才是保证了数据原子性的问题，就看其它笔记的整理了。