订单系统历史数据归档方案设计与实现

如何归档历史数据

订单数据会随着时间一直累积的数据，前面我们说过预估订单的数量每个月订单2000W，一年的订单数可达2.4亿，三年可达7.2亿。

数据量越大，数据库就会越慢，这是为什么？我们需要理解造成这个问题的根本原因。无论是“增、删、改、查”中的哪个操作，其本质都是查找数据，因为我们需要先找到数据，然后才能操作数据。

无论采用的是哪种存储系统，一次查询所耗费的时间,都取决于如下两个因素。

1）查找的时间复杂度。

2）数据总量。

查找的时间复杂度又取决于如下两个因素。

1）查找算法。

2）存储数据的数据结构。

这两个因素也是面试问题中经常考察的知识。 所以面试官并不是非要问一些“用不上”的问题来为难求职者，这些知识点不是用不上，而是求职者很多时候不知道怎么用。

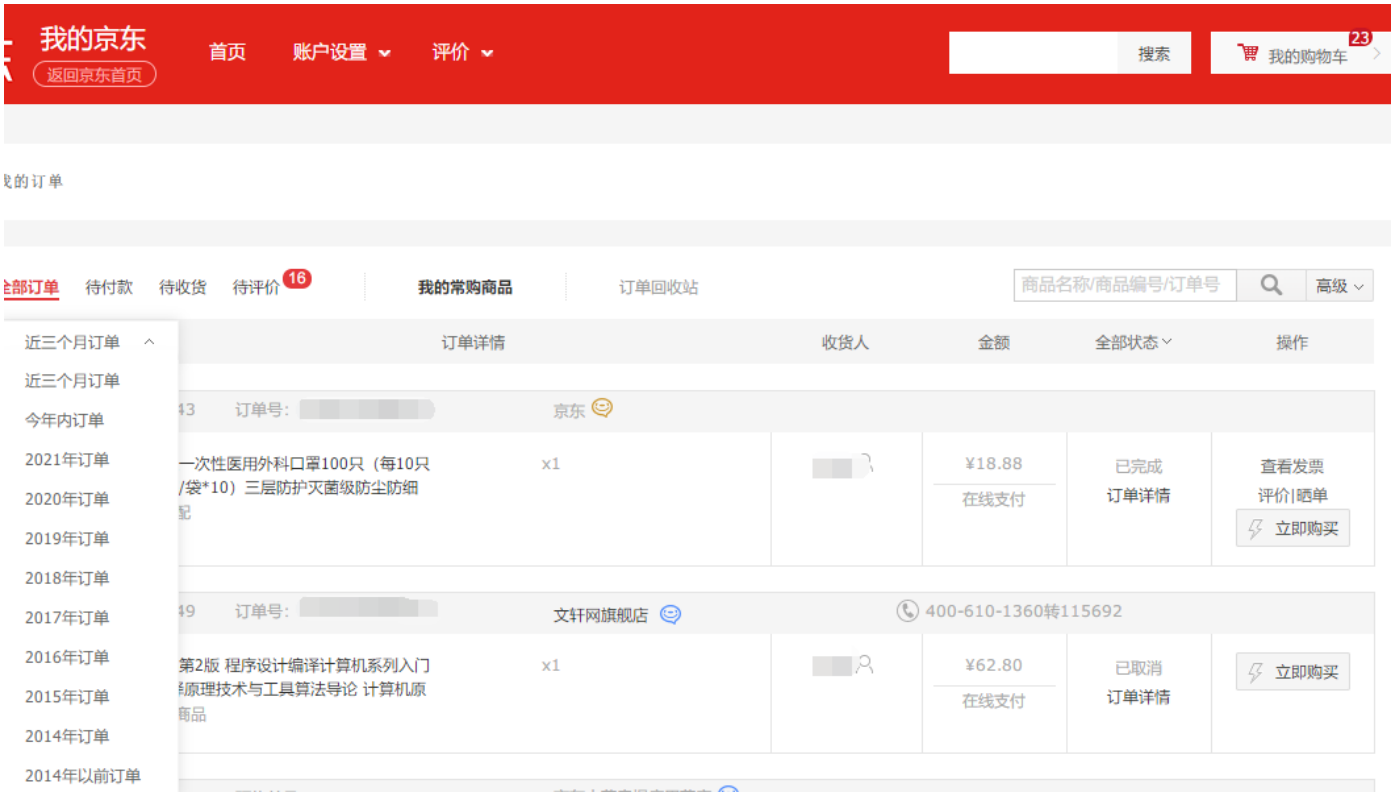
大多数做业务的系统，采用的都是现成的数据库，数据的存储结构和查找算法都是由数据库来实现的，对此，业务系统基本上无法做出任何改变。我们知道MySQL 的 InnoDB存储引擎，其存储结构是B+树，查找算法大多数时候是对树进行查找，查找的时间复杂度就是O(log n)，这些都是固定的。我们唯一能改变的就是数据总量了。

