四家工具 公安部和 关数 都比较简单 重点在百分点和大麦

先说说大麦网（非主流头型面试官小哥）

1 .Flume 处理日志重复问题

将agent节点上监听的日志文件夹下已经发送的日志文件移出，处理完，故障重新启动agent即可。 注：在agent节点失败的情况下，按照失败的时间点，将时间点之前的数据文件移出，将flume.agent.logdir配置的文件夹清空，重新启动agent。

2.你的推荐系统矩阵列表是怎么实现的

通过数组来实现 数组的行和列代表 333:777 6 一看就能明白吧

3.说说你日志处理具体怎么写的mapreduce 流程

4.hbase 优化问题

5.hadoop 数据倾斜问题

6.flume 监控问题

可以写一个control.sh 脚本来控制flume的启动、关闭、重启。  
    启动demo：  
./bin/flume-ng agent --conf ./conf/ --conf-file ./conf/flume.conf -n agent1 > ./start.log 2>&1 &

    从此以后，日志数据就从日志文件，通过xxx.py读取，进入到flie-channel，再被flume-ng-mongodb-sink读走，进入到目的地MongoDB Cluster。  
搭好基本功能之后，以后需要做的就是调整xxx.py、增强flume-ng-mongodb-sink。

7.HDFS 存储的什么类型

文本类型

8.java 中 怎么解决多线程问题 有几种方式 ，他们之间的不同

9

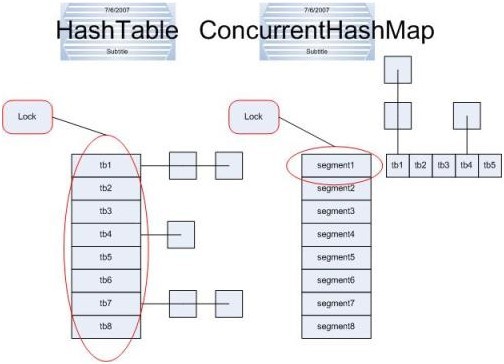
百分点公司（一个大胖带胡子 ，和一个小瘦一起面）

通过mapreduce 实现两个表的join

Hashtale hashmap currenthashtable 区别

解答 ：ConcurrentHashMap融合了hashtable和hashmap二者的优势。

Hashtable 同步 ，hashtable在的多线程情况下，同步操作能保证程序执行的正确性。

HashMap 不是同步的 ，所以hashmap在单线程情况下效率较高   


图左侧清晰的标注出来，lock每次都要锁住整个结构。

ConcurrentHashMap正是为了解决这个问题而诞生的。

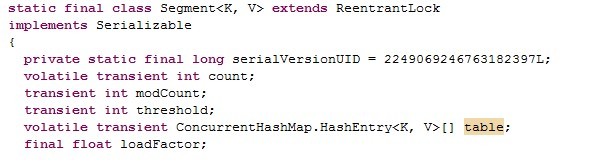
ConcurrentHashMap锁的方式是稍微细粒度的。 ConcurrentHashMap将hash表分为16个桶（默认值），诸如get,put,remove等常用操作只锁当前需要用到的桶。

试想，原来 只能一个线程进入，现在却能同时16个写线程进入（写线程才需要锁定，而读线程几乎不受限制，之后会提到），并发性的提升是显而易见的。

更令人惊讶的是ConcurrentHashMap的读取并发，因为在读取的大多数时候都没有用到锁定，所以读取操作几乎是完全的并发操作，而写操作锁定的粒度又非常细，比起之前又更加快速（这一点在桶更多时表现得更明显些）。只有在求size等操作时才需要锁定整个表。

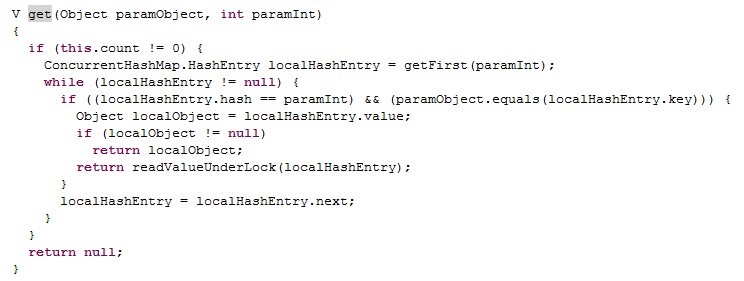
而在迭代时，ConcurrentHashMap使用了不同于传统集合的快速失败迭代器的另一种迭代方式，我们称为弱一致迭代器。在这种迭代方式中，当iterator被创建后集合再发生改变就不再是抛出 ConcurrentModificationException，取而代之的是在改变时new新的数据从而不影响原有的数 据，iterator完成后再将头指针替换为新的数据，这样iterator线程可以使用原来老的数据，而写线程也可以并发的完成改变，更重要的，这保证了多个线程并发执行的连续性和扩展性，是性能提升的关键。

下面分析ConcurrentHashMap的源码。主要是分析其中的Segment。因为操作基本上都是在Segment上的。先看Segment内部数据的定义。

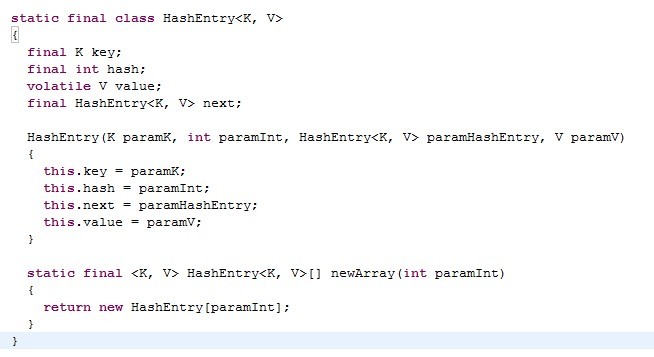


从上图可以看出，很重要的一个是table变量。是一个HashEntry的数组。Segment就是把数据存放在这个数组中的。除了这个量，还有诸如loadfactor、modcount等变量。

