我们组的Java分析的是ZooKeeper的观察者模式

首先简单介绍一下ZooKeeper：

ZooKeeper是Apache Hadoop下的一个子项目，是一个树形目录服务。它是一个分布式的、开源的分布式应用程序的协调服务。

Zookeeper提供的主要功能包括配置管理、分布式锁、集群管理等

先分析一下主要涉及到的类和接口：（类图）

在DataTree类含有两个成员变量：dataWatches和childWatches，用于监视数据与子节点。这两个变量都是IWatchManager类型的接口，在WatchManager这个类中实现。在这个实现类中有字典watchTable，用于实现watch名字到对象实例集合的映射。这个HashMap的key是String类型，代表节点的path，将path映射到各个节点所含有的监听器集合。

以监听器是如何注册的为例：在DataTree中含有addWatch函数，用于将监听器注册到数据树上。首先获取到监听器的模式，然后根据需求，分别注册到之前提到的两个监听器上。这个函数中调用WatchManager的addWatch，将监听器注册到管理者中进行管理。

监听器的删除用到DataTree中的removeCnxn，这个函数接收一个Watcher对象，调用WatchManager的removeWatcher方法对某个监听器进行注销。removeWatcher方法注销监听器的同时还会对一些垃圾进行回收，保证树的干净整洁，避免内存空间损伤或无效存储。

DataTree中的其他方法：

deleteNode删除节点、createNode增加节点、setData更新数据，会在调整完相应状态后触发WatchManager的triggerWatch方法，触发监听器传入各种状态

下面具体对监听器触发，即WatchManager的triggerWatch方法进行说明：

该函数接收多个参数，重点参数为path：节点的路径，type：触发的事件的类型等，这些变量用于找到需要触发的触发器集合，同时说明触发的事件类型和内容。

1. 函数中首先创建一个WatchedEvent对象，然后创建一个HashSet，使用synchronized加锁，确保线程安全；
2. 然后路径获取一个迭代器，用于遍历父路径，在接下来的循环中遍历父路径，使用watchTable的get方法来根据路径获取对应的监听器集合；
3. 检查他们的监听模式并添加到watchers集合中，对于不同的监听模式可能会修改监听器状态，如果当前路径下监听器集合为空，则需要从watchTable中移除；
4. 通过遍历获取到了所有需要触发的监听器集合后，只要遍历watches集合，调用process方法，根据事件类型触发相应的Watcher并更新；
5. 最后返回一个WatcherOrBitSet对象，其中包含触发的Watcher集合

这样就完成了对所有需要通知到的监听器的通知。

下面结合lock下的测试类WriteLockTest，来讲下ZooKeeper的论文中写到的是如何利用Watch实现锁机制的：

测试类WriteLockTest继承自ClientBase类，这样测试类就可以作为ZooKeeper服务的客户端了。

测试方法runTest中新建了三个写锁，分别位于对应节点；

1. 遍历三个节点，创建一个客户端用于建立和ZooKeeper服务的会话，并将第一个节点选举为领导者；
2. 然后设置锁的监听器，注册两个回调函数lockReleased与lockAcquired，用于上锁与释放锁；
3. 接着节点开始调用lock方法，在确认路径存在后，就需要用retryOperation尝试10次lock操作；这段代码中的zop是实现类LockZooKeeperOperation的一个实例。retryOperation会调用execute方法。

羊群效应比喻人的从众心理，在ZooKeeper分布式锁场景中表现为所有客户端都尝试对同一个临时节点加锁。当一个锁被占有时，所有的客户端都会监听这个临时节点。一旦其被释放，大量的客户端都会尝试去对同一个临时节点创建锁，大量请求带来了很大的网络负载

在ZooKeeper中避免羊群机制的伪代码如下，这段伪代码的含义是：

1. 每一个想要获取锁的客户端都需要在I下创建一个带有“/lock-”前缀的节点n，这些节点按申请顺序排序给予一个由小到大的后缀。EPHEMERAL表示创建的节点是临时节点，当创建节点的会话结束时，该节点将被自动删除。SEQUENTIAL表示在节点名后面添加一个单调递增的序号，确保并发情况下节点名称的唯一性；

2. 创建节点后，获取节点I下所有的子节点，放到列表C中，此时不设置监听；

3. 如果刚创建的n节点是列表C中后缀最小的一个节点，则表示该节点应该获得锁，直接退出；

4. 否则，获取后缀比n小的节点列表，并找到该列表中后缀最大的那一个节点p，也就是刚刚比n节点小的那个节点；

5. 用exists函数判断节点p是否存在，如果存在则对这个节点设置监听，以便在节点p发生变化（即前一节点锁释放节点被删除）时能够触发LockWatcher调用lock方法，后一个节点即可获得该互斥锁。这里用到了观察者模式，即在被观察者（前一节点）中注册观察者列表，当被观察者状态发生变化时（即节点被删除），发送通知给观察者（这里为后一个节点）；

6. 如果本轮节点n没有获取到锁，则转到步骤2继续尝试获取锁。

前面讲完了上锁，接下来讲一下解锁：

解锁时需要判断会话保持开启并且对应锁存在，然后创建一个匿名类实现ZooKeeperOperation接口并delete，删除完成后需要释放原先的互斥锁，以便紧跟在后面的节点上锁。

通过上述方法避免羊群效应，就可以实现领导人选举了。注意每次释放锁需要30s的时间来让之前的Leader挂掉，因此释放后需要等待30s。前一个领导人牺牲后实施顺位继承，而且不用引发竞争。