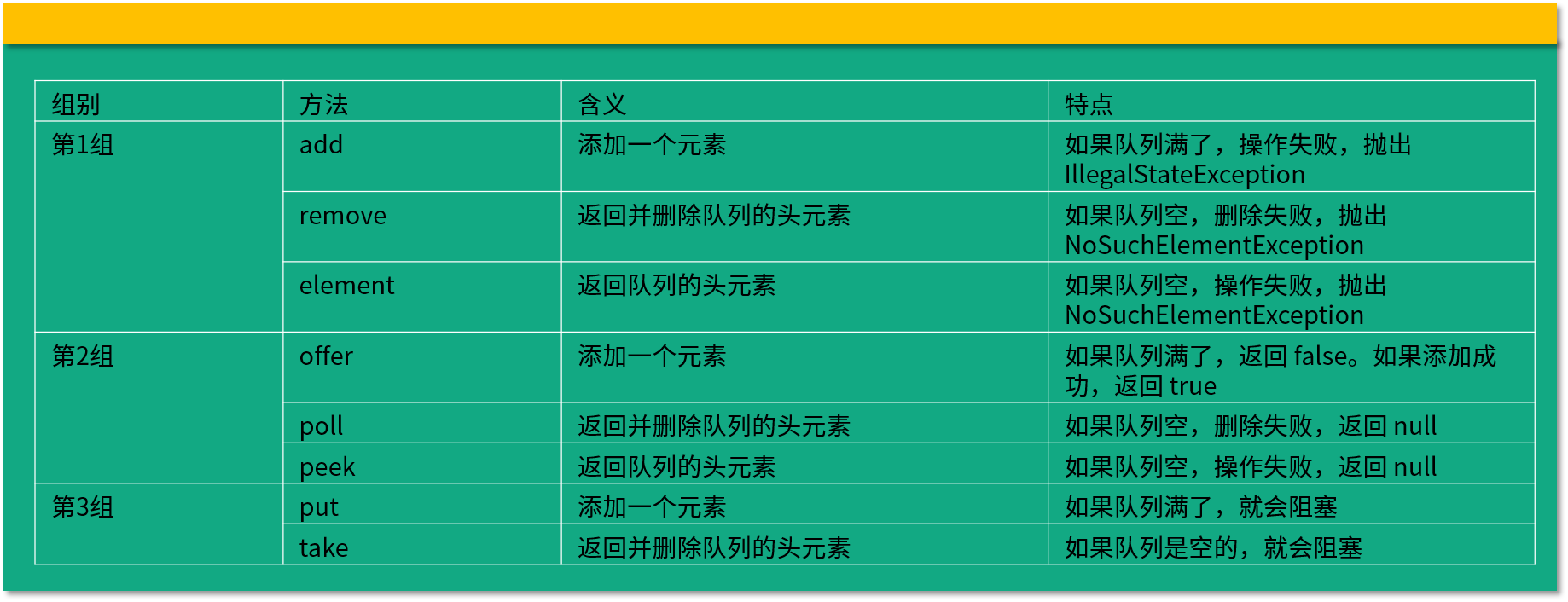
1. **阻塞队列方法**



1. **阻塞队列和非阻塞队列的并发安全原理**

阻塞队列：最主要是利用了 ReentrantLock 以及它的 Condition 来实现

非阻塞队列：利用 CAS 方法实现线程安全。

1. **原子类是如何利用CAS保证线程安全？**

以 AtomicInteger 为例

//JDK 1.8实现

public final int getAndAdd(int delta) {

   return unsafe.getAndAddInt(this, valueOffset, delta);

}

public class AtomicInteger extends Number implements java.io.Serializable {

   // setup to use Unsafe.compareAndSwapInt for updates

   private static final Unsafe unsafe = Unsafe.getUnsafe();

   private static final long valueOffset;

   static {

       try {

           valueOffset = unsafe.objectFieldOffset

               (AtomicInteger.class.getDeclaredField("value"));

       } catch (Exception ex) { throw new Error(ex); }

   }

   private volatile int value;

   public final int get() {return value;}

   ...

