我们为什么要分库分表?

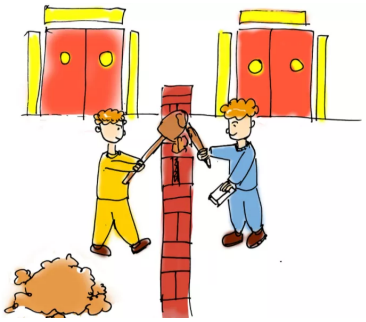
原创 雷架 爱笑的架构师 1月25日

收录于话题
#图解系列

13个

点击关注"爱笑的架构师"

右上角"设为星标"好文章送达比别人快



在文章开头先抛几个问题:

(1)什么时候才需要分库分表呢?我们的评判标准是什么?

(2)一张表存储了多少数据的时候,才需要考虑分库分表?

(3)数据增长速度很快,每天产生多少数据,才需要考虑做分库分表?

这些问题你都搞清楚了吗?相信看完这篇文章会有答案。

为什么要分库分表?

首先回答一下为什么要分库分表,答案很简单: 数据库出现性能瓶颈 。用大白话来说就是数据库快扛不住了。

数据库出现性能瓶颈,对外表现有几个方面:

- 大量请求阻塞

  在高并发场景下,大量请求都需要操作数据库,导致连接数不够了,请求处于阻塞状态。

- SQL 操作变慢

如果数据库中存在一张上亿数据量的表，一条 SQL 没有命中索引会全表扫描，这个查询耗时会非常久。

- 存储出现问题

  业务量剧增，单库数据量越来越大，给存储造成巨大压力。

从机器的角度看，性能瓶颈无非就是CPU、内存、磁盘、网络这些，要解决性能瓶颈最简单粗暴的办法就是提升机器性能，但是通过这种方法成本和收益投入比往往又太高了，不划算，所以重点还是要从软件角度入手。

数据库相关优化方案

数据库优化方案很多，主要分为两大类：软件层面、硬件层面。

软件层面包括：SQL 调优、表结构优化、读写分离、数据库集群、分库分表等；

硬件层面主要是增加机器性能。

SQL 调优

SQL 调优往往是解决数据库问题的第一步，往往投入少部分精力就能获得较大的收益。

SQL 调优主要目的是尽可能的让那些慢 SQL 变快，手段其实也很简单就是让 SQL 执行尽量命中索引。

开启慢 SQL 记录

如果你使用的是 Mysql，需要在 Mysql 配置文件中配置几个参数即可。

```
slow_query_log=on
long_query_time=1
slow_query_log_file=/path/to/log
```

调优的工具

常常会用到 explain 这个命令来查看 SQL 语句的执行计划，通过观察执行结果很容易就知道该 SQL 语句是不是全表扫描、有没有命中索引。

```
select id, age, gender from  user where name = '爱笑的架构师';
```

返回有一列叫“type”，常见取值有：

ALL、index、range、 ref、eq_ref、const、system、NULL（从左到右，性能从差到好）

ALL 代表这条 SQL 语句全表扫描了，需要优化。一般来说需要达到range 级别及以上。

表结构优化

以一个场景举例说明：

“user”表中有 user_id、nickname 等字段，“order”表中有order_id、user_id等字段，如果想拿到用户昵称怎么办？一般情况是通过 join 关联表操作，在查询订单表时关联查询用户表，从而获取导用户昵称。

但是随着业务量增加，订单表和用户表肯定也是暴增，这时候通过两个表关联数据就比较费力了，为了取一个昵称字段而不得不关联查询几十上百万的用户表，其速度可想而知。

这个时候可以尝试将 nickname 这个字段加到 order 表中（order_id、user_id、nickname），这种做法通常叫做数据库表冗余字段。这样做的好处展示订单列表时不需要再关联查询用户表了。

