<https://mp.weixin.qq.com/s/0E1X6AsKEgPycAkj1Lbn5w>

# 高并发中计数器的实现方式有哪些？

## 本文主要内容

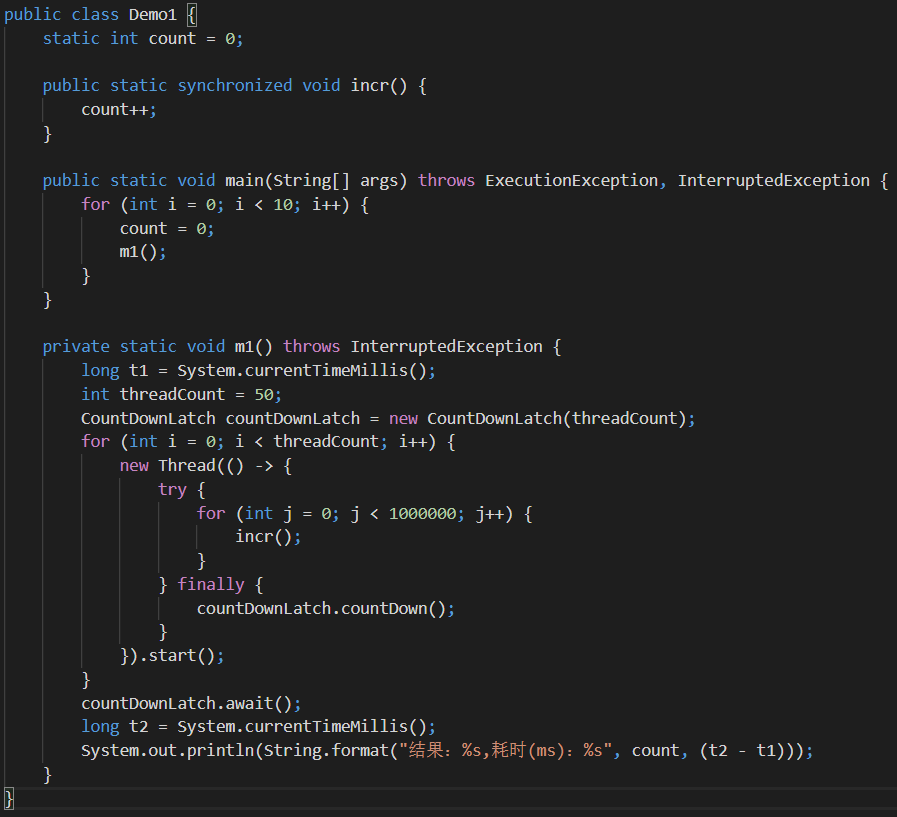
1. 4种方式实现计数器功能，对比其性能
2. 介绍LongAdder
3. 介绍LongAccumulator

**需求：一个jvm中实现一个计数器功能，需保证多线程情况下数据正确性。**

我们来模拟50个线程，每个线程对计数器递增100万次，最终结果应该是5000万。

我们使用4种方式实现，看一下其性能，然后引出为什么需要使用LongAdder、LongAccumulator。

## 方式一：synchronized方式实现



输出：

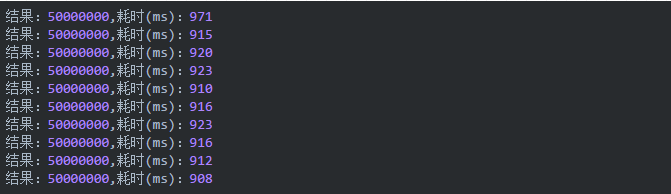


**平均耗时：390毫秒**

## 方式2：AtomicLong实现



输出：



**平均耗时：920毫秒**

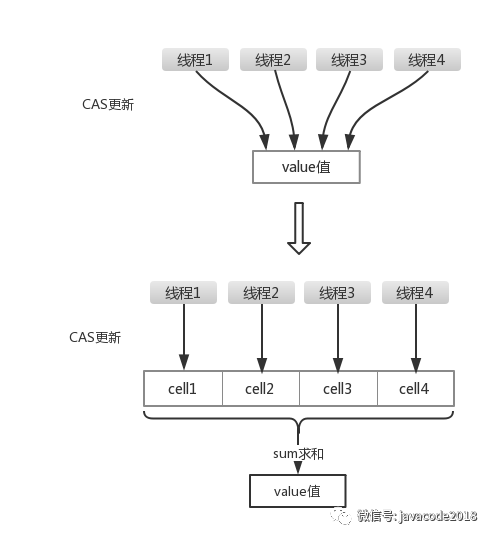
AtomicLong内部采用CAS的方式实现，并发量大的情况下，CAS失败率比较高，导致性能比synchronized还低一些。并发量不是太大的情况下，CAS性能还是可以的。

