https://mp.weixin.qq.com/s/TeKNFprF9jCKjj-ItsFxog

**为什么要分库分表？**

## 1 垂直方向

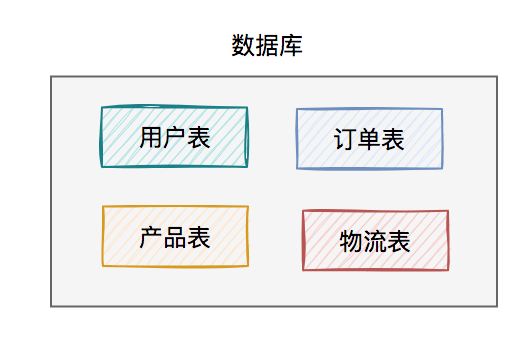
**垂直方向主要针对的是业务**，下面聊聊业务的发展跟分库分表有什么关系。

### 1.1 单库

在系统初期，业务功能相对来说比较简单，系统模块较少。

为了快速满足迭代需求，减少一些不必要的依赖。更重要的是减少系统的复杂度，保证开发速度，我们通常会使用单库来保存数据。

系统初期的数据库架构如下：



此时，使用的数据库方案是：一个数据库包含多张业务表。用户读数据请求和写数据请求，都是操作的同一个数据库。

### 1.2 分表

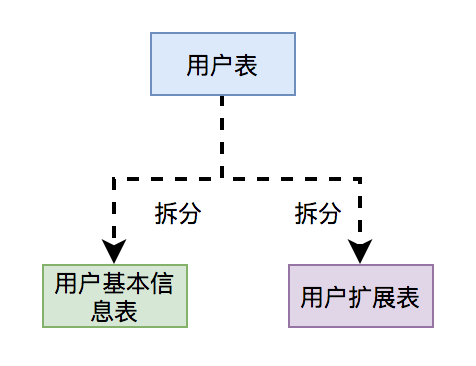
系统上线之后，随着业务的发展，不断的添加新功能。导致单表中的字段越来越多，开始变得有点不太好维护了。

