11 | 分库分表化后如何满足多维度查询?

本模块的前几讲里围绕着分库分表，以及外部依赖治理的话题进行了讨论，通过上述方案来提升写业务的高可用和高性能。

但分库分表以及无状态存储也带来了另外一个问题，**即数据按路由规则分散后，如何满足无路由字段的多维度富查询**?

异构定制化实现

在“第 8 讲”，我介绍了一个关于订单模块的分库分表案例，我们先回顾一下该方案的处理方式：

- 在提交订单时，采用用户账号作为分库字段；
- 在查询时，只有携带用户账号的 SQL 才能直接执行；
- 在下单后，售卖商品的商家可能希望查询自己店铺里的所有订单，此时按用户维度的分库分表则不能满足上述查询需求。

为了满足和原有分库维度不一样的查询，最简单的方式是按新的维度异构一套数据，它的架构如下图 1 所示，数据异构可以采用在本专栏模块二中介绍的 Binlog 进行处理。

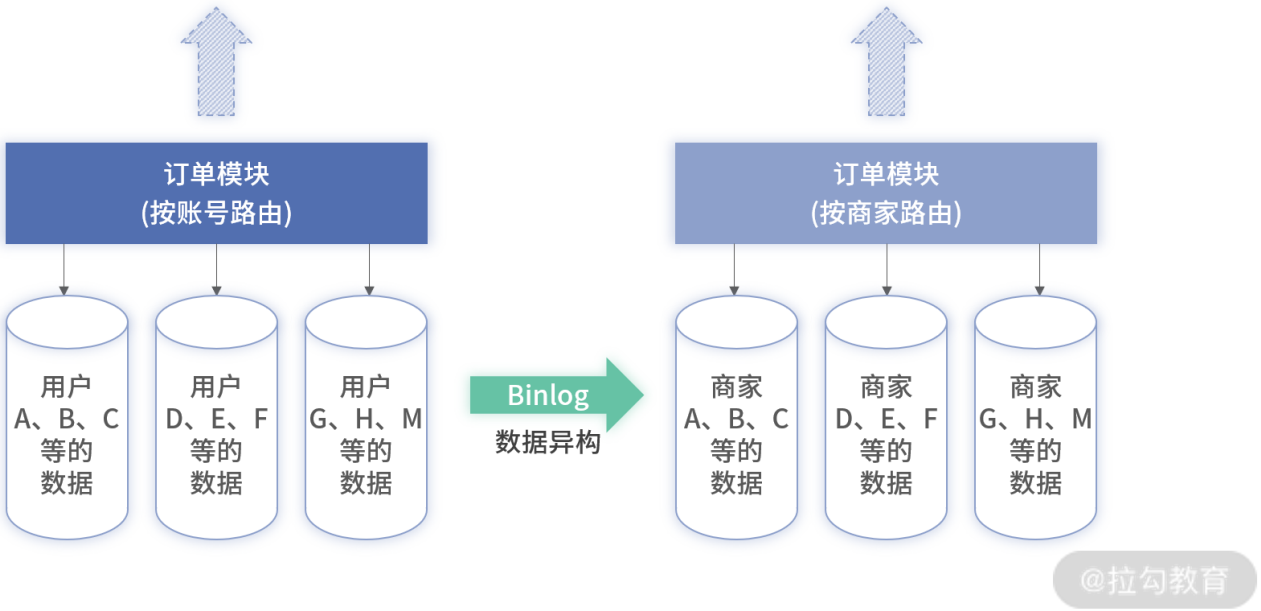

图 1：数据异构架构图

采用数据异构满足了上述按商家维度查看数据的诉求，但如果又来一个新的需求，需要按订单所属的来源（小程序、App、M 页或者 PC 站点）进行订单数据查询呢？此时，是否需要按来源维度进行数据异构呢?

