图文 094、案例实战：一个数据同步系统频繁OOM内存溢出的

详情 评论

案例实战：
一个数据同步系统频繁OOM内存溢出的排查实践

石杉老哥重磅力作：《互联网java工程师面试突击》（第3季）**【强烈推荐】**：

全程真题驱动，精研Java面试中**6大专题的高频考点**，从面试官的角度剖析面试

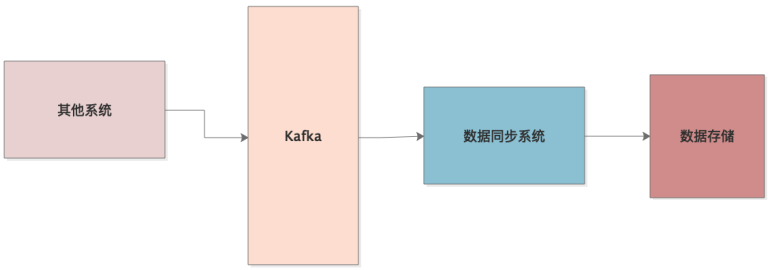
（点击下方蓝字试听）

《互联网Java工程师面试突击》（第3季）

1、案例背景

首先说一下案例背景，线上有一个数据同步系统，是专门负责从另外一个系统去同步数据的，简单来说，另外一个系统会不停的发布自己的数据到Kafka中去，然后我们有一个数据同步系统就专门从Kafka里消费数据，接着保存到自己的数据库中去，大概就是这样的一个流程。

我们看下图，就是这个系统运行的一个流程。



结果就这么一个非常简单的系统，居然时不时就报一个内存溢出的错误，然后就得重启系统，过了一段时间又会再次内存溢出一下。

而且这个系统处理的数据量是越来越大，因此我们发现他内存溢出的频率越来越高，到这个情况，就必须要处理一下了。

2、经验丰富的工程师：从现象看到本质

一般遇到这种现象，只要是经验丰富的工程师，应该已经可以具备从现象看到本质的能力了。我们可以来分析和思考一下，既然每次重启过后都会在一段时间以后出现内存溢出的问题，说明肯定是每次重启过后，内存都会不断的上涨。

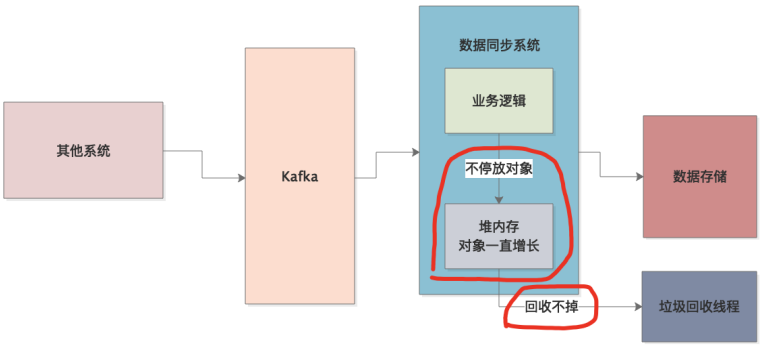
而且一般要高到JVM出现内存溢出，通常就是两种情况，要不然是并发太高，瞬间大量并发创建过多的对象，导致系统直接崩溃了。要不就是有内存泄漏之类的问题，就是很多对象都赖在内存里，无论你如何GC就是回收不掉。

那么这个场景是怎么回事呢？我们当时分析了一下，这个系统的负载并不是很高，虽然数据量不少，但并不是那种瞬时高并发的场景。

这么看来，很可能就是随着时间推移，有某种对象越来越多，赖在内存里了。然后不断的触发gc，结果每次gc都回收不掉这些对象。

一直到最后，内存实在不足了，就会内存溢出

我们看看下面的图，在下图里就画出了这个问题。



3、通过jstat来确认我们的推断

接着直接在一次重启系统之后，用jstat观察了一下JVM运行的情况：

我们发现，老年代的对象一直在增长，不停的在增长。每次Young GC过后，老年代的对象就会增长不少。

而且当老年代的使用率达到100%之后，我们发现会正常触发Full GC，但是Full GC根本回收不掉任何对象，导致老年代使用率还是100%！

然后老年代使用率维持100%一段时间过后，就会报内存溢出的问题，因为再有新的对象进入老年代，实在没有空间放他了！

所以这就基本确认了我们的判断，每次系统启动，不知道什么对象会一直进入堆内存，而且随着Young GC执行，对象会一直进入老年代，最后触发Full GC都无法回收老年代的对象，最终就是内存溢出。

4、通过MAT找到占用内存最大的对象！

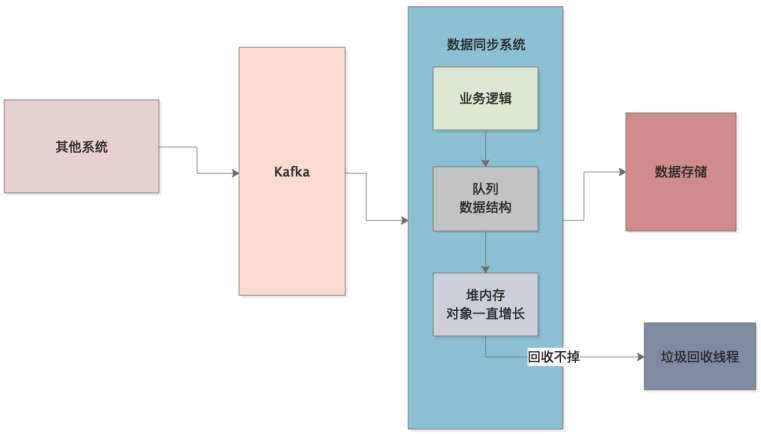
关于MAT分析内存快照的方法，之前已经讲解的很详细了，在这个案例中就不用重复截图了，直接说出过程和结论就好！

