# Mysql分库分表

在互联网还未崛起的时代,我们的传统应用都有这样一个特点：访问量、数据量都比较小，单库单表都完全可以支撑整个业务。随着互联网的发展和用户规模的迅速扩大,对系统的要求也越来越高。因此传统的MySQL单库单表架构的性能问题就暴露出来了。而有下面几个因素会影响数据库性能:

* 数据量  
  MySQL单库数据量在5000万以内性能比较好,超过阈值后性能会随着数据量的增大而变弱。MySQL单表的数据量是500w-1000w之间性能比较好,超过1000w性能也会下降。
* 磁盘  
  因为单个服务的磁盘空间是有限制的,如果并发压力下,所有的请求都访问同一个节点,肯定会对磁盘IO造成非常大的影响。
* 数据库连接  
  数据库连接是非常稀少的资源,如果一个库里既有用户、商品、订单相关的数据,当海量用户同时操作时,数据库连接就很可能成为瓶颈。

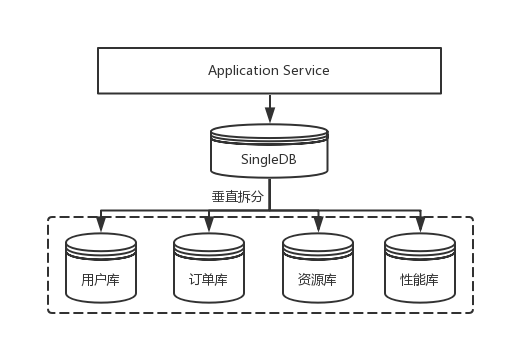
为了提升性能,所以我们必须要解决上述几个问题,那就有必要引进分库分表。

#### 垂直拆分 or 水平拆分？

当我们单个库太大时,我们先要看一下是因为表太多还是数据量太大，如果是表太多,则应该将部分表进行迁移(可以按业务区分),这就是所谓的垂直切分。如果是数据量太大,则需要将表拆成更多的小表,来减少单表的数据量,这就是所谓的水平拆分。

#### 垂直拆分

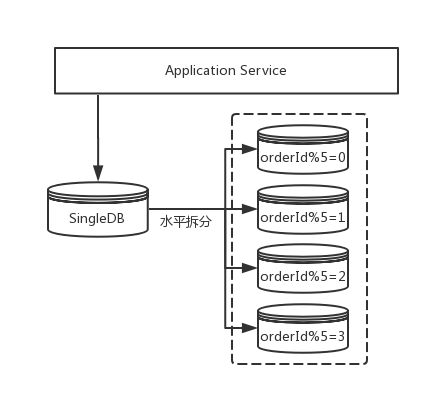
* 垂直分库  
  垂直分库针对的是一个系统中的不同业务进行拆分,比如用户一个库,商品一个库,订单一个库。 一个购物网站对外提供服务时,会同时对用户、商品、订单表进行操作。没拆分之前, 全部都是落到单一的库上的,这会让数据库的单库处理能力成为瓶颈。如果垂直分库后还是将用户、商品、订单放到同一个服务器上,只是分到了不同的库,这样虽然会减少单库的压力,但是随着用户量增大,这会让整个数据库的处理能力成为瓶颈,还有单个服务器的磁盘空间、内存也会受非常大的影响。 所以我们要将其拆分到多个服务器上，这样上面的问题都解决了，以后也不会面对单机资源问题。



* 垂直分表  
  也就是“大表拆小表”，基于列字段进行的。一般是表中的字段较多，将不常用的， 数据较大，长度较长（比如text类型字段）的拆分到“扩展表“。一般是针对那种几百列的大表，也避免查询时，数据量太大造成的“跨页”问题。

#### 水平拆分

* 水平分表  
  和垂直分表有一点类似,不过垂直分表是基于列的,而水平分表是基于全表的。水平拆分可以大大减少单表数据量,提升查询效率。
* 水平分库分表  
  将单张表的数据切分到多个服务器上去，每个服务器具有相应的库与表，只是表中数据集合不同。 水平分库分表能够有效的缓解单机和单库的性能瓶颈和压力，突破IO、连接数、硬件资源等的瓶颈。



**分片字段的选择？**

原则：尽量避免跨库跨表查询

#### 几种常用的分库分表的策略

* HASH取模  
  假设有用户表user,将其分成3个表user0,user1,user2.路由规则是对3取模,当uid=1时,对应到的是user1,uid=2时,对应的是user2.
* 范围分片

假设有用户表user,将其分成3个表user0,user1,user2  
从uid范围 1-10000的放到user0表,10001-20000的放到user1表，20001-30000的放到user2表。