看segment的get 函数的实现：



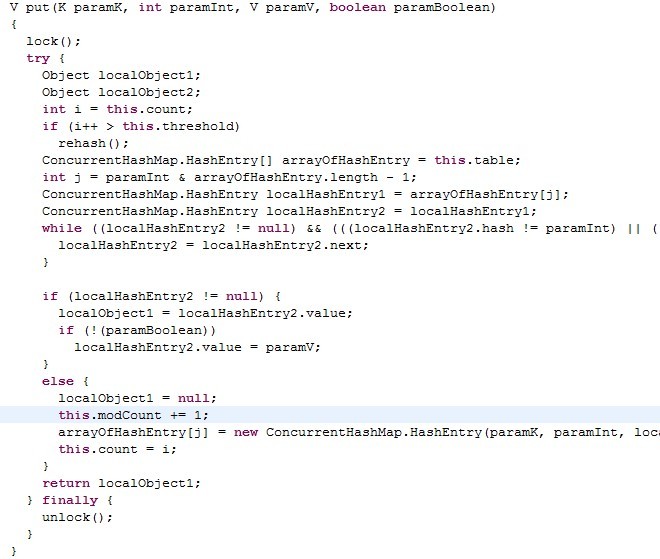
加上hashentry的代码：



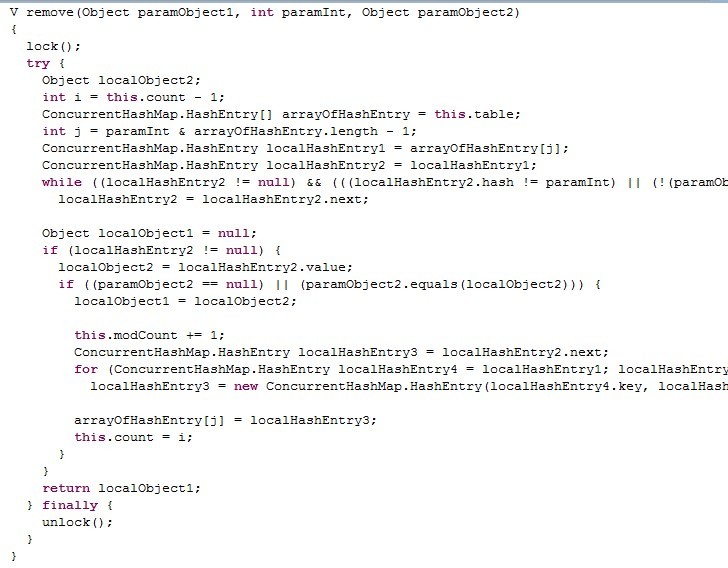
可以看出，hashentry是一个链表型的数据结构。

在segment的get函数中，通过getFirst函数得到第一个值，然后就是通过这个值的next，一路找到想要的那个对象。如果不空，则返回。如果为空，则可能是其他线程正在修改节点。比如上面说的弱一致迭代器在将指针更改为新值的过程。而之前的 get操作都未进行锁定，根据bernstein条件，读后写或写后读都会引起数据的不一致，所以这里要对这个e重新上锁再读一遍，以保证得到的是正确值。readValueUnderLock中就是用了lock()进行加锁。

put操作已开始就锁住了整个segment。这是因为修改操作时不能并发的。



同样，remove操作也是如此(类似put，一开始就锁住真个segment)。



但要注意一点区别，中间那个for循环是做什么用的呢？（截图未完全，可以自己找找代码查看一下）。从代码来看，就是将定位之后的所有entry克隆并拼回前面去，但有必要吗？每次删除一个元素就要将那之前的元素克隆一遍？这点其实是由entry的不变性来决定的，仔细观察entry定义，发现除了value，其他 所有属性都是用final来修饰的，这意味着在第一次设置了next域之后便不能再改变它，取而代之的是将它之前的节点全都克隆一次。至于entry为什么要设置为不变性，这跟不变性的访问不需要同步从而节省时间有关。

Jvm 机制 以及OOM发生的情况 简述一下

解答： 了解下jvm 运行机制 ，OMM 还有一些优化既可

用python 写一个脚本程序 向mysql 批量添加用户

Mysql 主从复制一段时间后 突然数据同步不了怎么办

show processlist; 查看下进程是否Sleep太多。发现很正常。

show master status; 也正常。 show slave status\G

再到Slave上查看

Slave\_IO\_Running: Yes   
Slave\_SQL\_Running: No

有一个No 也不行检查出来问题

解决：   
stop slave;   
#表示跳过一步错误，后面的数字可变   
set global sql\_slave\_skip\_counter =1;   
start slave;   
之后再用mysql> show slave status\G 查看：   
Slave\_IO\_Running: Yes   
Slave\_SQL\_Running: Yes   
ok，现在主从同步状态正常了。。。

方式二：重新做主从，完全同步

Hadoop1.x 和2.x 的差别

说说yarn工作流程

Java 数组反转实现

Java 序列化 SqenceFile 并说明你的序列化方式和适用场景

说说你hadoop 中都每个配置文件都配置了什么吧  
 说说你的都用了naginx 中都用了那些模块在什么条件下使用的