答案显然是不行的，主要有两个原因：

- 一是数据同步程序需要开发，如果来一个新需求就开发一套同步，浪费人力成本；
- 二是异构数据浪费资源。正是因为数据量太大才进行分库分表，如果异构一套会导致数据量翻倍，资源消耗也会加倍。

本讲后面会介绍两种方案，来解决上述两个问题。

借助分库网关实现

在“第 8 讲”里，我们介绍了**代理式的分库分表的架构方案：分库代理中间件解析用户指定的 SQL 并提取路由字段，根据路由字段去访问具体的分库进行数据的查询**。如下图 2 所示：

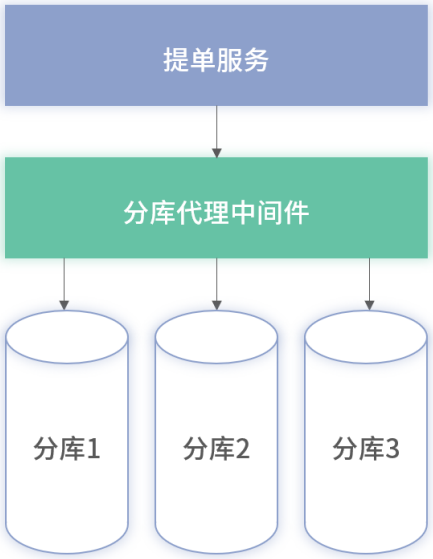

图 2：代理式分库分表方案

当用户没有指定路由字段时，可以在分库代理中间件进行转换处理。以订单为例，假设路由字段为用户账号，当查询时只指定了订单号，代理层无法计算到具体命中了哪个分库。但是代理层可以多线程并发地去请求所有的分库，查询此条订单信息。此方式，也可以查询到指定的订单信息。

但如果用户指定的查询带有排序和数量诉求，比如查询所有用户最近提交的 100 单，SQL 可能如下：

```
select 订单信息 from t_order order by createdTime limit 100
```

在没有路由字段时，分库分表的前 100 个订单如何获取呢？因为在极端情况下全局的前 100 条数据可能都分布在某一个分库里，为了保障一定能够获取到全局的前 100 条数据，代理层需要向每一个分库（上述有三个分库）都获取 100 条数据，并在代理层进行汇总排序，如下图 3 所示：

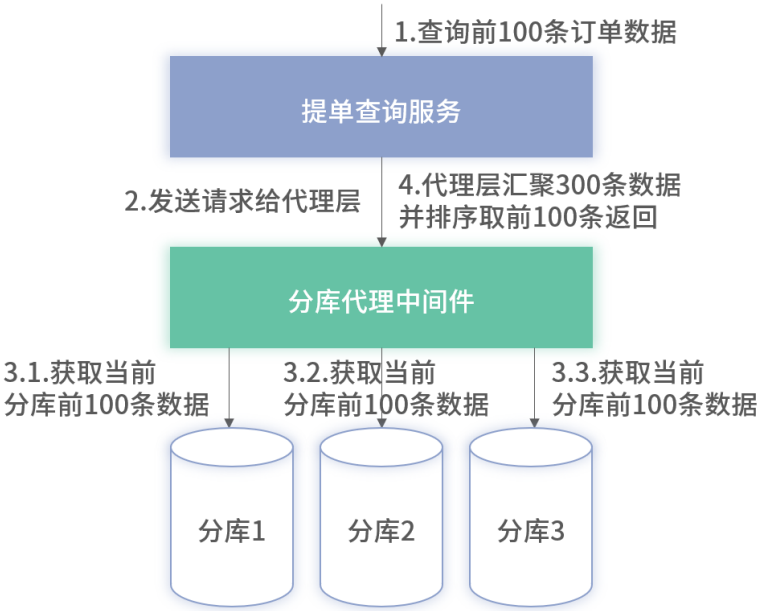

图 3：分库分表查询架构图

从上图中可以看到，不管是不带路由字段的条件查询还是排序聚合的查询，代理层都可以通过扫描分库来实现，比如上述的获取前 100 条订单数据。但在实现时，其实总共需要获取 300 条数据才能实现上述目标，这对于代理层的内存和 CPU 占用是非常巨大的，因为一次代理层的查询需要分裂出分库数量的查询，才能满足上述目标，这增加了调用量。

对于内嵌式的分库中间件的实现就更不行了，因为内嵌式的分库架构是和业务应用部署在同一台机器上的，它会消耗业务应用所在机器的网络、内存和 CPU 等资源，进而影响业务服务。

