为什么需要AtomicInteger?

对于全局变量的数值类型操作num++，如果没有synchronized关键字则是线程不安全的，num=num+1，明显，这个操作不具备原子性，多线程时必然会出现问题。

换成volatile修饰count变量?

volatile修饰的变量能够在线程间保持可见性，能被多个线程同时读，并且不会读取到过期值。但是不能保证非原子性操作。

AtomicInteger?

以getAndIncrement为例看下源码

```
public final int getAndIncrement() {
    for (;;) {
        //先取出AtomicInteger的当前值
        int current = get();
        //对当前值加1操作
        int next = current + 1;
        //这里很关键，通过compareAndSet方法比较当前值有没有被其它线程修改过，若修改过返回fa
        if (compareAndSet(current, next))
            return current;
    }
}
```

compareAndSet方法里面是调用了Unsafe类的compareAndSwapInt方法

```
public final native boolean compareAndSwapObject(Object var1, long var2, Object var4,

public final native boolean compareAndSwapInt(Object var1, long var2, int var4, int v

public final native boolean compareAndSwapLong(Object var1, long var2, long var4, lon
```

Unsafe是Java HotSpot提供的操作内存和线程的"后门"，官方或者对于生产环境并不建议使用Unsafe类，因为它的API不稳定、不安全，错误使用将给你的HotSpot jvm带来致命性的灾难。同时对于其他基本类型，比如char、float、double等并没有对应的上述判断是否被修改方法，故可以将其转为compareAndSwapInt来简介判断，因为在AtomicBoolean的源码中就是这么做的。