冗余字段的做法也有一个弊端，如果这个字段更新会同时涉及到多个表的更新，因此在选择冗余字段时要尽量选择不经常更新的字段。

架构优化

当单台数据库实例扛不住，我们可以增加实例组成集群对外服务。

当发现读请求明显多于写请求时，我们可以让主实例负责写，从实例对外提供读的能力；

如果读实例压力依然很大，可以在数据库前面加入缓存如 redis，让请求优先从缓存取数据减少数据库访问。

缓存分担了部分压力后，数据库依然是瓶颈，这个时候就可以考虑分库分表的方案了，后面会详细介绍。

硬件优化

硬件成本非常高，一般来说不可能遇到数据库性能瓶颈就去升级硬件。

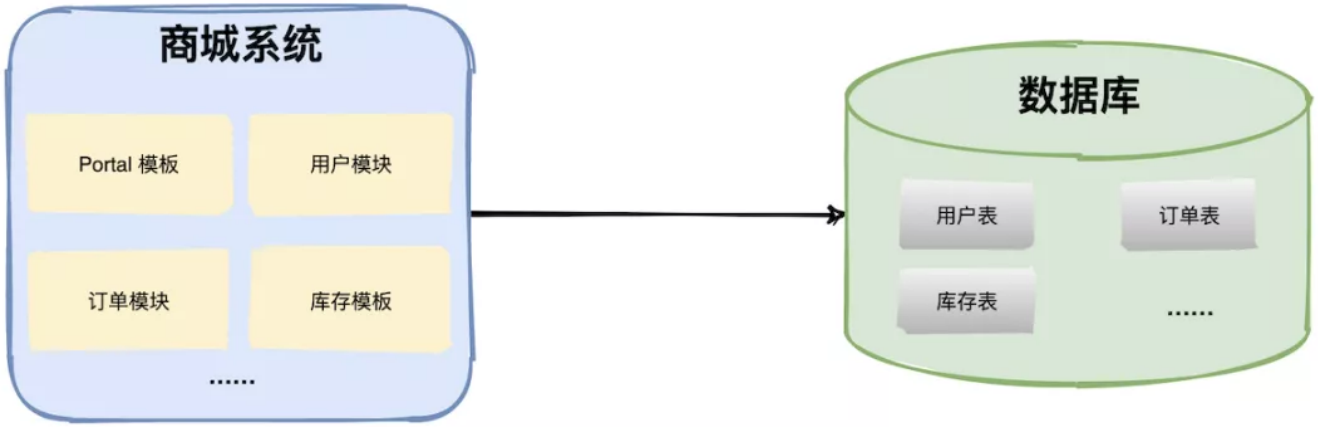
在前期业务量比较小的时候，升级硬件数据库性能可以得到较大提升；但是在后期，升级硬件得到的收益就不那么明显了。

分库分表详解

下面我们以一个商城系统为例逐步讲解数据库是如何一步步演进。

单应用单数据库

在早期创业阶段想做一个商城系统，基本就是一个系统包含多个基础功能模块，最后打包成一个 war 包部署，这就是典型的单体架构应用。



商城项目使用单数据库

如上图，商城系统包括主页 Portal 模板、用户模块、订单模块、库存模块等，所有的模块都共有一个数据库，通常数据库中有非常多的表。

因为用户量不大，这样的架构在早期完全适用，开发者可以拿着 demo到处找（骗）投资人。

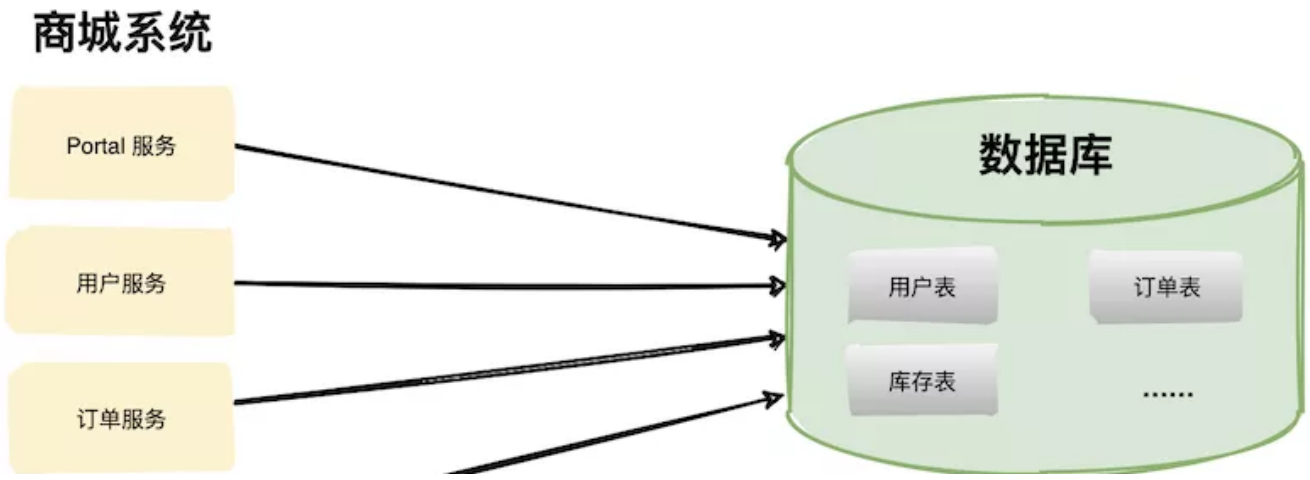
一旦拿到投资人的钱，业务就要开始大规模推广，同时系统架构也要匹配业务的快速发展。