}

Unsafe 的 getAndAddInt 方法的实现

public final int getAndAddInt(Object var1, long var2, int var4) {

   int var5;

   do {

       var5 = this.getIntVolatile(var1, var2);

   } while(!this.compareAndSwapInt(var1, var2, var5, var5 + var4));

   return var5;

}

分析compareAndSwapInt 这个方法：

它一共传入了 4 个参数，这 4 个参数是 var1、var2、var5、var5 + var4，为了方便理解，我们给它们取了新了变量名，分别 object、offset、expectedValue、newValue，具体含义如下：

* 第一个参数 object 就是将要操作的对象，传入的是 this，也就是 atomicInteger 这个对象本身；
* 第二个参数是 offset，也就是偏移量，借助它就可以获取到 value 的数值；
* 第三个参数 expectedValue，代表“期望值”，传入的是刚才获取到的 var5；
* 而最后一个参数 newValue 是希望修改的数值 ，等于之前取到的数值 var5 再加上 var4，而 var4 就是我们之前所传入的 delta，delta 就是我们希望原子类所改变的数值，比如可以传入 +1，也可以传入 -1。

总结：

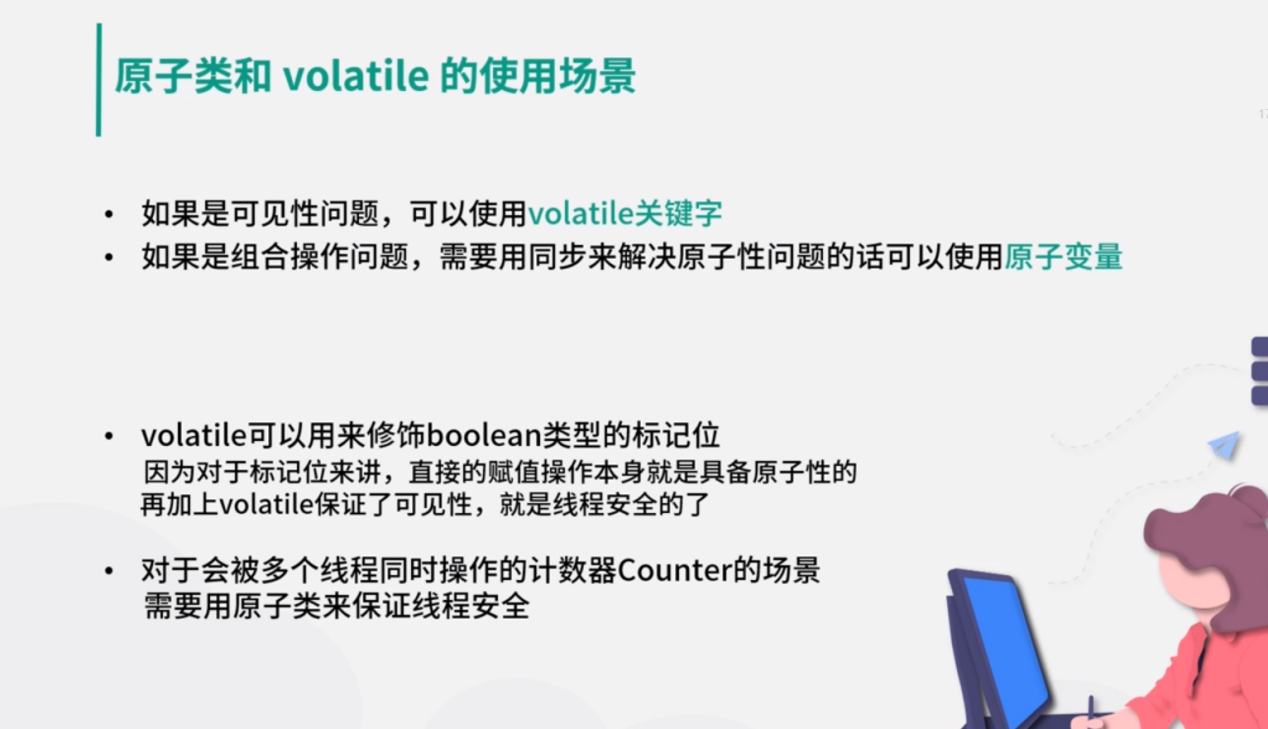
1、利用Unsate类（CAS的核心类）提供硬件级别的原子操作

2、通过Unsate的objectFieldOffset方法获取原子类的value的偏移量，即内存中的偏移地址

3、通过do-while循环（即自旋原理），通过compareAndSwapInt方法尝试更新value值，如果更新失败就重新获取，然后再次

尝试更新，直到更新成功

1. **原子类和 volatile**



1. **AtomicInteger 和 synchronized 的异同点**

