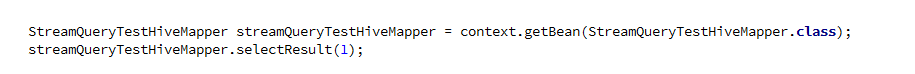
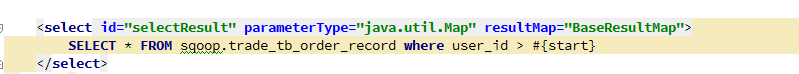
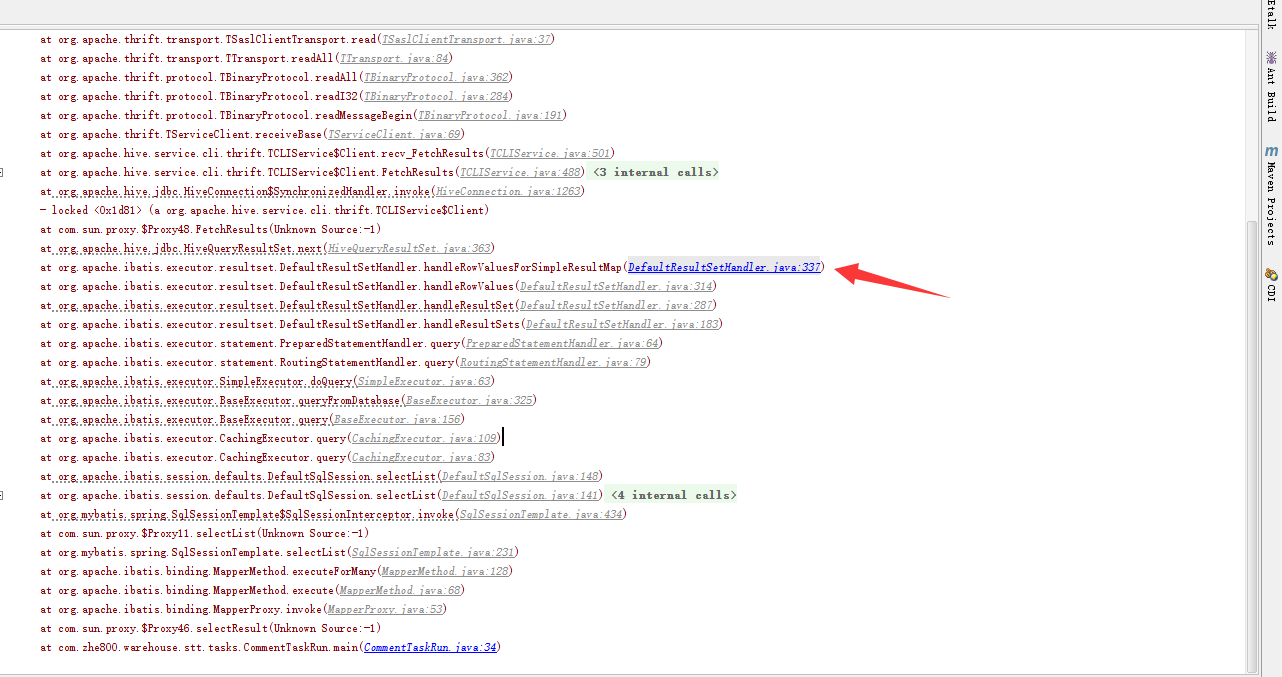
Mybatis+hive性能调优: stream query,I/O optimize

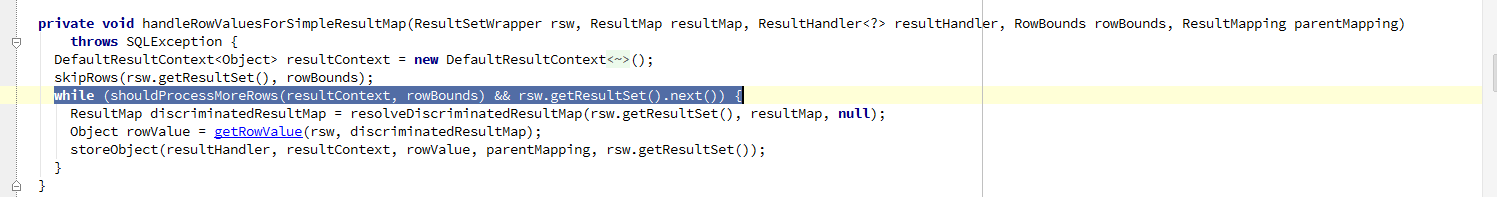
最近在调试线上线下oom的时候,偶然发现目前我们使用mybatis的方式有待改进,表现出来的现象就是,每一次mybatis查询,代表一行具体数据的pojo实例在内存中数目不断累积,但是程序却一直阻塞着不能先对已完成的部分进行处理,这里我用线下环境数据量比较大的一个表tb\_order\_record做查询实验,代码执行到selectResult之后就阻塞住了,