多应用单数据库

在前期为了抢占市场，这一套系统不停地迭代更新，代码量越来越大，架构也变得越来越臃肿，现在随着系统访问压力逐渐增加，系统拆分就势在必行了。

为了保证业务平滑，系统架构重构也是分了几个阶段进行。

第一个阶段将商城系统单体架构按照功能模块拆分为子服务，比如：Portal 服务、用户服务、订单服务、库存服务等。



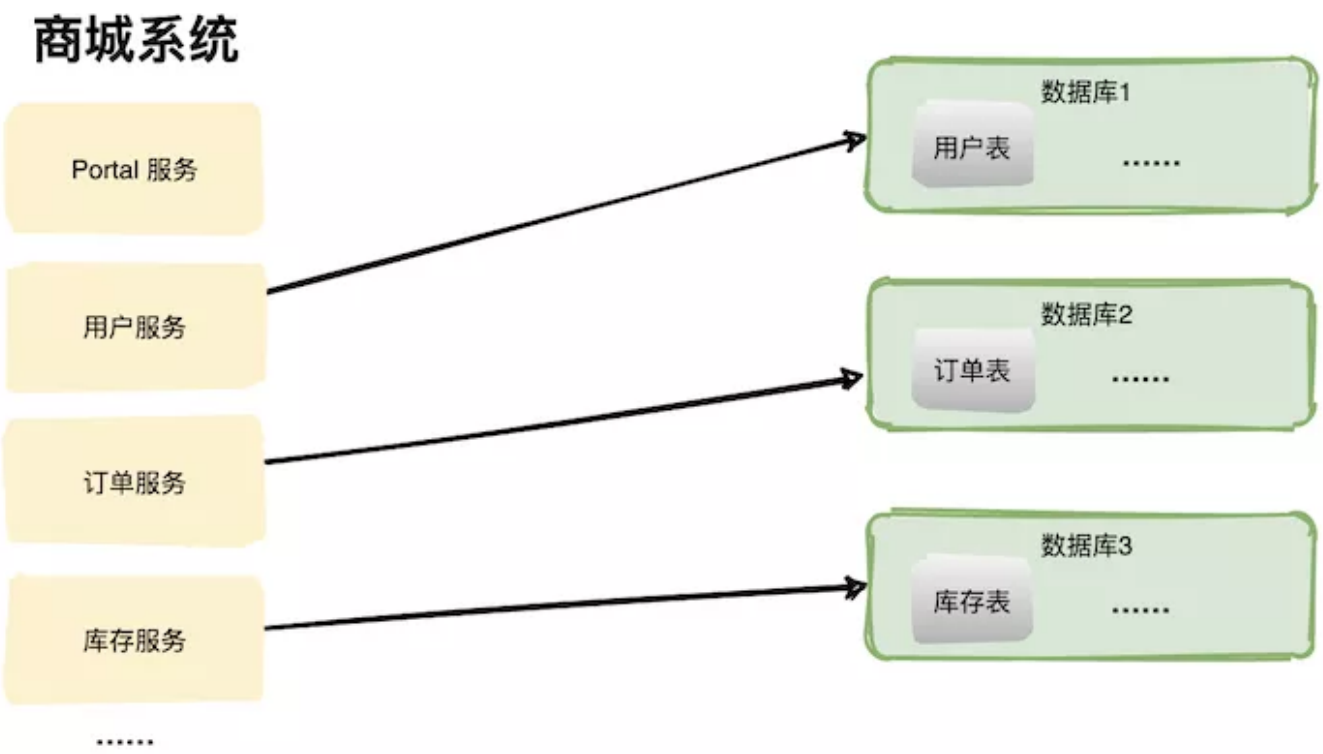
多应用单数据库

如上图，多个服务共享一个数据库，这样做的目的是底层数据库访问逻辑可以不用动，将影响降到最低。

多应用多数据库

随着业务推广力度加大，数据库终于成为了瓶颈，这个时候多个服务共享一个数据库基本不可行了。我们需要将每个服务相关的表拆出来单独建立一个数据库，这其实就是“分库”了。