所以，解决海量数据导致存储系统慢的问题，方法非常简单，就是一个“拆”字，把大数据拆分成若干份小数据，学名称为“分片”(Shard)。拆开之后,每个分片里的数据就没那么多了，然后让查找尽量落在某一个分片上,以此来提升查找性能。

存档历史订单数据

订单数据一般保存在MySQL 的订单表里，说到拆分MySQL 的表，前面我们不是已经将到了“分库分表”吗？其实分库分表很多的时候并不是首选的方案，应该先考虑归档历史数据。

以京东为例



可以看到在“我的订单”中查询时，分为了近三个月订单、今年内订单、2021年订单、2020年订单等等，这就是典型的将订单数据归档处理。

所谓归档，也是一种拆分数据的策略。简单地说，就是把大量的历史订单移到另外一张历史订单表或数据存储中。为什这么做呢？订单数据有个特点：具备时间属性的，并且随着系统的运行，数据累计增长越来越多。但其实订单数据在使用上有个特点，最近的数据使用最频繁，超过一定时间的数据很少使用，这被称之为热尾效应。

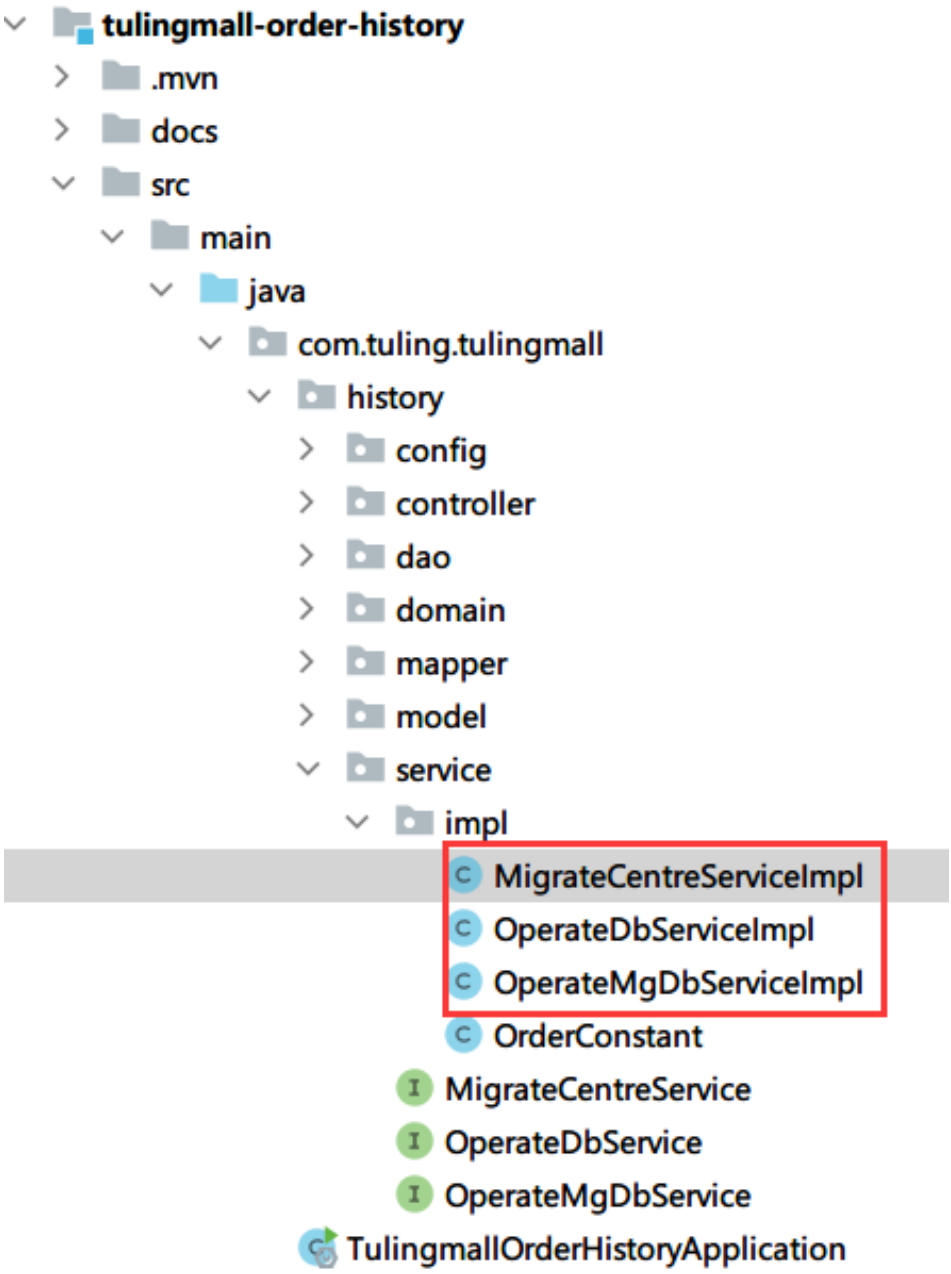
因为新数据只占数据息量中很少的一部分，所以把新老数据分开之后，新数据的数据量就少很多，查询速度也会因此快很多。虽然与之前的总量相比，老数据没有减少太多，但是因为老数据很少会被访问到，所以即使慢一点儿也不会有太大的问题，而且还可以使用其他的存储系统提升查询速度。