时间分片  
按月分片，按季度分片等等,可以做到冷热数据。

# 分库分表案例

博客表：t\_blog

t\_blog字段：blog\_id、user\_id、title、author、create\_time

分库分表方案：

* 1. 按照创建日期、

T\_blog\_2019、T\_blog\_2020

T\_blog\_201901、T\_blog\_201902

缺点：按照博客id查询、按用户id查询需要全表扫描

* 1. 按blog\_id hash取模

T\_blog\_0、T\_blog\_1、T\_blog\_2、T\_blog\_3

缺点：按用户id查询需要跨表

* 1. 按user\_id hash取模

T\_blog\_0、T\_blog\_1、T\_blog\_2、T\_blog\_3

缺点：按照博客id需要全表扫描

* 1. 最终方案

这里我们采用第三种方案，按user\_id hash取模分库分表，并额外添加一张表T\_blog\_user,该表包含字段blog\_id、user\_id。如果需要按照blog\_Id查询博客时，我们可以先按照blog\_id在t\_blog\_user表查找该博客的user\_id,然后再用user\_id和blog\_id为条件，对user\_id hash取模查询，此时就不需要进行全表扫描查询了。

优化：我们可以对T\_blog\_user表数据进行缓存

#### 分库分表后引入的问题

* 分布式事务问题  
  如果我们做了垂直分库或者水平分库以后,就必然会涉及到跨库执行SQL的问题,这样就引发了互联网界的老大难问题-"分布式事务"。那要如何解决这个问题呢？  
  1.使用分布式事务中间件

2.使用MySQL自带的针对跨库的事务一致性方案(XA),不过性能要比单库的慢10倍左右。

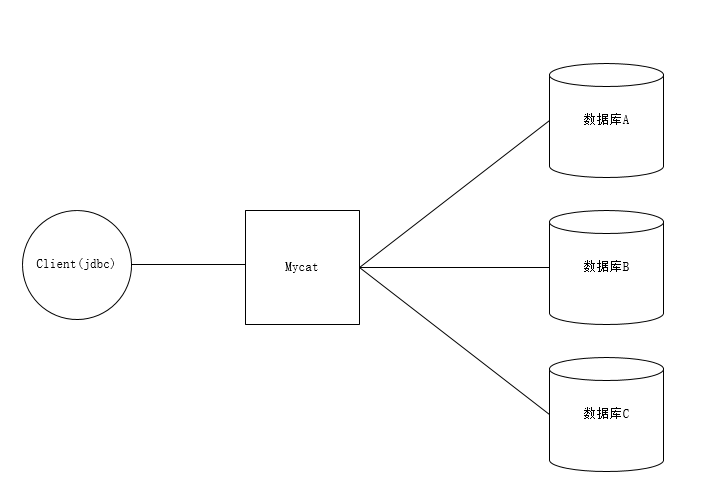
3.能否避免掉跨库操作(比如将用户和商品放在同一个库中)

* 跨库join的问题  
  分库分表后表之间的关联操作将受到限制，我们无法join位于不同分库的表，也无法join分表粒度不同的表， 结果原本一次查询能够完成的业务，可能需要多次查询才能完成。粗略的解决方法： 全局表：基础数据，所有库都拷贝一份。 字段冗余：这样有些字段就不用join去查询了。 系统层组装：分别查询出所有，然后组装起来，较复杂。
* 横向扩容的问题  
  当我们使用HASH取模做分表的时候,针对数据量的递增,可能需要动态的增加表,此时就需要考虑因为reHash导致数据迁移的问题。
* 结果集合并、排序的问题  
  因为我们是将数据分散存储到不同的库、表里的,当我们查询指定数据列表时,数据来源于不同的子库或者子表,就必然会引发结果集合并、排序的问题。如果每次查询都需要排序、合并等操作,性能肯定会受非常大的影响。走缓存可能一条路!

#### 使用分库分表中间件

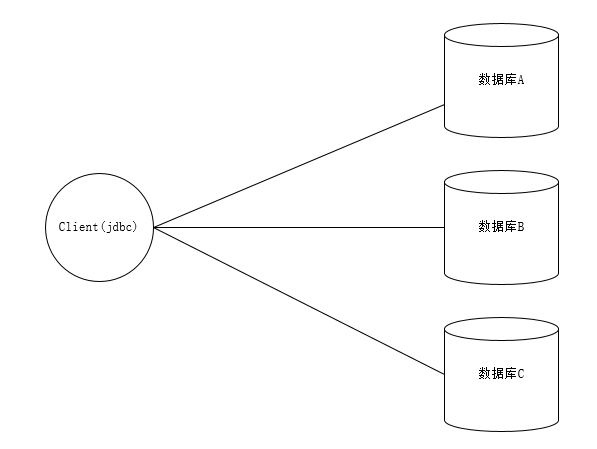
**Mycat**

Mycat 是一个开源的分布式数据库系统，是一个实现了 MySQL 协议的的Server，前端用户可以把它看作是一个数据库代理，用 MySQL 客户端工具和命令行访问，而其后端可以用MySQL 原生（Native）协议与多个 MySQL 服务器通信，也可以用 JDBC 协议与大多数主流数据库服务器通信，其核心功能是分表分库，即将一个大表水平分割为 N 个小表，存储在后端 MySQL 服务器里或者其他数据库里。