一个用户表就包含了几十甚至上百个字段，管理起来有点混乱。

这时候该怎么办呢？

答：分表。

将用户表拆分为：用户基本信息表 和 用户扩展表。



用户基本信息表中存的是用户最主要的信息，比如：用户名、密码、别名、手机号、邮箱、年龄、性别等核心数据。

这些信息跟用户息息相关，查询的频次非常高。

而用户扩展表中存的是用户的扩展信息，比如：所属单位、户口所在地、所在城市等等，非核心数据。

这些信息只有在特定的业务场景才需要查询，而绝大数业务场景是不需要的。

所以**通过分表把核心数据和非核心数据分开，让表的结构更清晰，职责更单一，更便于维护。**

除了按实际业务分表之外，我们还有一个常用的分表原则是：把调用频次高的放在一张表，调用频次低的放在另一张表。

有个非常经典的例子就是：订单表和订单详情表。

### 1.3 分库

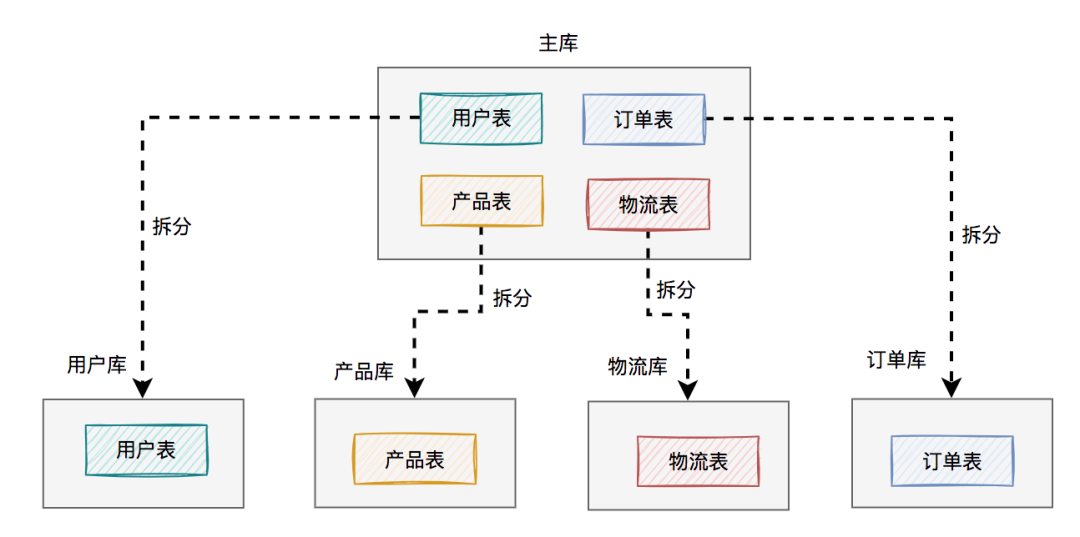
不知不觉，系统已经上线了一年多的时间了。经历了N个迭代的需求开发，功能已经非常完善。

系统功能完善，意味着系统各种关联关系，错综复杂。

此时，如果不赶快梳理业务逻辑，后面会带来很多隐藏问题，会把自己坑死。

这就需要**按业务功能，划分不同领域**了。把相同领域的表放到同一个数据库，不同领域的表，放在另外的数据库。

具体拆分过程如下：



将用户、产品、物流、订单相关的表，从原来一个数据库中，拆分成单独的用户库、产品库、物流库和订单库，一共四个数据库。

在这里为了看起来更直观，每个库我只画了一张表，实际场景可能有多张表。

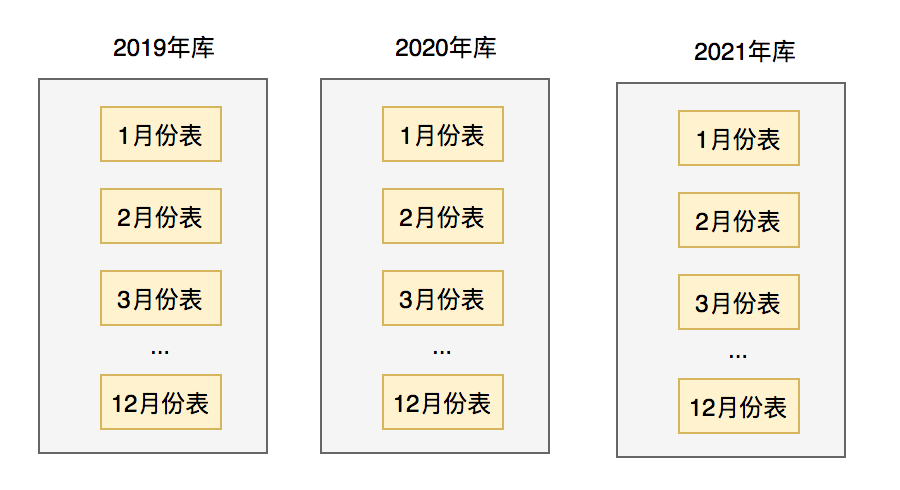
这样按领域拆分之后，每个领域只用关注自己相关的表，职责更单一了，一下子变得更好维护了。