总的来说，数据库最重要的特征是为了满足写时的 ACID。对于读业务而言，数据库需要借助索引来提升性能。但过多的索引也会反过来导致写的性能变差，因为索引是在写入的时候实时构建的。因此，目前来看其实 MySQL+ 代理层并不十分适合。

基于 ElasticSearch 实现

借助分库网关+分库虽然能够实现多维度查询的能力，但整体上性能不佳且对正常的写入请求有一定的影响。除了上述的方案外，**业界应对多维度实时查询的最常见方式便是借助 ElasticSearch**。为了方便，后面都简称 ElasticSearch 为 ES。

什么是 ES

ES 是基于 Lucene 之上进行封装的可开箱即用的搜索引擎。其中，Lucene 提供了基于倒排序的全文索引的构建功能和查询的能力，但在更加贴近应用层的数据结构设计、存储架构层面涉及较少，它更多地被称为一个底层工具。

ES 在 Lucenne 的全文检索功能之外，还具备以下 3 个特点：

1. 自带了分布式的系统架构，能够很好地应对海量的数据，且分布式架构更加高可用，能够有效地满足生产环境的要求。
2. 支持带结构的数据（如数据库的 schema），提供了非常丰富的数据结构，可以直接映射数据库等存储的数据结构，更方便易用；
3. 另外是 ES 提供了近实时的数据索引功能，数据写入后就可以搜索查询，而不用像传统的搜索引擎要分钟级或更高的异构索引构建。

更加详细的介绍可以参考 ES 的官网，见这里，建议直接阅读最新版的文档。

什么是倒排序索引

倒排序索引的内容较多，此处只做简单介绍，帮助你理解本讲后续内容，有兴趣的同学可以深入研究。

对于倒排序索引，它是借助分词维护的二维表。以本讲的标题：“分库分表后如何满足多维度查询”（假设编号为 1）为例，写入 ES 后，会建立如下表格：

单词	文档 ID	说明
分库	1(1)	1(1)，前一个 1 表示文档编号 后一个“分库”这个词表示出现频率
分表	1(1)	同上
后	1(1)	同上
…	…	同上

上述的“单词”列得到的各个词，是和各个语言特定的，中文有中文分词器、英文有英文的分词器。

所有写入 ES 的内容，都会按上述模式进行分词。**相比数据库，ES 里所有的内容都可以分词建立索引且 ES 不需要保障数据的 ACID 等特性，因此 ES 整体上更适合查询类和模糊匹配等场景**。对于模糊匹配，数据库只能使用 like 等手段，性能是非常差的。

ES 里的所有内容都可以建立索引，虽然能带来提升性能的好处，但也会带来副作用，就是会非常消耗存储空间，这个在使用时需要预先考虑。

如何使用

在使用 ES 满足多维度查询时，第一步需要做的便是数据异构，将数据库的数据同步至 ES 中。在进行数据异构时，仍建议采用在模块二中介绍了**Binlog**进行：

- 一是因为 Binlog 可以保障数据最终一致性；
- 二是基于 Binlog 的方式，同步代码编写更加简单且不易出错。只需要订阅 Binlog 发出来的数据即可，不用在业务代码的每一个修改的地方进行特殊处理。

