Random

以最简单的NextInt入手，代码如下

```
public int nextInt() {
    return this.next(32);
}
```

可以看到进入的是next方法，代码如下

```
protected int next(int num) {
    AtomicLong seed = this.seed;

    long oldSeed;
    long newSeed;
    do {
        oldSeed = seed.get();
        newSeed = oldSeed * 25214903917L + 11L & 281474976710655L;
    } while(!seed.compareAndSet(oldSeed, newSeed));

     return (int)(newSeed >>> 48 - num);
}
```

上面代码：

1） 获取oldSeed
2）根据 oldSeed计算 newSeed
3）如果 newSeed 不等于 oldSeed，那么将newSeed 的值 更新到 seed
4）返回结果

解释：

1） 用到的是 AtomicLong： 原因是如果用普通的类型的 long ，那么根据
newSeed = oldSeed * 25214903917L + 11L & 281474976710655L
多线程情况下oldSeed的值还没来得及更新， 拿到的 newSeed 都是一样的，所以这里用的是 AtomicLong

2）用到的是AtomicLong，多线程情况下虽然可以解决“多个线程拿到一样的 结果”。 但是需要多个线程竞争，产生大量就会 CAS自旋

ThreadLocalRandom

这个类就是为了提高多线程生成随机数效率的问题，看名字就可以猜到 根 ThreadLocal 类有关。

思想是每个线程持有独自的Random。这样就不会产生多线程抢占而引发的CAS自旋问题

性能对比测试

```
public class RandomTest {
```

```
2
3       public static void main(String[] args) {
4               testRandomSpeed();
5
6               // testThreadLocalRandomSpeed();
7       }
8
9       public static void testThreadLocalRandomSpeed() {
10              AtomicInteger ai = new AtomicInteger();
11
12              for (int i = 0; i < 5; i++) {
13                      new Thread(new Runnable() {
14                              ThreadLocalRandom r = ThreadLocalRandom.current();
15                              @Override
16                              public void run() {
17                                      while (true) {
18                                              r.nextInt();
19                                              ai.addAndGet(1);
20                                      }
21                              }
22                      }).start();
23              }
24
25              try {
26                      Thread.sleep(10000);
27                      System.out.println(ai.get());
28                      System.exit(0);
29              } catch (InterruptedException e) {
30                      e.printStackTrace();
31              }
32      }
33
34      public static void testRandomSpeed() {
35              AtomicInteger ai = new AtomicInteger();
36
37              Random r = new Random();
38              for (int i = 0; i < 5; i++) {
39                      new Thread(new Runnable() {
40
41                              @Override
42                              public void run() {
43                                      while (true) {
44                                              r.nextInt();
45                                              ai.addAndGet(1);
46                                      }
47                              }
48                      }).start();
49              }
50
51              try {
52                      Thread.sleep(10000);
53                      System.out.println(ai.get());
54                      System.exit(0);
55              } catch (InterruptedException e) {
```

```
                    e.printStackTrace();
                }
            }
        }
```

对比结果

本机对比结果是同样开 5个线程 ThreadLocalRandom 是 Random 的 4 ~ 5 倍

另外还有一个结果 只开一个线程的情况下，产生随机数效率 比 5个线程快80倍左右，这是由于单线程没有CAS自旋

换个思路：是不是可以单线程 生成一个随机数池，用的时候从里面取