这样拆分数据的另外一个好处是，拆分订单时，系统需要改动的代码非常少。对订单表的大部分操作都是在订单完成之前执行的，这些业务逻辑都是完全不用修改的。即使是像退货退款这类订单完成之后的操作，也是有时限的，这些业务逻辑也不需要修改,还是按照之前那样操作订单即可。

基本上只有查询统计类的功能会查到历史订单，这些都需要稍微做些调整。按照查询条件中的时间范围，选择去订单表还是历史订单中查询就可以了。很多大型互联网电商在逐步发展壮大的过程中，长达数年的时间采用的都是这种订单拆分的方案，正如我们前面看到的京东就是如此。

商城历史订单服务的实现

商城历史订单的归档由tulingmall-order-history服务负责，其中比较关键的是三个Service



既然是历史订单的归档，归档到哪里去呢？我们可以归档到另外的MySQL数据库，也可以归档到另外的存储系统，这个看自己的业务需求即可，在我们的系统中，我们选择归档到MongoDB数据库。

对于数据的迁移归档，我们总是在MySQL中保留3个月的订单数据，超过三个月的数据则迁出。前面我们说过，预估每月订单2000W，一张订单下的商品平均为10个，如果只保留3个月的数据，则订单详情数为6亿，分布到32个表中，每个表容纳的记录数刚好在2000W左右，这也是为什么前面的分库分表将订单表设定为32个的原因。

在我们的实现中，OperateDbServiceImpl负责读取MySQL的订单数据和删除已迁出的订单，OperateMgDbServiceImpl负责将订单数据批量插入MongoDB，MigrateCentreServiceImpl负责进行调度服务。

在进行数据迁移的过程需要注意以下两点：

分布式事务?