在内存快照中，我们发现了一个问题，那就是有一个队列数据结构，直接引用了大量的数据，就是这个队列数据结构占满了内存！

那这个队列是干什么用的？

简单来说，从Kafka消费出来的数据会先写入这个队列，接着从这个队列再慢慢写入数据库中，主要是要额外做一些中间的数据处理和转换，所以自己在中间又加了一个队列。

我们看下面的图。



那么这个队列是怎么用的？问题就出在这里了！

大家都知道，从Kafka消费数据，是可以一下子消费一批出来的，比如消费几百条数据出来。

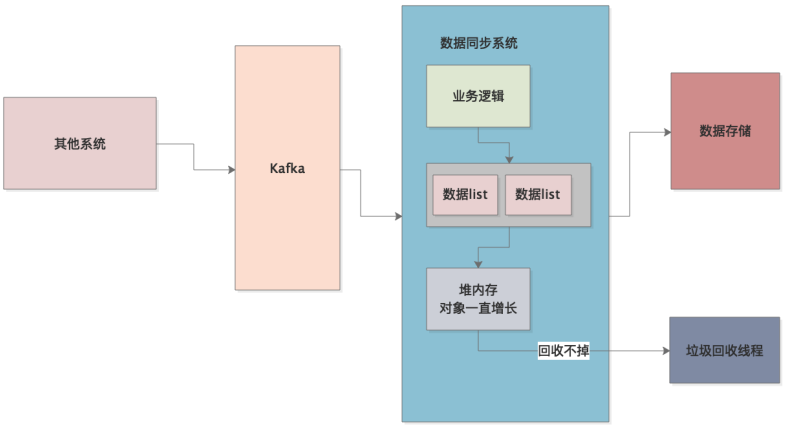
因此当时这个写代码的工程师，直接就是每次消费几百条数据出来给做成一个List，然后把这个List放入到队列里去！

最后就搞成了这种情况：比如一个队列有1000个元素，每个元素都是一个List，每个List里都有几百条数据！

这种做法怎么行？会导致内存中的队列里积压几十万条，甚至百万条数据！最终一定会导致内存溢出！

而且只要你数据还停留在队列中，就是没有办法被回收的。

我们看下面的图。



上图就是一个典型的对生产和消费的速率没控制好的例子。

从Kafka里消费出来数据放入队列的速度很快，但是从队列里消费数据进行处理，然后写入存储的速度较慢，最终会导致内存队列快速积压数据，导致内存溢出。

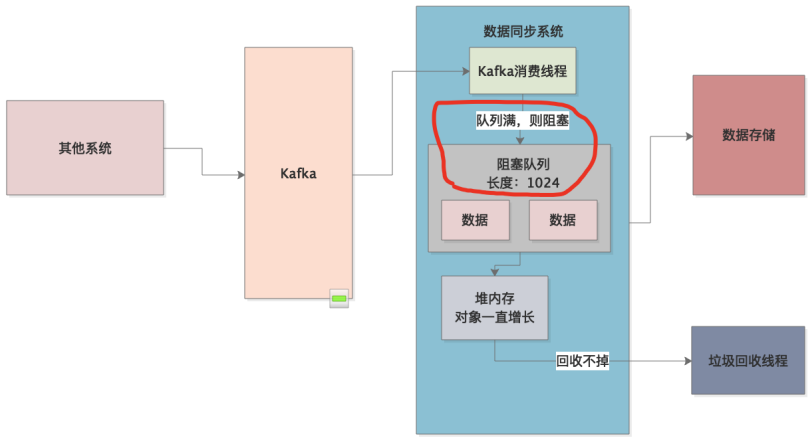
而且这种队列每个元素都是一个List的做法，会导致内存队列能容纳的数据量大幅度膨胀。

最终解决这个问题也很简单，把上述内存队列的使用修改了一下，做成了定长的阻塞队列。

比如最多1024个元素，然后每次从Kafka消费出来数据，一条一条数据写入队列，而不是做成一个List放入队列作为一个元素。

因此这样内存中最多就是1024个数据，一旦内存队列满了，此时Kafka消费线程就会停止工作，因为被队列给阻塞住了。不会让内存队列中的数据过多。

我们看下面解决问题之后的图：



5、本文小结

本文是我们整个专栏的最后一个案例，相信大家认真学完这个专栏后，就会感受到我们设计这个专栏的思路。

专栏的核心是通过一步一图和大白话的方式，让大家学会JVM的核心运行原理，接着学习了JVM GC优化的核心原理和OOM问题的核心原理。

接着我们给大家讲解了JVM的GC问题以及OOM的常见发生场景和解决方法。

同时我们带给了大家数十个来源于我们真实生产环境的JVM优化案例，包括GC优化案例和OOM优化案例

大量的优化案例让大家可以对各种不同场景的问题有一个了解，同时积累起来了对不同问题进行分析、排查和解决的思路。

在这个过程中，如果大家反复去把这些案例看几遍，吸收透彻了，本质上就会积累起来较为丰富的JVM优化实践的经验积累

当你日后真的在工作中需要解决JVM问题的时候，就会发现这些知识全部都可以派上用场了。

End

如果断更联系QQ/微信642600657