### 1.4 分库分表

有时候按业务，只分库，或者只分表是不够的。比如：有些财务系统，需要按月份和年份汇总，所有用户的资金。

这就需要做：分库分表了。

每年都有个单独的数据库，每个数据库中，都有12张表，每张表存储一个月的用户资金数据。



这样分库分表之后，就能非常高效的查询出某个用户每个月，或者每年的资金了。

此外，还有些比较特殊的需求，比如需要按照地域分库，比如：华中、华北、华南等区，每个区都有一个单独的数据库。

甚至有些游戏平台，按接入的游戏厂商来做分库分表。

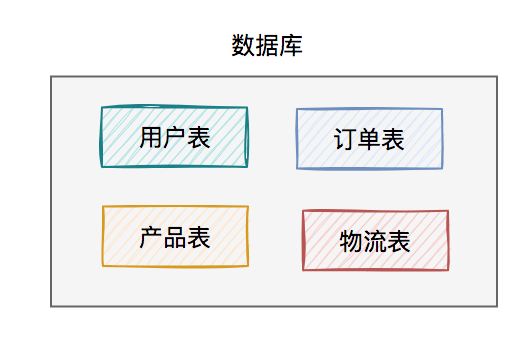
## 2 水平方向

水分方向主要针对的是数据，下面聊聊数据跟分库分表又有什么关系。

### 2.1 单库

在系统初期，由于用户非常少，所以系统并发量很小。并且存在表中的数据量也非常少。

这时的数据库架构如下：



此时，使用的数据库方案同样是：一个master数据库包含多张业务表。

用户读数据请求和写数据请求，都是操作的同一个数据库，该方案比较适合于并发量很低的业务场景。

### 2.2 主从读写分离

系统上线一段时间后，用户数量增加了。

此时，你会发现用户的请求当中，读数据的请求占据了大部分，真正写数据的请求占比很少。

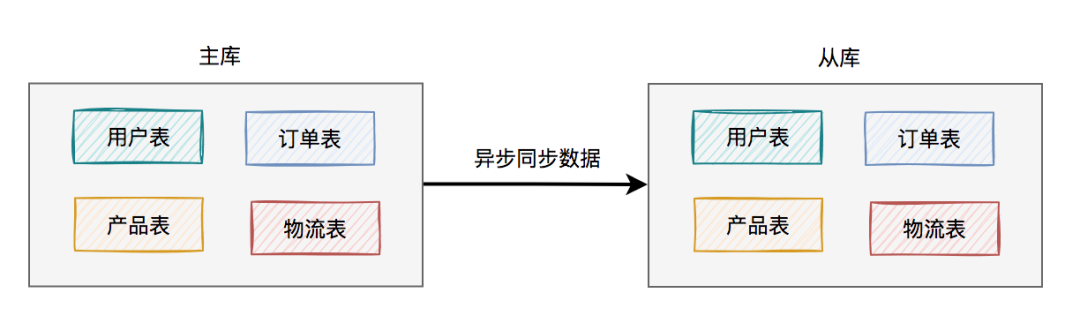
众所周知，数据库连接是有限的，它是非常宝贵的资源。而每次数据库的读或写请求，都需要占用至少一个数据库连接。

