链滴

Copy-on-Write 模式

作者：vcjmhg
原文链接：https://ld246.com/article/1627465413844
来源网站：链滴

概述

Copy-on-Write模式，与其说是一种技术，还不如说是一种思想即**写时复制**，它在诸多方面都有应用比如当我们使用fork()方法在linux中创建子进程时，并不复制整个进程的地址空间，而是让子进程和进程共享同一个内存空间；只有父进程或者子进程需要写入时，才复制地址空间，让父进程和子进程有独立的内存空间。这就是一种典型的Copy-on-Wirte模式，也是一种延迟写思想的体现。

这种思想在Java的容器的实现上也有着广泛的应用。

在Java中的应用

CopyOnWriteArrayList 和 CopyOnWriteArraySet 这两个 Copy-on-Write 容器，它们背后的设计想就是 Copy-on-Write；通过 Copy-on-Write 这两个容器实现的读操作是无锁的，由于无锁，所以读操作的性能发挥到了极致。

我们以CopyOnWriteArrayList为例，讲讲其内部是如何实现的。

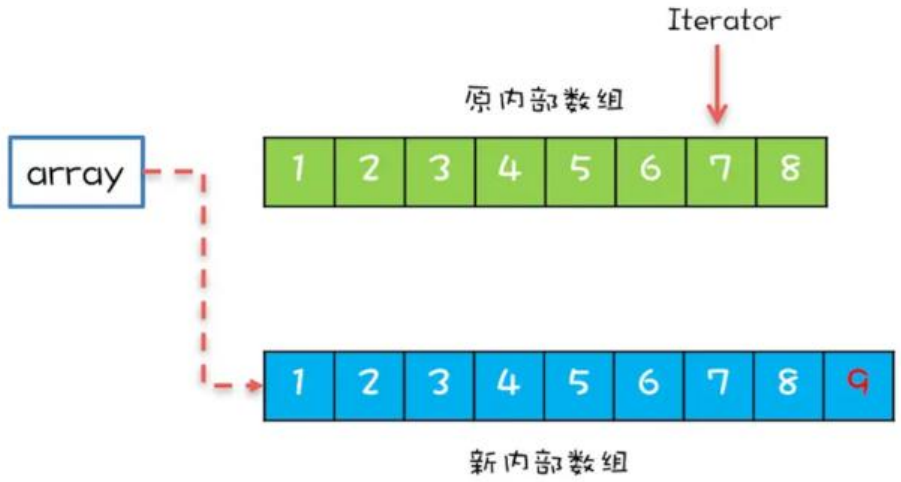
首先我们可以看其set()和get()方法的实现：

```
public E get(int index) {
    //直接返回对应位置的值
    return elementAt(getArray(), index);
}

public E set(int index, E element) {
    //加同步锁保证线程安全
    synchronized (lock) {
        Object[] es = getArray();
        E oldValue = elementAt(es, index);
        //设置的值不和原来的值相等，则进行复制操作
        if (oldValue != element) {
            es = es.clone();
            es[index] = element;
        }
        // Ensure volatile write semantics even when oldvalue == element
        setArray(es);
        return oldValue;
    }
}
```

get()方法内部比较简单就是直接返回对应位置值，核心实际上是在set()，首先在执行set()方法设置值时，先和原来的旧值进行判断，如果不同则进行copy新的空间。

整个过程用图表示如下：



CopyOnWriteArrayList内部维护一个Array数组当进行**“读操作”**直接迭代即可，当发生写操作的候，会复制一块新的数据空间，复制完成之后再将数组指针指向新的数组空间。如果在这个过程中发读操作，则读取的仍然原来的旧数组（快照）**。

通过这种方式可以大大提高**“读的效率”**。

但如果要使用CopyOnWriteArrayList有两个注意点：

1. 要注意应用场景：CopyOnWriteArrayList 仅适用于 **写操作非常少的场景，而且能够容忍读写的暂不一致。**

2. CopyOnWriteArrayList **迭代器是只读的**，不支持增删改。因为迭代器遍历的仅仅是一个快照，对快照进行增删改是没有意义的。

CopyOnWriteArraySet 内部是借助与CopyOnWriteArrayList 实现的，只是在add的时候调用的是CopyOnWriteArrayList 的addIfAbsent() 来实现去重，此处不再详述。

总结

Copy-On-Write本质上讲是一种**延迟复制**的思想，在执行读操作的时候，不进行复制，而只有真正行写操作的时候才进行复制。它的应用也非常广泛，比如linux中的fork() 方法，java中的CopyOnWriteArrayList。在使用时要注意其应用场景，因为其每次发生写操作都会复制新的空间，因此它天然适合**多，写少的场景，并且能够忍受一定时间的读写不一致。**

参考

1. 《 Copy-on-Write模式：不是延时策略的COW》
2. 《并发容器：都有哪些“坑”需要我们填？》