单数据库的能够支撑的并发量是有限的，拆成多个库可以使服务间不用竞争，提升服务的性能。



多应用多数据库

如上图，从一个大的数据中分出多个小的数据库，每个服务都对应一个数据库，这就是系统发展到一定阶段必要要做的“分库”操作。

现在非常火的微服务架构也是一样的，如果只拆分应用不拆分数据库，不能解决根本问题，整个系统也很容易达到瓶颈。

分表

说完了分库，那什么时候分表呢?

如果系统处于高速发展阶段，拿商城系统来说，一天下单量可能几十万，那数据库中的订单表增长就特别快，增长到一定阶段数据库查询效率就会出现明显下降。

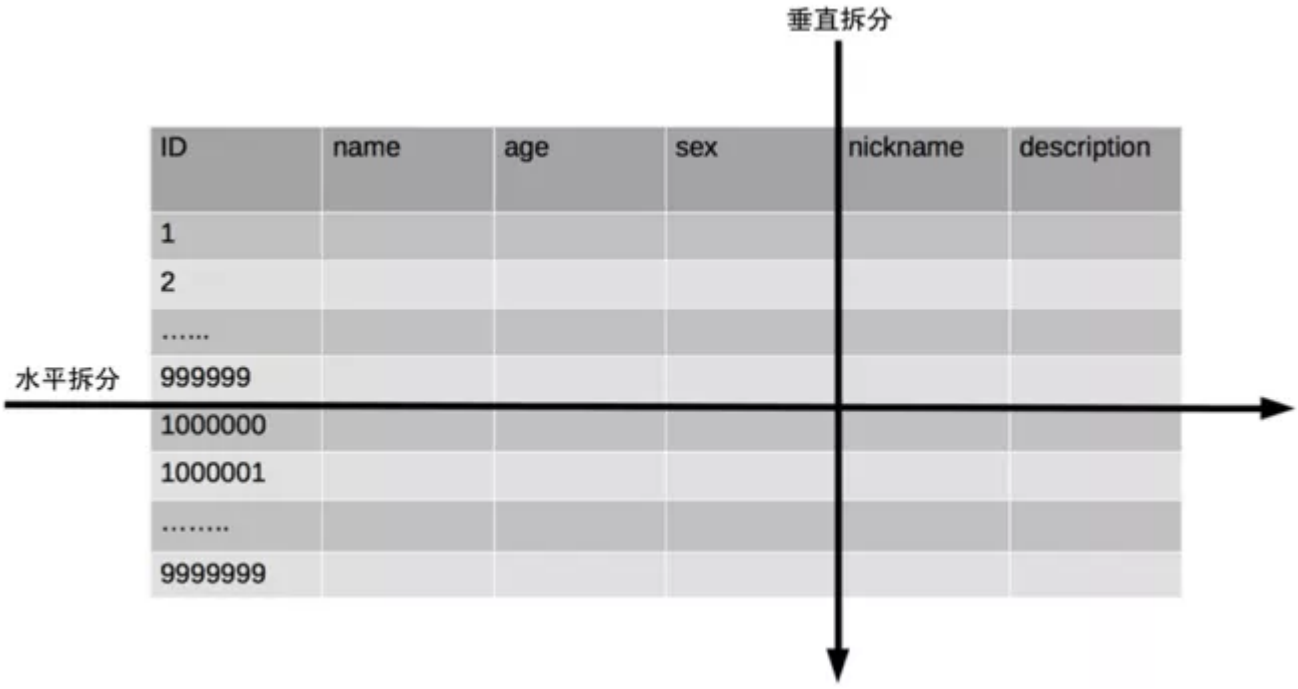
因此，当单表数据增量过快，业界流传是超过500万的数据量就要考虑分表了。当然500万只是一个经验值，大家可以根据实际情况做出决策。

那如何分表呢?

