从帖子中心开始，聊“1对多”类业务数据库水平切分架构实践

本文将以“帖子中心”为例，介绍“1对多”类业务，随着数据量的逐步增大，数据库性能显著降低，数据库水平切分相关的架构实践：

- 如何来实施水平切分
- 水平切分后常见的问题
- 典型问题的优化思路及实践

一、什么是1对多关系

所谓的“1对1”“1对多”“多对多”，来自数据库设计中的“实体-关系”ER模型，用来描述实体时间的映射关系：

1对1

一个用户只有一个登录名，一个uid对应一个login_name，这是一个1对1的关系。

1对多

一个用户可以发多条微博，一条微博只有一个发送者；

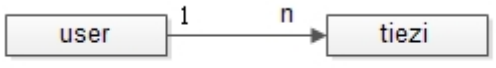
一个uid对应多个msg_id，一个msg_id只对应一个uid；

这是一个1对多的关系。

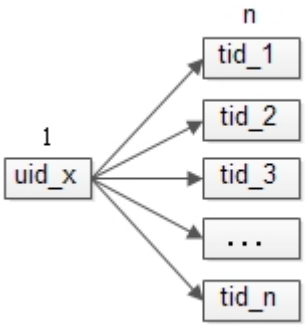
多对多

一个用户可以关注多个用户，一个用户也可以被多个粉丝关注，这是一个多对多的关系。

二、帖子中心业务分析



帖子中心是一个典型的1对多业务。



一个用户可以发布多个帖子，一个帖子只对应一个发布者。

任何脱离业务的架构设计都是耍流氓，先来看看帖子中心对应的业务需求。

帖子中心，是一个提供帖子发布，修改，删除，查看，搜索的服务。

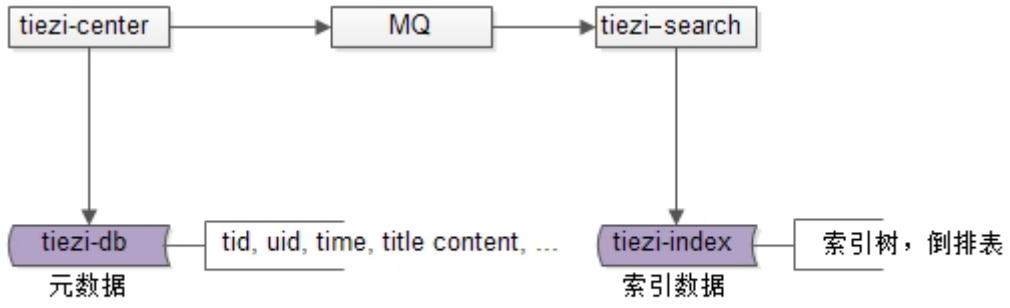
写操作需求

- 发布(insert)帖子
- 修改(update)帖子
- 删除(delete)帖子

读操作需求

- 通过tid查询帖子实体，单行查询。
- 通过uid查询用户发布过的帖子，列表查询。
- 帖子检索，例如通过时间、标题、内容搜索符合条件的帖子。

在数据量较大，并发量较大的时候，通常通过**元数据与索引数据分离**的架构来满足不同类型的需求：



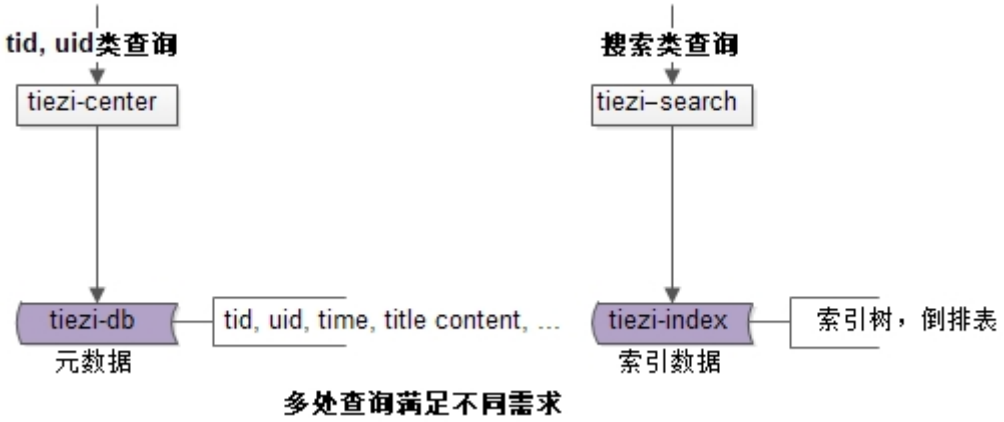
索引外置架构

架构中的几个关键点：

- tiezi-center服务。

- tiezi-db：提供元数据存储。
- tiezi-search搜索服务。
- tiezi-index：提供索引数据存储。
- MQ：tiezi-center与tiezi-search通讯媒介，一般不直接使用RPC调用，而是通过MQ对两个子系统解耦。