基于 Binlog 的 ES 数据异构如下图 4 所示：

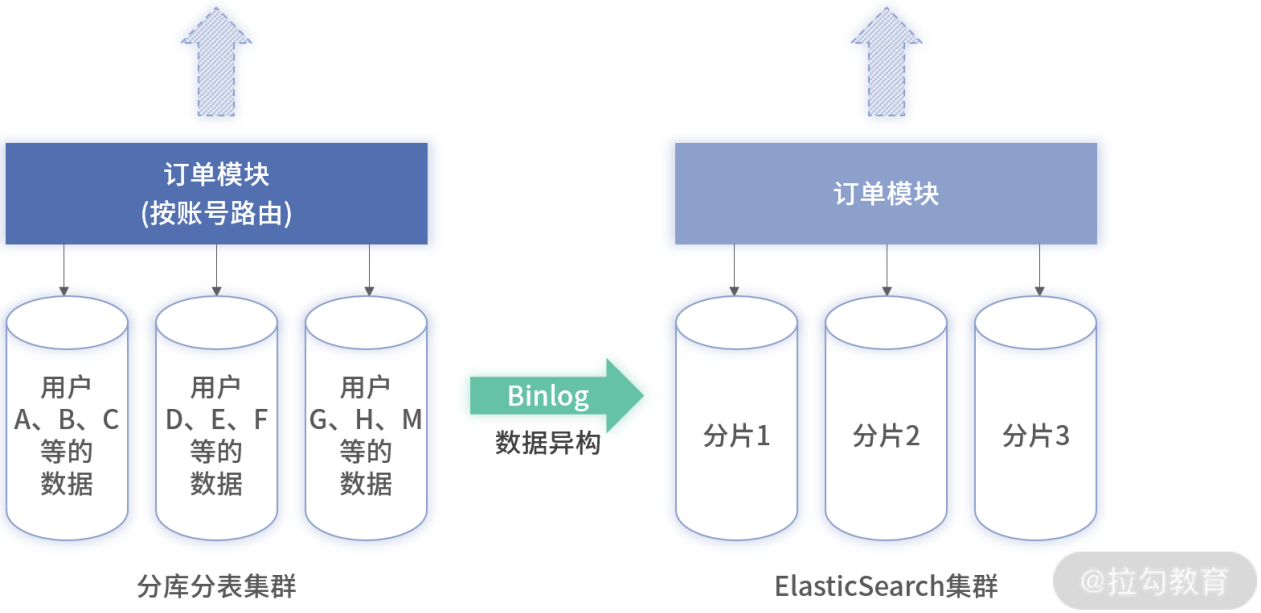

图 4：基于 Binlog 的 ES 数据异构图

上述异构的数据同步至 ES 时，ES 中的数据结构应该如何设计来满足存储呢？在正式设计前，需要搞明白 ES 中的几个重要概念。接下来将以数据库中的几个概念进行类比，如下表所示：

数据库	ES	描述
数据库 （Database）	索引 （Index）	
表 （Table）	类型 （Type）	
行 （Row）	文档 （Document）	
列 （Column）	字段 （Field）	

以数据库作为类比，你应该对 ES 中的几个概念非常清楚了，此处就不过多介绍了。我重点说一下 ES 中的类型（**Type**），**它并不完全对应数据库的表。数据库中的表与表之间是隔离的，没有关联的。而 ES 中同一个索引（Index）下的不同类型（Type）里，如果存在相同的字段，ES 会认为它们是同一个字段**。

这个隐含逻辑对熟悉数据库概念的用户来说，有很大的迷惑性。因此，ES 从版本 5 中已经逐渐将类型（Type）移除了。或者你可以直接这样简单理解，**ES 中一个索引就只能包含一个类型（Type）即可**，更加详细的介绍见 这里 。

了解了上述概念后，现在以一个实际的案例进行演练。以购物时的用户作为参考，用户数据库需要存储用户信息和用户的多个收货地址才能完成业务需求。数据库至少会有两张表，一张为用户表（user），另一张为收货地址表（delivery_address ），为一对多的关系。数据库表结构大致如下：

```
create table user{
  id bigint not null,
  user_id varchar(30) not null comment  '用户账号编号',
  nick_name varchar(50) not null,
  telephone_num varchar(50) not null,
  email varchar(80)
}
create table delivery_address{
  id bigint not null,
  user_id varchar(30) not null comment '用户账号编号',
  prov_id bigint not null,
  city_id bigint not null,
  county_id bigint not null,
  detail_address bigint not null
}
```

基于上述的数据库表结构，完成的 ES 的结构设计如下：

```
{
  "mappings": {
    "properties": {
      "user_id": {
        "type": "long"
      },
      "nick_name":{
        "type":"keyword"
      },
      "telephone_num":{
        "type":"keyword"
      },
      "email":{
        "type":"keyword"
      },
      "delivery_address": {
        "type": "nested",
        "properties":{
          "prov_id":{
            "type":"long"
          },
          "prov_name":{
            "type":"text"
          },
          "city_id":{"type":"long"},
          "city_name":{
            "type":"text"
          },
          "county_id":{
            "type":"long"
          },
          "county_name":{
            "type":"text"
          },
          "detail_address":{
            "type":"text"
          }
        }
      }
    }
  }
}
```