分表有几个维度，一是水平切分和垂直切分，二是单库内分表和多库内分表。

水平拆分和垂直拆分

就拿用户表（user）来说，表中有7个字段：id,name,age,sex,nickname,description，如果 nickname 和 description 不常用，我们可以将其拆分为另外一张表：用户详细信息表，这样就由一张用户表拆分为了用户基本信息表+用户详细信息表，两张表结构不一样相互独立。但是从这个角度来看垂直拆分并没有从根本上解决单表数据量过大的问题，因此我们还是需要做一次水平拆分。



拆分表

还有一种拆分方法，比如表中有一万条数据，我们拆分为两张表，id 为奇数的：1，3，5，7.....放在 user1，id 为偶数的：2，4，6，8.....放在 user2中，这样的拆分办法就是水平拆分了。

水平拆分的方式也很多，除了上面说的按照 id 拆表，还可以按照时间维度取拆分，比如订单表，可以按每日、每月等进行拆分。

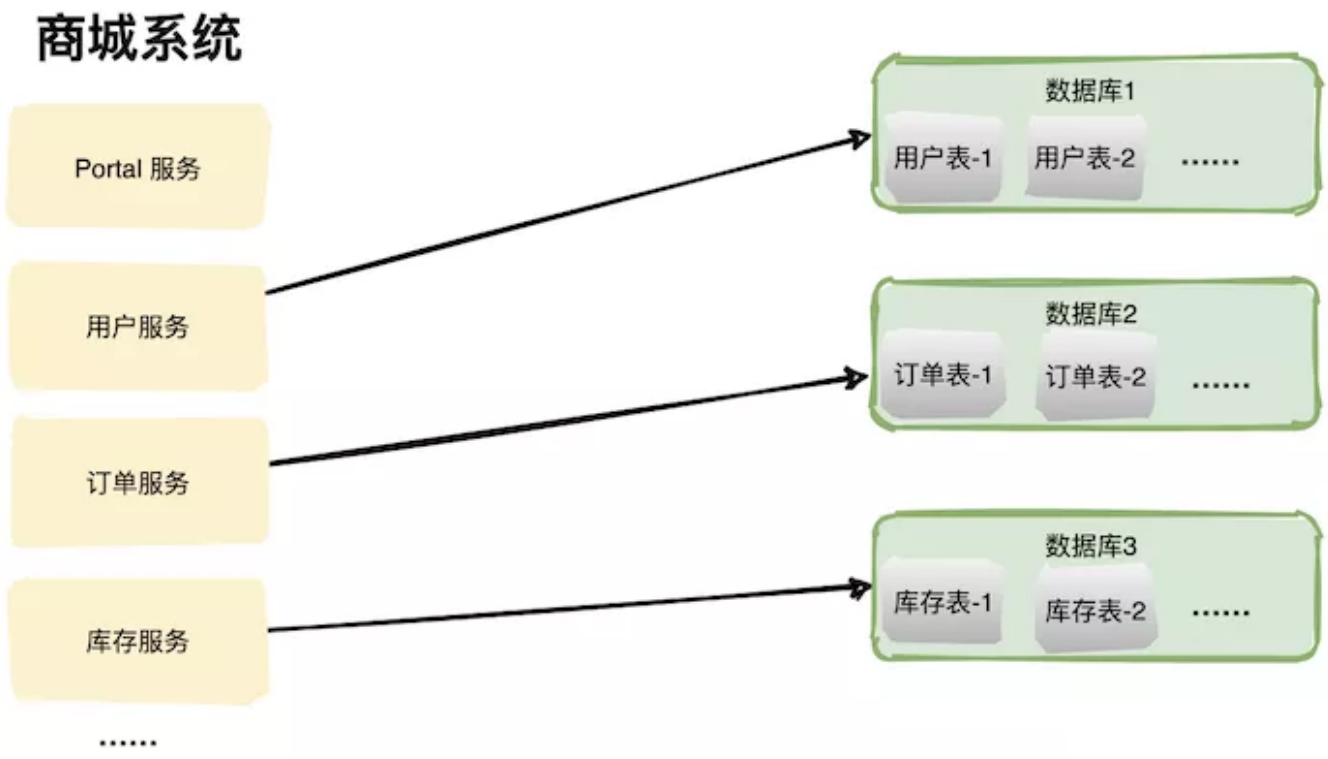
- 每日表：只存储当天的数据。
- 每月表：可以起一个定时任务将前一天的数据全部迁移到当月表。
- 历史表：同样可以用定时任务把时间超过 30 天的数据迁移到 history表。