AtomicLong属于JUC中的原子类，还不是很熟悉的可以看一下：[JUC中原子类，一篇就够了](https://mp.weixin.qq.com/s?__biz=MzA5MTkxMDQ4MQ==&mid=2648933181&idx=1&sn=a1e254365d405cdc2e3b8372ecda65ee&chksm=88621b03bf159215ca696c9f81e228d0544a7598b03fe30436babc95c6a95e848161f61b868c&token=743622661&lang=zh_CN&scene=21#wechat_redirect)

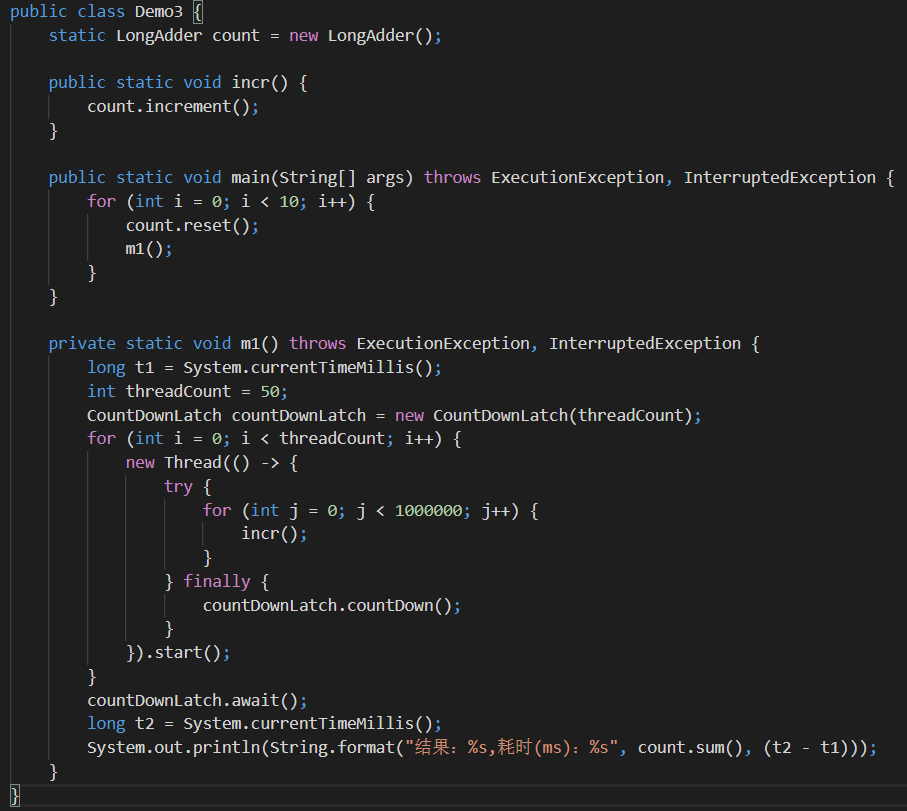
## 方式3：LongAdder实现

先介绍一下LongAdder，说到LongAdder，不得不提的就是AtomicLong，AtomicLong是JDK1.5开始出现的，里面主要使用了一个long类型的value作为成员变量，然后使用循环的CAS操作去操作value的值，并发量比较大的情况下，CAS操作失败的概率较高，内部失败了会重试，导致耗时可能会增加。

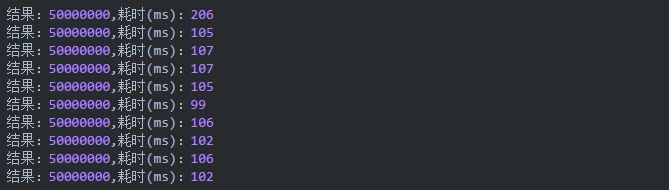
**LongAdder是JDK1.8开始出现的，所提供的API基本上可以替换掉原先的AtomicLong**。LongAdder在并发量比较大的情况下，操作数据的时候，相当于把这个数字分成了很多份数字，然后交给多个人去管控，每个管控者负责保证部分数字在多线程情况下操作的正确性。当多线程访问的时，通过hash算法映射到具体管控者去操作数据，最后再汇总所有的管控者的数据，得到最终结果。相当于降低了并发情况下锁的粒度，所以效率比较高，看一下下面的图，方便理解：