如果写数据请求需要的数据库连接，被读数据请求占用完了，不就写不了数据了？

这样问题就严重了。

为了解决该问题，我们需要把读库和写库分开。

于是，就出现了主从读写分离架构：



考虑刚开始用户量还没那么大，选择的是一主一从的架构，也就是常说的一个master一个slave。

所有的写数据请求，都指向主库。一旦主库写完数据之后，立马异步同步给从库。这样所有的读数据请求，就能及时从从库中获取到数据了（除非网络有延迟）。

读写分离方案可以解决上面提到的单节点问题，相对于单库的方案，能够更好的保证系统的稳定性。

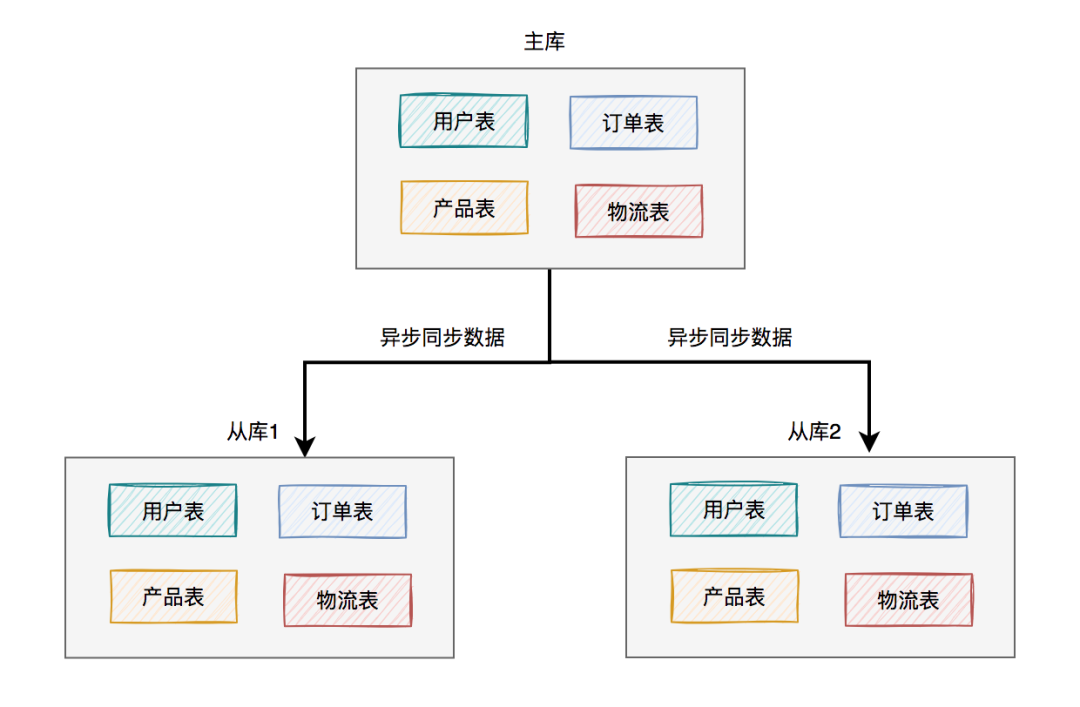
因为如果主库挂了，可以升级从库为主库，将所有读写请求都指向新主库，系统又能正常运行了。

读写分离方案其实也是分库的一种，它相对于为数据做了备份，它已经成为了系统初期的首先方案。

但这里有个问题就是：如果用户量确实有些大，如果master挂了，升级slave为master，将所有读写请求都指向新master。

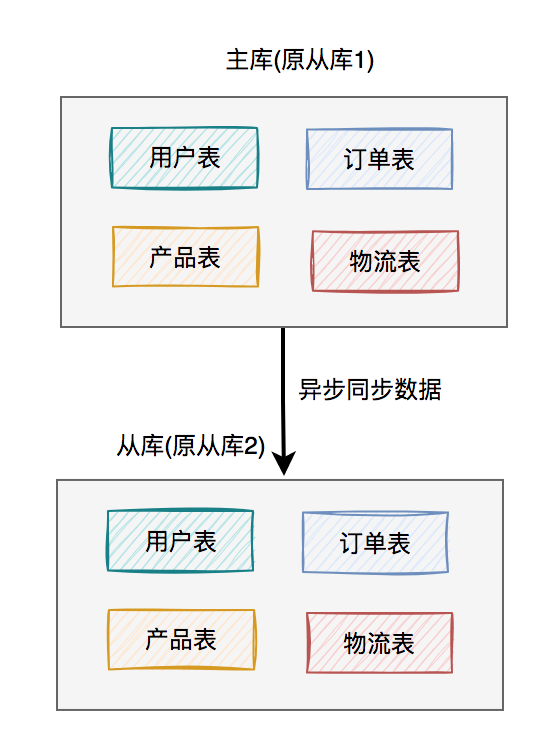
但此时，如果这个新master根本扛不住所有的读写请求，该怎么办？

这就需要一主多从的架构了：



上图中我列的是一主两从，如果master挂了，可以选择从库1或从库2中的一个，升级为新master。假如我们在这里升级从库1为新master，则原来的从库2就变成了新master的的slave了。