总结一下水平拆分和垂直拆分的特点：

- 垂直切分：基于表或字段划分，表结构不同。
- 水平切分：基于数据划分，表结构相同，数据不同。

单库内拆分和多库拆分

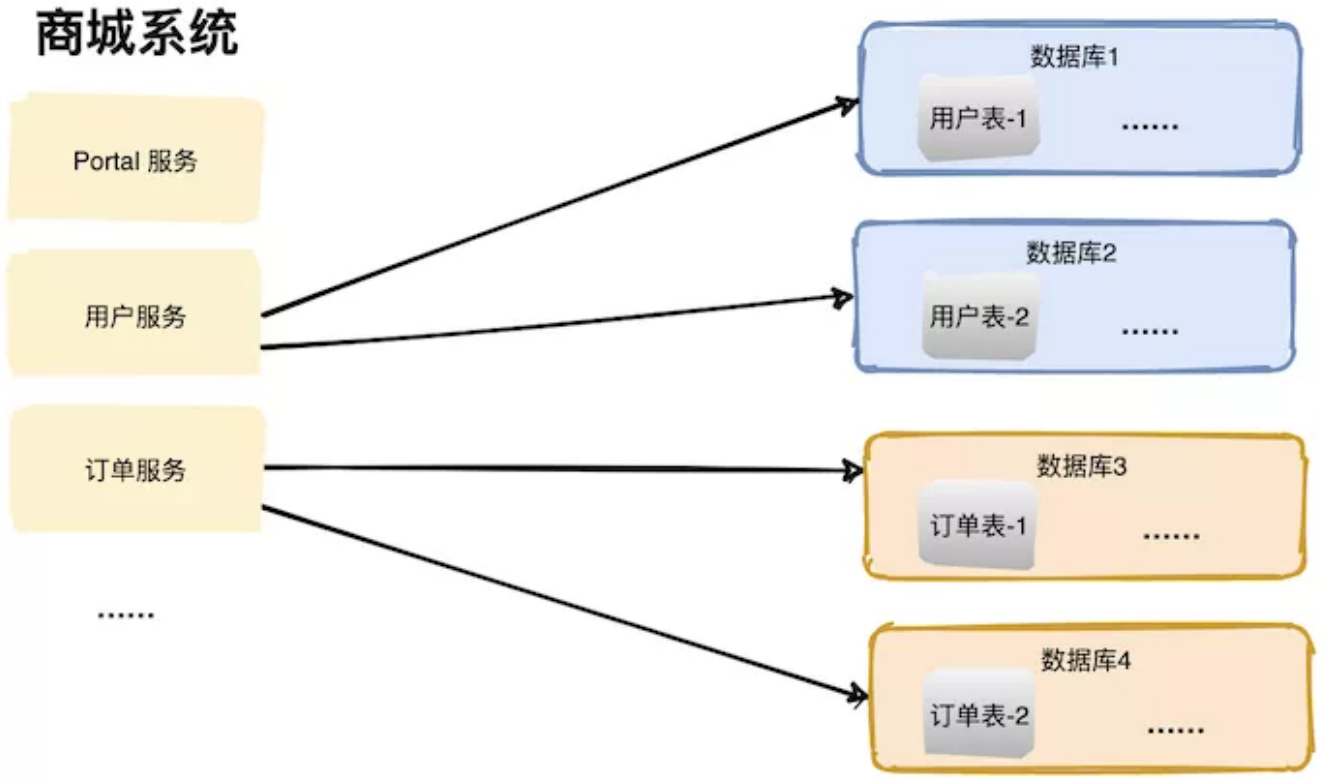
拿水平拆分为例，每张表都拆分为了多个子表，多个子表存在于同一数据库中。比如下面用户表拆分为用户1表、用户2表。