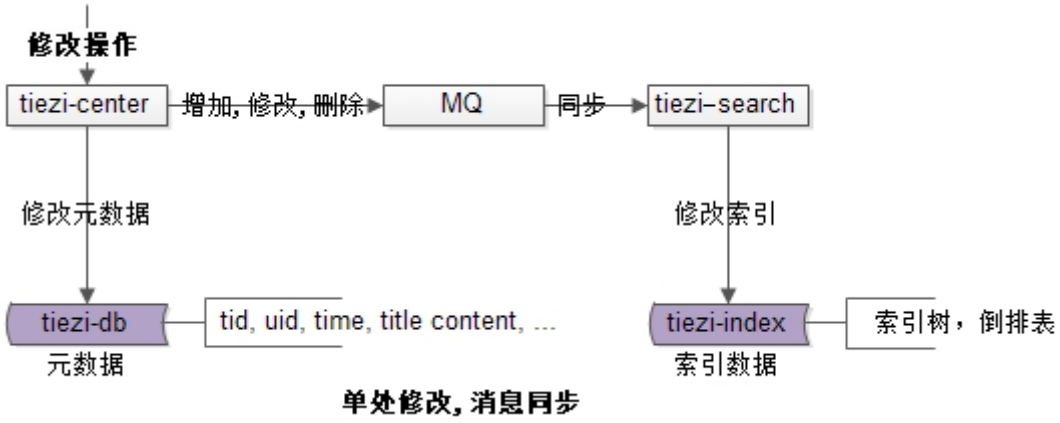
其中，tiezi-center和tiezi-search分别满足两类不同的读需求：



如上图所示：

- tid和uid上的查询需求，可以由tiezi-center从元数据读取并返回。
- 其他类检索需求，可以由tiezi-search从索引数据检索并返回。

对于写操作：



如上图所示：

- 增加，修改，删除的操作都会从tiezi-center发起。
- tiezi-center修改元数据。
- tiezi-center将信息修改通知发送给MQ。
- tiezi-search从MQ接受修改信息。

- tiezi-search修改索引数据。

tiezi-search可以使用Solr，ES等开源架构实现，数据量/并发量达到10亿/10万级别时可以自研搜索引擎，这一块不是本文的重点，后文将重点描述帖子中心元数据这一块的水平切分设计。

三、帖子中心元数据设计

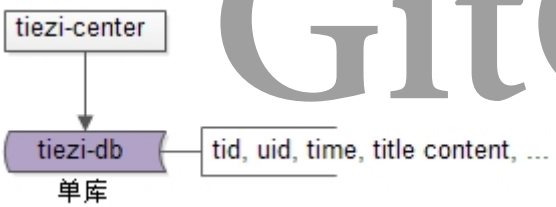
通过帖子中心业务分析，很容易了解到，其核心元数据为：

Tiezi(tid, uid, time, title, content, ...)

其中：

- tid为帖子ID，主键。
- uid为用户ID，发帖人。
- time, title, content ...等为帖子属性。

数据库设计上，在业务初期，单库就能满足元数据存储要求，其典型的架构设计为：



- tiezi-center：帖子中心服务，对调用者提供友好的RPC接口。
- tiezi-db：对帖子数据进行存储。

在相关字段上建立索引，就能满足相关业务需求：

- 帖子记录查询，**通过tid查询，约占读请求量的90%**

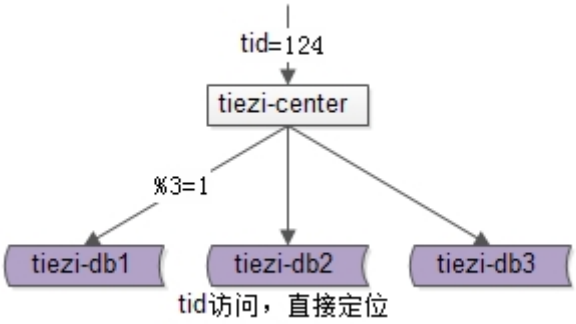
```
select * from t_tiezi where tid=$tid
```

- 帖子列表查询，**通过uid查询其发布的所有帖子，约占读请求量的10%**

```
select * from t_tiezi where uid=$uid
```