调整之后的架构图如下：



这样就能解决上面的问题了。

除此之外，如果查询请求量再增大，我们还可以将架构升级为一主三从、一主四从...一主N从等。

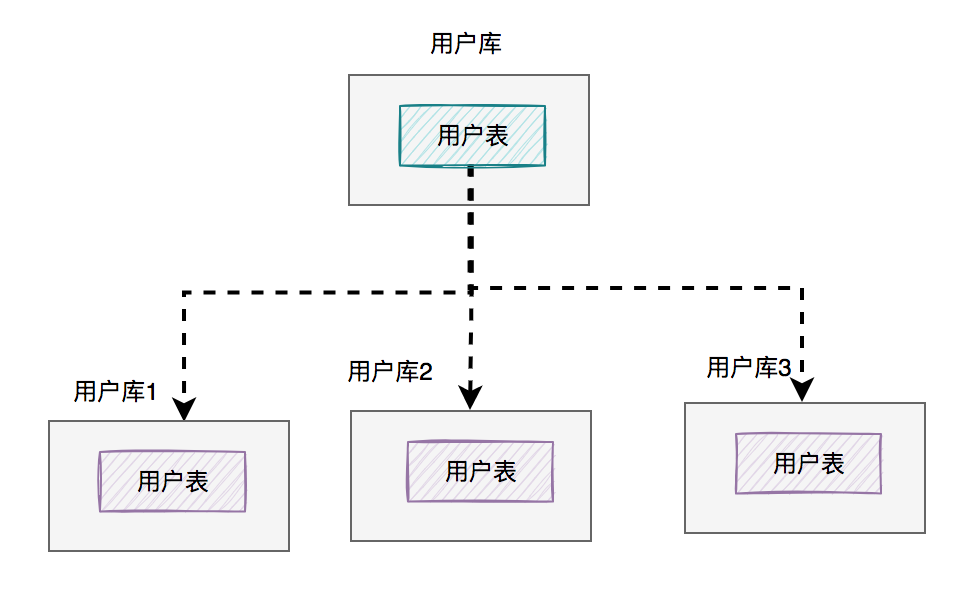
### 2.3 分库

上面的读写分离方案确实可以解决读请求大于写请求时，导致master节点扛不住的问题。但如果某个领域，比如：用户库。如果注册用户的请求量非常大，即写请求本身的请求量就很大，一个master库根本无法承受住这么大的压力。

这时该怎么办呢？

答：建立多个用户库。

用户库的拆分过程如下：



在这里我将用户库拆分成了三个库（真实场景不一定是这样的），每个库的表结构是一模一样的，只有存储的数据不一样。

### 2.4 分表

用户请求量上来了，带来的势必是数据量的成本上升。即使做了分库，但有可能单个库，比如：用户库，出现了5000万的数据。

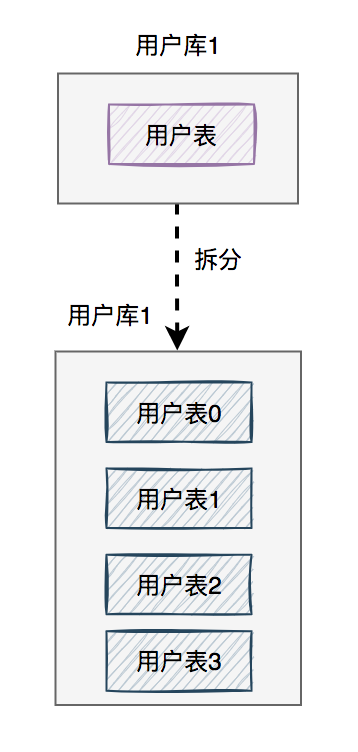
**根据经验值，单表的数据量应该尽量控制在1000万以内**，性能是最佳的。如果有几千万级的数据量，用单表来存，性能会变得很差。