代码：



输出：



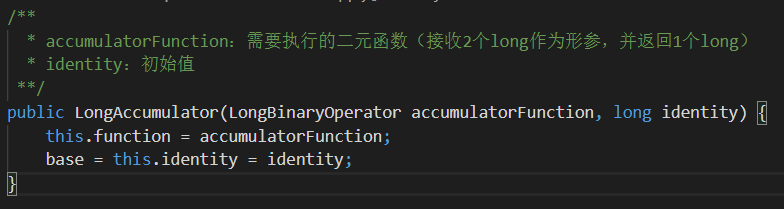
**平均耗时：100毫秒**

代码中new LongAdder()创建一个LongAdder对象，内部数字初始值是0，调用increment()方法可以对LongAdder内部的值原子递增1。reset()方法可以重置LongAdder的值，使其归0。

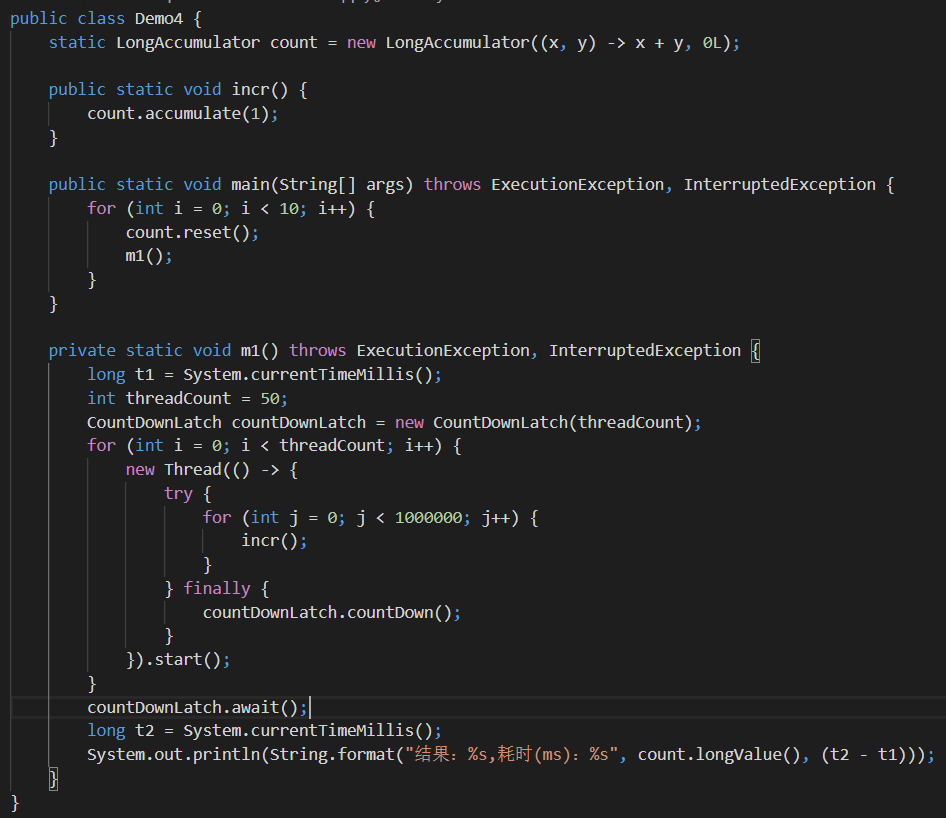
## 方式4：LongAccumulator实现

**LongAccumulator介绍**

LongAccumulator是LongAdder的功能增强版。LongAdder的API只有对数值的加减，而LongAccumulator提供了自定义的函数操作，其构造函数如下：



示例代码：



输出：



**平均耗时：100毫秒**

LongAccumulator的效率和LongAdder差不多，不过更灵活一些。

调用new LongAdder()等价于new LongAccumulator((x, y) -> x + y, 0L)。

从上面4个示例的结果来看，LongAdder、LongAccumulator全面超越同步锁及AtomicLong的方式，建议在使用AtomicLong的地方可以直接替换为LongAdder、LongAccumulator，吞吐量更高一些。