四、帖子中心水平切分-tid切分法

当数据量越来越大时，需要对帖子数据的存储进行线性扩展，既然是帖子中心，并且帖子记录查询量占了总请求的90%，很容易想到通过tid字段取模来进行水平切分：

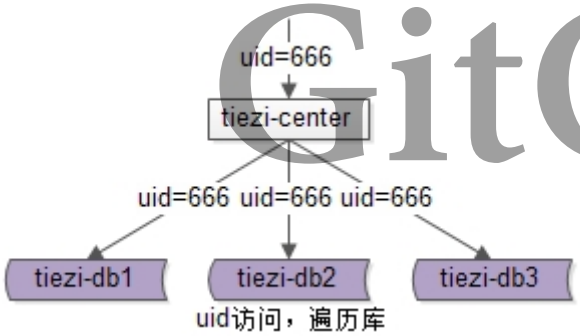


这个方法简单直接，优点是：

- 100%写请求可以直接定位到库。
- 90%的读请求可以直接定位到库。

缺点是：

- 一个用户发布的所有帖子可能会落到不同的库上，10%的请求通过uid来查询会比较麻烦。



如上图，一个uid访问需要遍历所有库。

五、帖子中心水平切分-uid切分法

有没有一种切分方法，确保同一个用户发布的所有帖子都落在同一个库上，而在查询一个用户发布的所有帖子时，不需要去遍历所有的库呢？

解答：使用uid来分库可以解决这个问题。

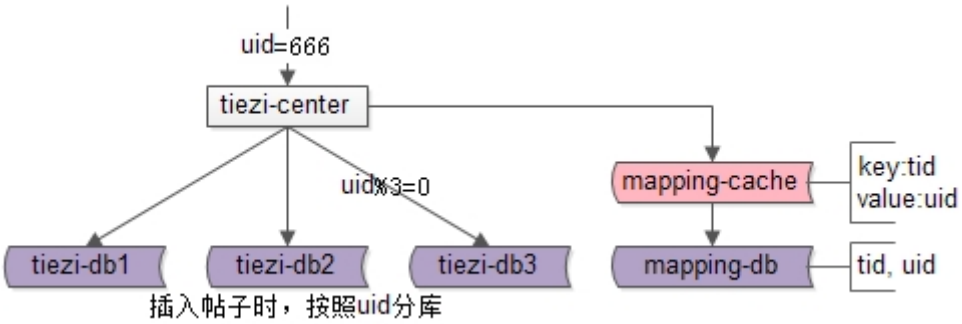
新出现的问题：如果使用uid来分库，确保了一个用户的帖子数据落在同一个库上，**那通过tid来查询，就不知道这个帖子落在哪个库上了**，岂不是还需要遍历全库，需要怎么优化呢？

解答：tid的查询是单行记录查询，只要在数据库（或者缓存）新增索引记录，新增tid到uid的映射关系，就能解决这个问题。

新增一个索引库：t_mapping(tid, uid)

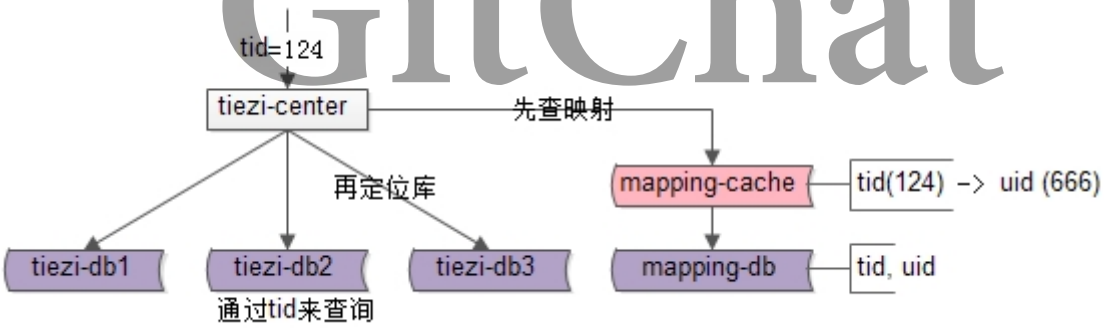
- 这个库只有两列，可以承载很多数据。
- 即使数据量过大，索引库可以利用tid水平切分。
- 这类kv形式的索引结构，可以很好的利用cache优化查询性能。
- 一旦帖子发布，tid和uid的映射关系就不会发生变化，cache的命中率会非常高。

使用uid分库，并增加索引库记录tid到uid的映射关系之后，每当有uid上的查询：



可以通过uid直接定位到库。

每当有tid上的查询：



- 先查询索引表，通过tid查询到对应的uid.
- 再通过uid定位到库。

这个方法的优点是：

- 一个用户发布的所以帖子落在同一个库上。
- 10%的请求过过uid来查询列表，可以直接定位到库。
- 索引表cache命中率非常高，因为tid与uid的映射关系不会变。