如果数据量太大了，需要建立的索引也会很大，从小到大检索一次数据，会非常耗时，而且非常消耗cpu资源。

这时该怎么办呢？

答：分表，这样可以控制每张表的数据量，和索引大小。

表拆分过程如下：



我在这里将用户库中的用户表，拆分成了四张表（真实场景不一定是这样的），每张表的表结构是一模一样的，只是存储的数据不一样。

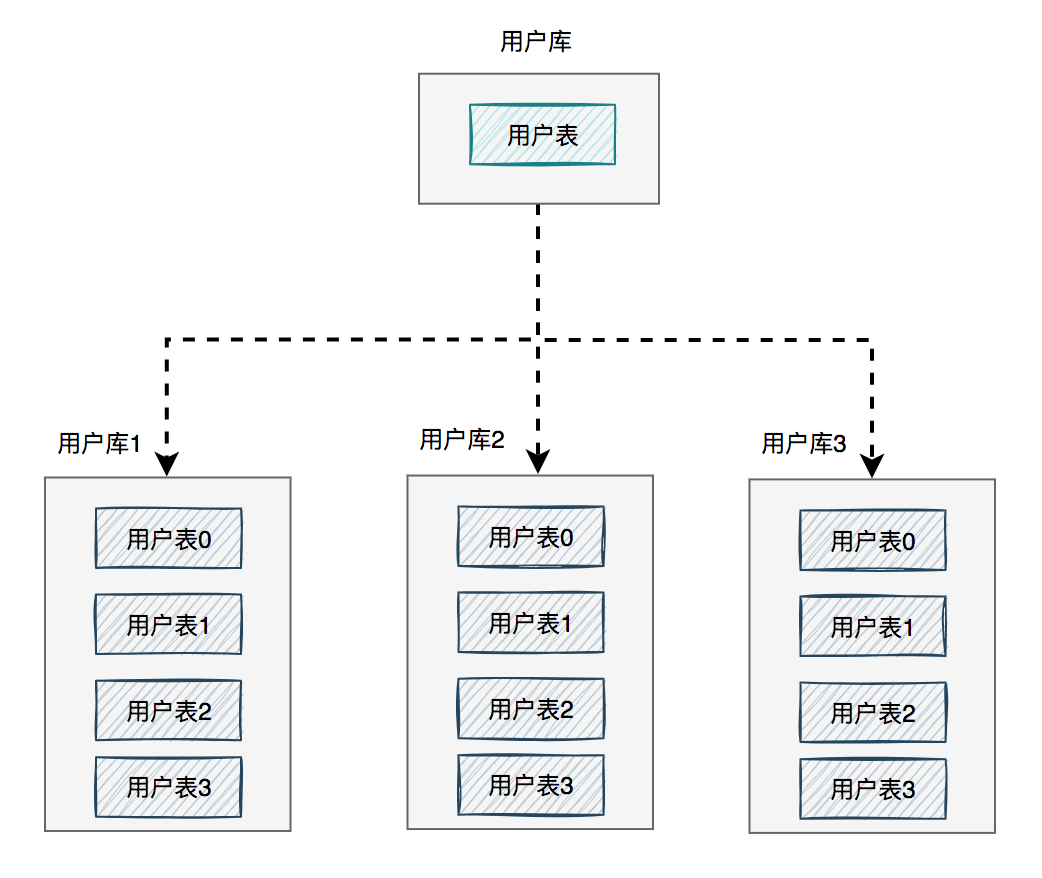
如果以后用户数据量越来越大，只需再多分几张用户表即可。

### 2.5 分库分表

当系统发展到一定的阶段，用户并发量大，而且需要存储的数据量也很多。这时该怎么办呢？

答：需要做分库分表。

如下图所示：



图中将用户库拆分成了三个库，每个库都包含了四张用户表。

如果有用户请求过来的时候，先根据用户id路由到其中一个用户库，然后再定位到某张表。

路由的算法挺多的：

* 根据id取模，比如：id=7，有4张表，则7%4=3，模为3，路由到用户表3。
* 给id指定一个区间范围，比如：id的值是0-10万，则数据存在用户表0，id的值是10-20万，则数据存在用户表1。
* 一致性hash算法