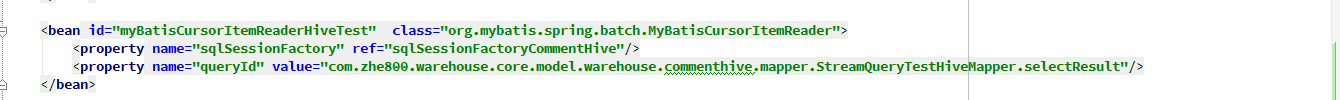


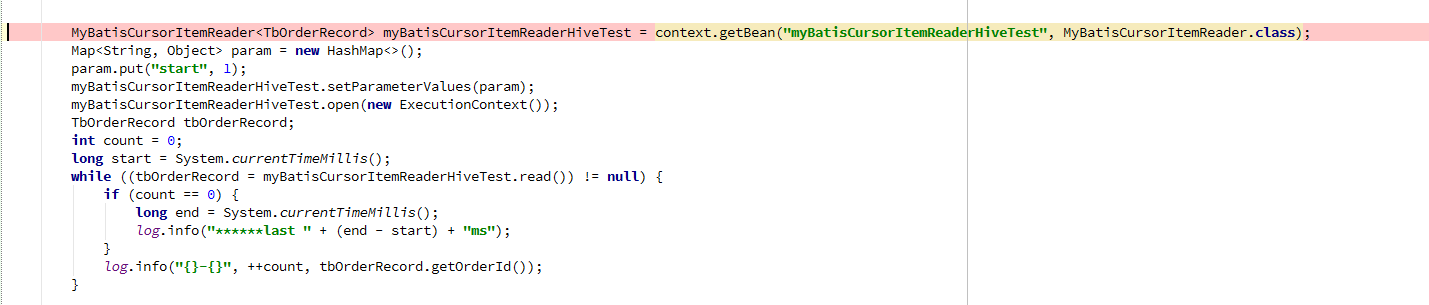
在线程栈中一路追查,发现如下代码需要遍历完RS的所有的结果,才能满足返回条件:

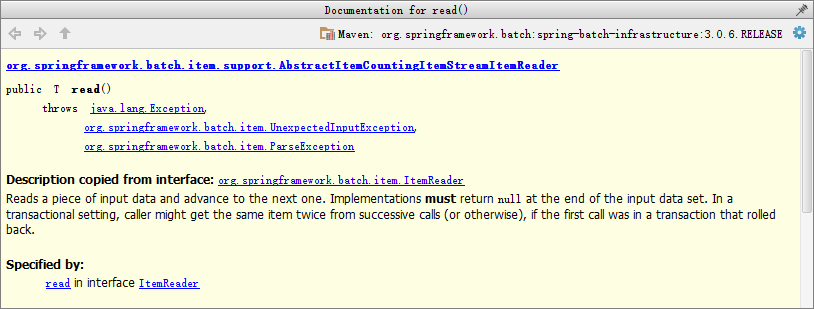




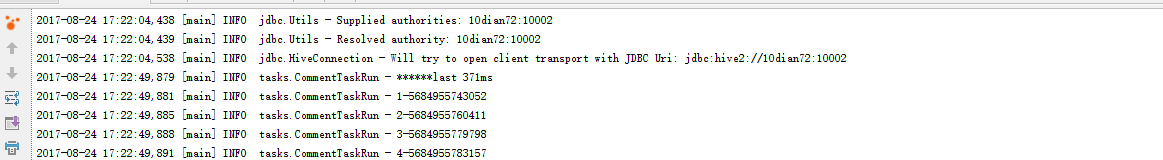
上图中的TbOrderRecord实例数在select没有返回的情况,数量在稳步上升(6000+到8000+),这样的情况对于资源是一种极大的浪费,在select没有遍历完所有数据之前,我们不能对已经完成的数据进行任何处理,那么我们能不能同时开始对已经完成的数据开始进行处理呢,答案是肯定的,一个方法是我们直接使用java的jdbc api,自己去控制ResultSet.next(对于mysql而言,要这样使用有一个前提条件,后面详述),但是这样就失去了mybatis的优点(结果映射,动态sql,sql统一管理等)得不偿失,好在mybatis已经在2016年的一次更新(版本3.4.0+)中加入了对stream的支持(DefaultCursor),对jdbc的stream进行了封装,由于我们项目是使用mybatis+spring进行的开发,这里我不直接使用DefaultCursor,而是使用mybatis+spring进行stream操作的实验(mybatis 3.4.0,mybatis-spring 1.3.0,spring-batch-infrastructure 3.0.6.RELEASE),首先要使用MyBatisCursorItemReader对要进行的查询进行一层封装,然后代码稍作改变,注意终止的条件判断.



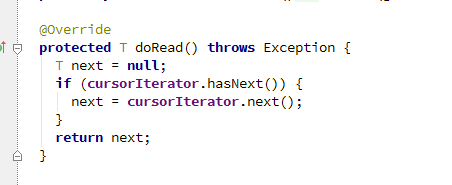




可以看到程序在数十秒之后就已经开始返回已经完成的数据了

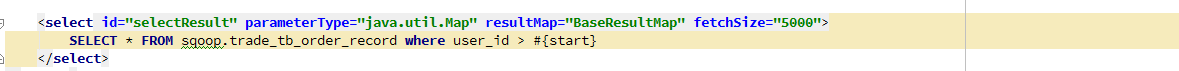


原因在于MyBatisCursorItemReader将rs的next暴露给了开发者,让开发者可以自己控制数据获取和遍历结束.



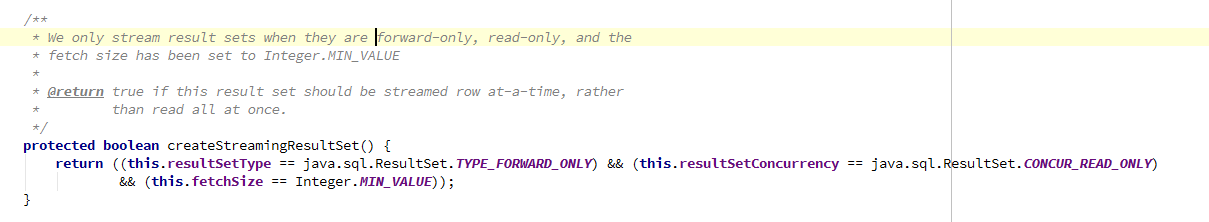
这个在有些场景尤其适用(比如从hive查出的统计结果数据量比较大,往mysql写入,真正实现边读边写,相信运行效率会大幅度提高).

另外从上图中发现的第二个问题是数据实体数目上涨速度比较慢,通过查找hive-jdbc发现在HiveQueryResultSet中的默认fetchSize是50(通过它自己的默认静态内部类Builder设置),这个fetchSize代表了我们程序每次网络交互从服务端拿回的数据条数,这个50明显在我们离线统计中偏小,调整后实验如下:



可以看到在数秒之内,查回来映射为对象的数据量已经10000,效率比之前的50每次的取高了很多,这个fetchSize的好处是减少了IO传输次数,坏处也很明显,需要服务端维护一个相应size的数据传送缓冲区,需要开发时找一个合适的平衡点

同理Mysql+mybatis stream也很适合一些场景(主键不是自增的,需要查询很多数据的表,一次查出全部会导致OOM,比如tb\_order\_record主键是order\_id,不能使用主键分段查询,使用limit分段查询到后面的后果大家也已经知晓),前文中提到的mysql要使用stream query需要做的特殊处理,在mysql驱动中的StatementImpl有详细的说明



也就是如下改造,即可支持straem query