可以看到上述的 ES 结构和数据库中的表结构还是有一定的差异，具体的差异和产生的原因主要有以下几点。

1. 数据库中是一对多的两张表，而在 ES 中只用了一个冗余宽表（用户和用户的多个收货地址都放在一个 Document 结构里）。使用冗余宽表是因为**ES 即使在冗余的情况下，被冗余的收货地址仍然支持搜索**（上述的 Nested 关键字支持此特性），而数据库如果在用户表里设置了冗余字段存储用户的多个收货地址后，该冗余的地址字段就不支持查询了。此外，**ES 对于多张单独的 Document 的级联查询性能不好，ES 首推冗余存储**，更加详细的解释见 这里 。
2. 并不是所有的字段都设置了分词，比如电话号码就没有分词（使用了 keyword 关键字）。因为电话号码在业务上是不需要支持模糊匹配的。在你设计索引时也最好遵守此原则，**对于不需要模糊匹配的字段不设置分词，因为分词需要构建倒排序索引，浪**

费存储。

3. 在 ES 的收货地址结构里，增加了省份名称、市名称等，而数据库里没有。因为在实际业务场景里，有根据中文名称查询地址的需求。即使不分库分表，上述的数据库表结构里的字段也不能支持按名称查询，因为它没有存储省市县的名称。**ES 的目的就是面向查询，因此在设计 ES 结构时，需要根据查询需求冗余一些字段进来。**

ES 的架构与深翻页

在介绍 ES 架构前，首先要明确 ES 架构中的三个重要概念：**节点（Node）、分片（Shard）、集群（Cluster）**。

- 节点简单理解就是部署的机器，可以是物理机或者是 Docker。
- 分片类似数据库分库分表架构里的一个分库，存储一部分数据。此外，分片还分为主分片和副本分片。主分片类似数据库分库里的主分库，副本分片就是从分库。分片部署在节点里，一个节点可以包含一至多个分片。
- 多个节点在一起便组成了集群。

基于上述概念的架构如下图 5 所示：

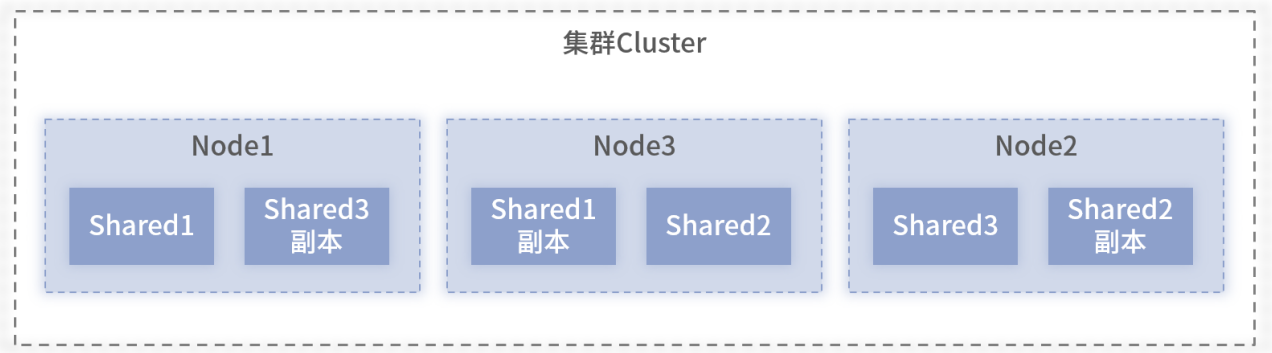

图 5：ES 的架构简介

可以看到 ES 的架构里没有代理式网关，ES 里所有的节点都可以接受用户的请求。对于类似第二小节里提到的排序+数量查询，**ES 和代理式分库分表的架构比较类似，接受请求的节点并行地去获取所有其他节点，并在该节点里进行集群排序和过滤**，具体流程如下图 6 所示：