考察迁移的过程，我们是逐表批次删除，对于每张订单表，先从MySQL从获得指定批量的数据，写入MongoDB，再从MySQL中删除已写入MongoDB的部分，这里存在着一个多源的数据操作，为了保证数据的一致性，看起来似乎需要分布式事务。但是其实这里并不需要分布式事务，解决的关键在于写入订单数据到MongoDB时，我们要记住同时写入当前迁入数据的最大订单ID，让这两个操作执行在同一个事务之中。

```
@Bean
MongoTransactionManager transactionManager(MongoDatabaseFactory factory){
    //事务操作配置
    TransactionOptions txnOptions = TransactionOptions.builder()
            .readPreference(ReadPreference.primary())
            .readConcern(ReadConcern.MAJORITY)
            .writeConcern(WriteConcern.MAJORITY)
            .build();
    return new MongoTransactionManager(factory,txnOptions);
}
```

```
@Transactional
public void saveToMgDb(List<OmsOrderDetail> orders, long curMaxOrderId,String tableName) {
    log.info("准备将表{}数据迁移入MongoDB，参数curMaxOrderId = {}",tableName,curMaxOrderId);
    mongoTemplate.insert(orders,OmsOrderDetail.class);
    /*记录本次迁移的最大订单ID，下次迁移时需要使用*/
    Query query = new Query(Criteria.where(ORDER_MAX_ID_KEY).is(tableName));
    Update update = new Update();
    update.set(ORDER_MAX_ID_KEY,curMaxOrderId);
    UpdateResult updateResult = mongoTemplate.upsert(query,update,MongoOrderId.class);
    log.info("已记录表{}本次迁移最大订单ID = {}",tableName,curMaxOrderId);
}
```

这样，在MySQL执行数据迁移时，总是去MongoDB中获得上次处理的最大OrderId，作为本次迁移的查询起始ID

```
/*获得上次处理的最大OrderId，作为本次迁移的起始ID*/
long currMaxOrderId = operateMgDbService.getMaxOrderId(factTableName);
log.info("本次表[{}]数据迁移查询记录起始ID = {}",factTableName,currMaxOrderId);
List<OmsOrderDetail> fetchRecords = operateDbService.getOrders(currMaxOrderId,
        tableNo,maxDate,FETCH_RECORD_NUMBERS);
```

当然数据写入MongoDB后，还要记得删除MySQL中对应的数据。

在这个过程中，我们需要注意的问题是，尽量不要影响线上的业务。迁移如此大量的数据，或多或少都会影响数据库的性能，因此应该尽量选择在闲时迁移而且每次数据库操作的记录数不宜太多。按照一般的经验，对MySQL的操作的记录条数每次控制在10000一下是比较合适，在我们的系统中缺省是2000条。更重要的是，迁移之前一定要做好备份，这样的话，即使不小心误操作了，也能用备份来恢复。

如何批量删除大量数据

在迁移历史订单数据的过程中，还有一个很重要的细节问题:如何从订单表中删除已经迁走的历史订单数据?

虽然我们是按时间迁出订单表中的数据，但是删除最好还是按ID来删除，并且同样要控制住每次删除的记录条数，太大的数量容易遇到错误。

```
delete from ${orderTableName} o
WHERE o.id >= #{minOrderId} and o.id <= #{maxOrderId}
order by id
```

这样每次删除的时候，由于条件变成了主键比较，而在MySQL的InnoDB存储引擎中，表数据结构就是按照主键组织的一棵B+树，同时B+树本身就是有序的，因此优化后不仅查找变得非常快,而且也不需要再进行额外的排序操作了。

为什么要加一个排序的操作呢？因为按ID排序后，每批删除的记录基本上都是ID连续的一批记录，由于B+树的有序性，这些ID相近的记录，在磁盘的物理文件上，大致也是存放在一起的，这样删除效率会比较高，也便于MySQL回收页。

关于大批量删除数据，还有一个点需要注意一下，执行删除语句后，最好能停顿一小会，因为删除后肯定会牵涉到大量的B+树页面分裂和合并，这个时候MySQL的本身的负载就不小了，停顿一小会，可以让MySQL的负载更加均衡。

> 有道云笔记链接： https://note.youdao.com/s/Bkf9izuB