缺点是：

- 90%的tid请求，以及100%的修改请求，不能直接定位到库，需要先进行一次索引表的查询，当然这个查询非常非常块，通常在5ms内可以返回。

- 数据插入时需要查询元数据与索引表，可能引发潜在的一致性问题。

六、帖子中心水平切分-基因法

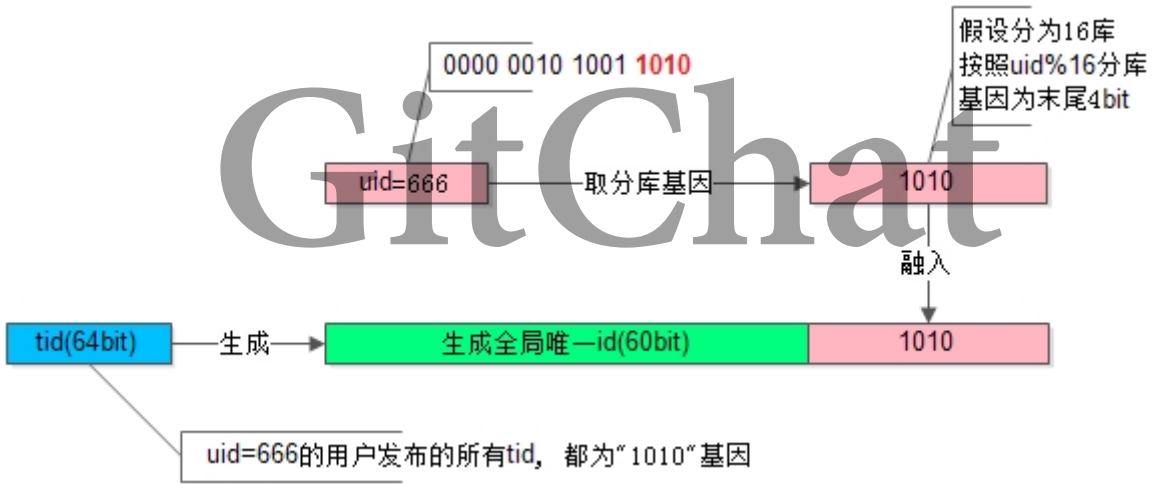
有没有一种方法，既能够通过uid定位到库，又不需要建立索引表来进行二次查询呢，这就是本文要叙述的“1对多”业务分库最佳实践，**基因法。**

什么是分库基因？

通过uid分库，假设分为16个库，采用uid%16的方式来进行数据库路由，这里的uid%16，其本质是**uid的最后4个bit决定这行数据落在哪个库上**，这4个bit，就是分库基因。

什么是基因法分库？

在“1对多”的业务场景，使用“1”分库，在“多”的数据id生成时，id末端加入分库基因，就能同时满足“1”和“多”的分库查询需求。



如上图所示，uid=666的用户发布了一条帖子：

- 使用uid%16分库，决定这行数据要插入到哪个库中。
- 分库基因是uid的最后4个bit，即1010。
- 在生成tid时，先使用一种分布式ID生成算法生成前60bit（上图中绿色部分）。
- 将分库基因加入到tid的最后4个bit（上图中粉色部分），拼装成最终的64bit帖子tid（上图中蓝色部分）。

通过这种方法保证，同一个用户发布的所有帖子的tid，都落在同一个库上，tid的最后4个bit都相同，于是：

- 通过uid%16能够定位到库。

- 通过tid%16也能定位到库。

完美！

潜在问题一：**同一个uid发布的tid落在同一个库上，会不会出现数据不均衡？**

回答：只要uid是均衡的，每个用户发布的平均帖子数是均衡的，每个库的数据就是均衡的。

潜在问题二：**最开始分16库，分库基因是4bit，未来要扩充成32库，分库基因变成了5bit，那怎么办？**

回答：需要提前做好容量预估，例如事先规划好5年内数据增长256库足够，就提前预留8bit基因。

七、总结

将以“帖子中心”为典型的“1对多”类业务，在架构上，采用元数据与索引数据分离的设计方法：

- 帖子服务，元数据满足uid和tid的查询需求。
- 搜索服务，索引数据满足复杂搜索寻求。

对于元数据的存储，在数据量较大的情况下，有三种常见的切分方法：

- tid切分法，按照tid分库，同一个用户发布的帖子落在不同的库上，通过uid来查询要遍历所有库。
- uid切分法，按照uid分库，同一个用户发布的帖子落在同一个库上，需要通过索引表或者缓存来记录tid与uid的映射关系，通过tid来查询时，先查到uid，再通过uid定位库。
- 基因法，按照uid分库，在生成tid里加入uid上的分库基因，保证通过uid和tid都能直接定位到库。

八、还有哪些未尽事宜

以好友关系为典型的**“多对多”类业务**，以订单中心为典型的**“多KEY”类业务**的水平拆分架构又应该怎么处理，敬请期待下期。