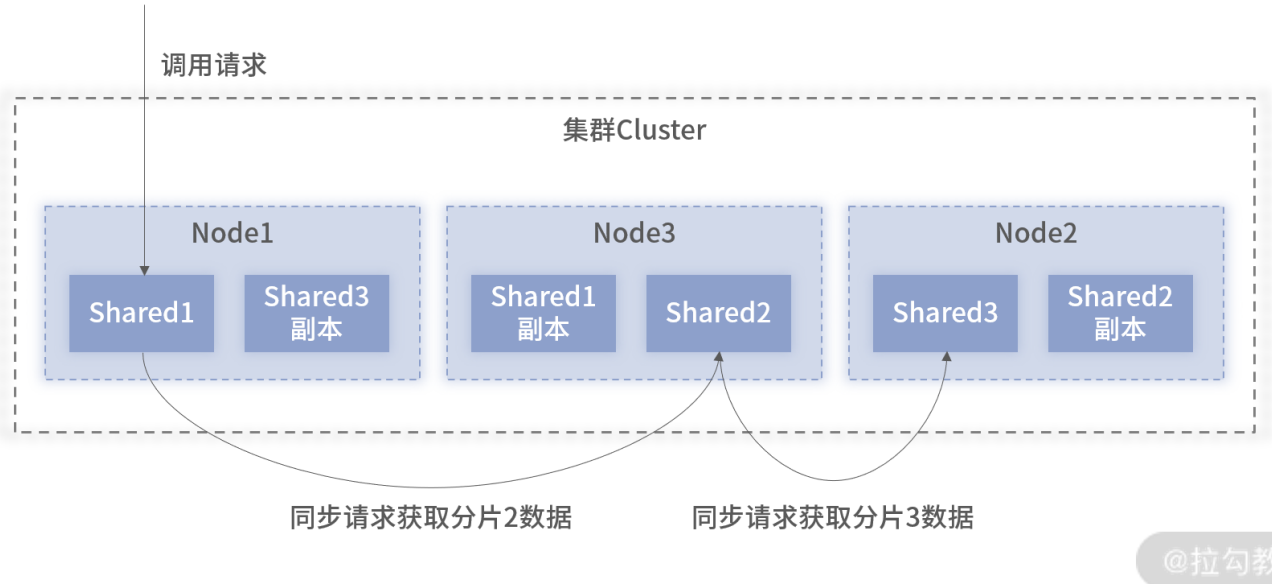

图 6：请求处理流程

虽然 ES 使用了倒排序增加了检索的性能，但如果你要搜索第 1000 条数据之后的 100 条数据，在接受请求的节点就需要获取 1100* 节点数量条数据，即使如 ES 这种面向查询的存储也是搞不定的。因此，**ES 默认有一个设置，最多只能查询 10000 条数据，超过了直接报错**。

上述描述的案例，有一个通俗的叫法：**深翻页**。对于深翻页，不管是 ES 和代理式网关都是无法直接支持的。解决上述问题，有一个牺牲用户体验的做法，就是按游标查询，或者叫每次查询都带上上一次查询经过排序后的最大 ID。以 SQL 举例，大致语法如下：

```
select 内容 from table where id >lastMaxId order by id limit pageSize
```

上述的有损用户体验主要体现在，用户无法指定页码进行翻页，只能在文章列表里一页一页地翻。

注意：ES 是近实时的但不是实时的，默认有 1s 的延迟。所以需要你根据具体业务情况进行取舍考虑。

总结

本讲介绍了从最简单、但资源消耗严重的异构定制化的方案，到使用 ES 来最终应对多维度查询的方案。各个方案都有各自的好处，但也有带来负面的影响。比如要使用 ES，你就需要学习 ES 的知识并要有专业的人去维护它。

此外，在 ES 的方案里，我引用了很多英文的官方文档。这也是我想给你重点强调的，**学习一项新的技能，需要从源头获取信息，而不是借助搜索引擎，把东一点西一点的信息拼凑起来零零散散地学习**。通过在源头获取原作者体系化的、第一手真实的信息，能让你的学习事半功倍。当然，直接阅读英文原生文档可能会点难，但这也是你成为架构师必备的技能。

最后，留一道讨论题。你所负责的系统在分库分表之后是如何满足多维度查询的？欢迎留言区留言，我们一起交流。

下一讲将介绍12 | 如何利用数据库实现并发扣减?