**CopyOnWriteArrayList**

都清楚ArrayList并不是线程安全的，在读线程在读取ArrayList的时候如果有写线程在写数据的时候，基于fast-fail机制，会抛出ConcurrentModificationException异常也就是说ArrayList并不是一个线程安全的容器，当然您可以用Vector,或者使用Collections的静态方法将ArrayList包装成一个线程安全的类，但是这些方式都是采用java关键字synchronzied对方法进行修饰，利用独占式锁来保证线程安全的。但是，由于独占式锁在同一时刻只有一个线程能够获取到对象监视器，很显然这种方式效率并不是太高。

回到业务场景中，有很多业务往往是读多写少的，比如系统配置的信息，除了在初始进行系统配置的时候需要写入数据，其他大部分时刻其他模块之后对系统信息只需要进行读取，又比如白名单，黑名单等配置，只需要读取名单配置然后检测当前用户是否在该配置范围以内。类似的还有很多业务场景，它们都是属于读多写少的场景

Doug Lea大师就为我们提供CopyOnWriteArrayList容器可以保证线程安全，保证读读之间在任何时候都不会被阻塞

在**读线程的角度来看，即读线程任何时候都是获取到最新的数据，满足数据实时性**。既然我们说到要进行优化，必然有trade-off,我们就可以**牺牲数据实时性满足数据的最终一致性即可**。而CopyOnWriteArrayList就是通过Copy-On-Write(COW)，即写时复制的思想来通过延时更新的策略来实现数据的最终一致性，并且能够保证读线程间不阻塞。

COW通俗的理解是当**我们往一个容器添加元素的时候，不直接往当前容器添加，而是先将当前容器进行Copy，复制出一个新的容器，然后新的容器里添加元素，添加完元素之后，再将原容器的引用指向新的容器。对CopyOnWrite容器进行并发的读的时候，不需要加锁，因为当前容器不会添加任何元素。**所以CopyOnWrite容器也是一种读写分离的思想，延时更新的策略是通过在写的时候针对的是不同的数据容器来实现的，放弃数据实时性达到数据的最终一致性。

**3. CopyOnWriteArrayList的实现原理**

实际上CopyOnWriteArrayList内部维护的就是一个数组数组引用是被volatile修饰，注意这里仅仅是修饰的是数组引用，其中另有玄机，稍后揭晓。关于volatile很重要的一条性质是它能够够保证可见性说，我们自然而然最关心的就是读写的时候，**分别为get和add方法的实现**。

可以看出来**get方法实现非常简单，几乎就是一个“单线程”程序，没有对多线程添加任何的线程安全控制，也**没有加锁也没有CAS操作等等，原因是，所有的读线程只是会读取数据容器中的数据，并不会进行修改。

**add方法的逻辑也比较容易理解，请看上面的注释。需要注意这么几点：**

1，采用ReentrantLock，保证同一时刻只有一个写线程正在进行数组的复制，否则的话内存中会有多份被复制的数据；

2，前面说过数组引用是volatile修饰的，因此将旧的数组引用指向新的数组，根据volatile的happens-before规则，写线程对数组引用的修改对读线程是可见的。

3，由于在写数据的时候，是在新的数组中插入数据的，从而保证读写实在两个不同的数据容器中进行操作。