官方使用指南：<http://www.mycat.io/document/mycat-definitive-guide.pdf>

**Sharding-JDBC**Sharding-JDBC定位为轻量级java框架，使用客户端直连数据库，以jar包形式提供服务，无proxy代理层，无需额外部署，无其他依赖，DBA也无需改变原有的运维方式。



**Sharding-jdbc 使用案例参考博文**<https://blog.csdn.net/l1028386804/article/details/79368021>

此外还有淘宝的TDDL,支付宝的OneProxy,360的Atlas等。

## 扩容

**停机扩容（不推荐）**

这个方案就跟停机迁移一样，步骤几乎一致，唯一的一点就是那个导数的工具，是把现有库表的数据抽出来慢慢倒入到新的库和表里去。这有点不太靠谱，因为既然分库分表就说明数据量实在是太大了，可能多达几亿条，甚至几十亿，你这么玩儿，可能会出问题。

从单库单表迁移到分库分表的时候，数据量并不是很大，单表最大也就两三千万。那么你写个工具，多弄几台机器并行跑，1小时数据就导完了。这没有问题。

如果 3 个库 + 12 个表，跑了一段时间了，数据量都 1~2 亿了。光是导 2 亿数据，都要导个几个小时，6 点，刚刚导完数据，还要搞后续的修改配置，重启系统，测试验证，10 点才可以搞完。所以不能这么搞。

总结：数据需要重新在各表中迁移，停机时间长，系统在此期间无法对外提供服务。

**优化后的方案**

一开始上来就是 32 个库，每个库 32 个表，那么总共是 1024 张表。

我可以告诉各位同学，这个分法，第一，基本上国内的互联网肯定都是够用了，第二，无论是并发支撑还是数据量支撑都没问题。

每个库正常承载的写入并发量是 1000，那么 32 个库就可以承载32 \* 1000 = 32000 的写并发，如果每个库承载 1500 的写并发，32 \* 1500 = 48000 的写并发，接近 5 万每秒的写入并发。

有些除非是国内排名非常靠前的这些公司，他们的最核心的系统的数据库，可能会出现几百台数据库的这么一个规模，128个库，256个库，512个库。

1024 张表，假设每个表放 500 万数据，在 MySQL 里可以放 50 亿条数据。

每秒 5 万的写并发，总共 50 亿条数据，对于国内大部分的互联网公司来说，其实一般来说都够了。

谈分库分表的扩容，第一次分库分表，就一次性给他分个够，32 个库，1024 张表，可能对大部分的中小型互联网公司来说，已经可以支撑好几年了。

一个实践是利用 32 \* 32 来分库分表，即分为 32 个库，每个库里一个表分为 32 张表。一共就是 1024 张表。根据某个 id 先根据 32 取模路由到库，再根据 32 取模路由到库里的表。

|  |  |  |
| --- | --- | --- |
| orderId | id % 32 (库) | id / 32 % 32 (表) |
| 259 | 3 | 8 |
| 1189 | 5 | 5 |
| 352 | 0 | 11 |

刚开始的时候，这个库可能就是逻辑库，建在一个数据库上的，就是一个mysql服务器可能建了 n 个库，比如 32 个库。后面如果要拆分，就是不断在库和 mysql 服务器之间做迁移就可以了。然后系统配合改一下配置即可。

比如说最多可以扩展到32个数据库服务器，每个数据库服务器是一个库。如果还是不够？最多可以扩展到 1024 个数据库服务器，每个数据库服务器上面一个库一个表。因为最多是1024个表。

这么搞，是不用自己写代码做数据迁移的，都交给 dba 来搞好了，但是 dba 确实是需要做一些库表迁移的工作，但是总比你自己写代码，然后抽数据导数据来的效率高得多吧。

哪怕是要减少库的数量，也很简单，其实说白了就是按倍数缩容就可以了，然后修改一下路由规则。

这里对步骤做一个总结：

1. 设定好几台数据库服务器，每台服务器上几个库，每个库多少个表，推荐是 32库 \* 32表，对于大部分公司来说，可能几年都够了。
2. 路由的规则，orderId 模 32 = 库，orderId / 32 模 32 = 表
3. 扩容的时候，申请增加更多的数据库服务器，装好 mysql，呈倍数扩容，4 台服务器，扩到 8 台服务器，再到 16 台服务器。
4. 由 dba 负责将原先数据库服务器的库，迁移到新的数据库服务器上去，库迁移是有一些便捷的工具的。
5. 我们这边就是修改一下配置，调整迁移的库所在数据库服务器的地址。
6. 重新发布系统，上线，原先的路由规则变都不用变，直接可以基于 n 倍的数据库服务器的资源，继续进行线上系统的提供服务。

总结：预估系统可能达到的数据规模，设置数据库实例个数及每个数据库表的个数，一次分配足够的库表。这种扩容方式不需要程序员写程序把原来的表数据在各个表间做迁移，只需要把最初创建好的数据库实例迁移到其它的mysql服务器上。这种方式迁移减少了扩容的复杂性且速度更快。这里建议在扩容的时候按倍数新增mysql服务器，比如，原来4台扩展到8台，主要是为了平衡各个mysql服务器的读写请求，将请求均匀分布到各个服务器。