单库拆分

在一个数据库中将一张表拆分为几个子表在一定程度上可以解决单表查询性能的问题，但是也会遇到一个问题：单数据库存储瓶颈。

所以在业界用的更多的还是将子表拆分到多个数据库中。比如下图中，用户表拆分为两个子表，两个子表分别存在于不同的数据库中。



多库拆分

一句话总结：分表主要是为了减少单张表的大小，解决单表数据量带来的性能问题。

分库分表带来的复杂性

既然分库分表这么好，那我们是不是在项目初期就应该采用这种方案呢？不要激动，冷静一下，分库分表的确解决了很多问题，但是也给系统带来了很多复杂性，下面简要说一说。

（1）跨库关联查询

在单库未拆分表之前，我们可以很方便使用 join 操作关联多张表查询数据，但是经过分库分表后两张表可能都不在一个数据库中，如何使用 join 呢？

有几种方案可以解决：

- 字段冗余：把需要关联的字段放入主表中，避免 join 操作；
- 数据抽象：通过ETL等将数据汇合聚集，生成新的表；
- 全局表：比如一些基础表可以在每个数据库中都放一份；
- 应用层组装：将基础数据查出来，通过应用程序计算组装；

（2）分布式事务

单数据库可以用本地事务搞定，使用多数据库就只能通过分布式事务解决了。

常用解决方案有：基于可靠消息（MQ）的解决方案、两阶段事务提交、柔性事务等。

（3）排序、分页、函数计算问题

在使用 SQL 时 order by， limit 等关键字需要特殊处理，一般来说采用分片的思路：

先在每个分片上执行相应的函数，然后将各个分片的结果集进行汇总和再次计算，最终得到结果。

（4）分布式 ID

如果使用 Mysql 数据库在单库单表可以使用 id 自增作为主键，分库分表了之后就不行了，会出现id 重复。

常用的分布式 ID 解决方案有：

- UUID
- 基于数据库自增单独维护一张 ID表
- 号段模式
- Redis 缓存
- 雪花算法（Snowflake）
- 百度uid-generator
- 美团Leaf
- 滴滴Tinyid

这些方案后面会写文章专门介绍，这里不再展开。

（5）多数据源

分库分表之后可能会面临从多个数据库或多个子表中获取数据，一般的解决思路有：客户端适配和代理层适配。

业界常用的中间件有：

- shardingsphere（前身 sharding-jdbc）
- Mycat

总结

如果出现数据库问题不要着急分库分表，先看一下使用常规手段是否能够解决。

分库分表会给系统带来巨大的复杂性，不是万不得已建议不要提前使用。作为系统架构师可以让系统灵活性和可扩展性强，但是不要过度设计和超前设计。在这一点上，架构师一定要有前瞻性，提前做好预判。大家

学会了吗?

- END -

推荐阅读:

牛逼！简单的代码提交能玩出这么多花样

高并发场景下，到底先更新缓存还是先更新数据库?

Docker不香吗？为什么还要用k8s

日常求赞求关注环节:

需要与我深入交流的可以加我私人微信，想进技术交流群的可以备注【学技术】。最后的最后，恳求大家点赞+分享，雷架需要你们帮忙，爱了，爱了♡

听说点个"在看"就有好运气~

收藏算白嫖，点赞、再看、分享走一波才是真爱~

真粉还会去留言区留个言😄

收录于话题 #图解系列·13个

上一篇

用大白话给你解释Zookeeper的选举机制

下一篇

高并发场景下，到底先更新缓存还是先更新数据库?

喜欢此内容的人还喜欢

Kafka核心知识总结!

月伴飞鱼

大厂经典面试题：Redis为什么这么快?

捡田螺的小男孩