前言：数据库典型架构实践

本章，将介绍数据库架构设计中的一些基本概念，常见问题以及对应解决方案，为了便于读者理解，将以“用户中心”为例，讲解数据库架构设计的常见玩法。

用户中心

用户中心是一个非常常见的业务，主要提供用户注册、登录、信息查询与修改的服务，其核心元数据为：

User(uid, uname, passwd, sex, age, nickname, …)

其中：

- uid为用户ID，主键。
- uname, passwd, sex, age, nickname, …等为用户的属性。

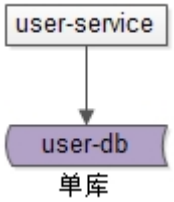
数据库设计上，一般来说在业务初期，单库单表就能够搞定这个需求。

图示说明

为了方便大家理解，后文图片说明较多：

- “灰色”方框，表示service，服务。
- “紫色”圆框，标识master，主库。
- “粉色”圆框，表示slave，从库。

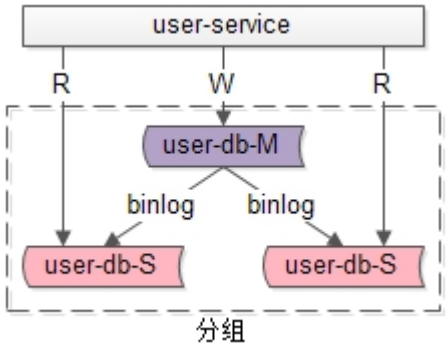
单库架构



最常见的架构设计如上：

- user-service：用户中心服务，对调用者提供友好的RPC接口。
- user-db：一个库进行数据存储。

分组架构



什么是分组？

答：分组架构是最常见的**一主多从，主从同步，读写分离**数据库架构：

- user-service：依旧是用户中心服务。
- user-db-M(master)：主库，提供数据库写服务。
- user-db-S(slave)：从库，提供数据库读服务。

主和从构成的数据库集群称为“组”。

分组有什么特点？

答：同一个组里的数据库集群：

- 主从之间通过binlog进行数据同步。
- 多个实例数据库结构完全相同。
- 多个实例存储的数据也完全相同，本质上是讲数据进行复制。

分组架构究竟解决什么问题？

答：大部分互联网业务读多写少，数据库的读往往最先成为性能瓶颈，如果希望：

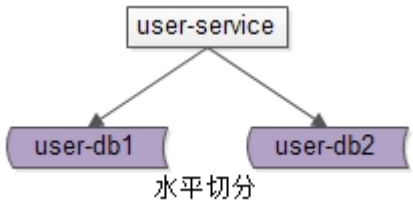
- 线性提升数据库读性能。
- 通过消除读写锁冲突提升数据库写性能。

- 通过冗余从库实现数据的“读高可用”。

此时可以使用分组架构，需要注意的是，分组架构中，数据库的主库依然是写单点。

一句话总结，**分组解决的是“数据库读写高并发量高”问题**，所实施的架构设计。

分片架构



什么是分片？

答：分片架构是大伙常说的水平切分(sharding)数据库架构：

- user-service：依旧是用户中心服务。
- user-db1：水平切分成2份中的第一份。
- user-db2：水平切分成2份中的第二份。

分片后，多个数据库实例也会构成一个数据库集群。

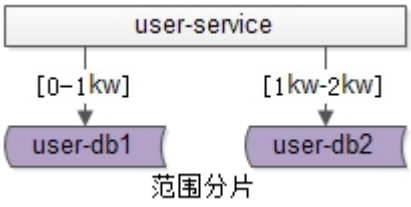
水平切分，到底是分库还是分表？

答：**强烈建议分库**，而不是分表来实施水平切分，因为：

- 分表依然公用一个数据库文件，仍然有磁盘IO的竞争。
- 分库能够很容易的将数据迁移到不同数据库实例，甚至数据库机器上，扩展性更好。

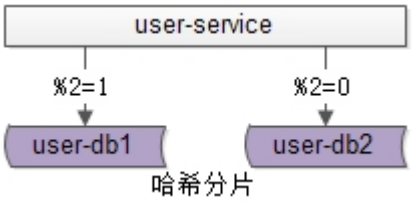
水平切分，用什么算法？

答：常见的水平切分算法有**“范围法”和“哈希法”**：



范围法如上图：以用户中心的业务主键uid为划分依据，将数据水平切分到两个数据库实例上去：

- user-db1：存储0到1千万的uid数据。
- user-db2：存储0到2千万的uid数据。



哈希法如上图：也是以用户中心的业务主键uid为划分依据，将数据水平切分到两个数据库实例上去：

- user-db1：存储uid取模得1的uid数据。
- user-db2：存储uid取模得0的uid数据。

这两种方法在互联网都有使用，其中哈希法使用较为广泛。

分片有什么特点？

答：同一个分片里的数据库集群：

- 多个实例之间本身不直接产生联系，不像主从间有binlog同步。
- 多个实例数据库结构，也完全相同。
- 多个实例存储的数据之间没有交集，所有实例间数据并集构成全局数据。

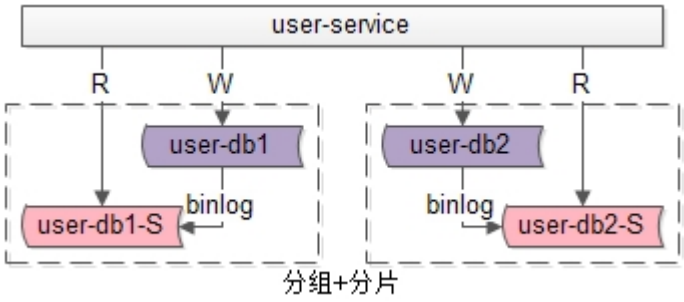
分片架构究竟解决什么问题？

答：大部分互联网业务数据量很大，单库容量容易成为瓶颈，此时通过分片可以：

- 线性提升数据库写性能，需要注意的是，分组架构是不能线性提升数据库写性能的。
- 降低单库数据容量。

一句话总结，**分片解决的是“数据库数据量大”问题**，所实施的架构设计。

分组+分片架构

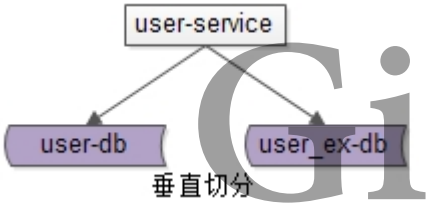


如果业务读写并发量很高，数据量也很大，通常需要实施分组+分片的数据库架构：

- 通过分片来降低单库的数据量，线性提升数据库的写性能。
- 通过分组来线性提升数据库的读性能，保证读库的高可用。

垂直切分

除了水平切分，垂直切分也是一类常见的数据库架构设计，垂直切分一般和业务结合比较紧密。



还是以用户中心为例，可以这么进行垂直切分：

User(uid, uname, passwd, sex, age, …)

User_EX(uid, intro, sign, …)

- 垂直切分开的表，主键都是uid。
- 登录名，密码，性别，年龄等属性放在一个垂直表（库）里。
- 自我介绍，个人签名等属性放在另一个垂直表（库）里。

如何进行垂直切分？

答：根据业务对数据进行垂直切分时，一般要考虑属性的“长度”和“访问频度”两个因素：

- 长度较短，访问频度较高的放在一起。
- 长度较长，访问频度较低的放在一起。

这是因为，数据库会以行(row)为单位，将数load到内存(buffer)里，在内存容量有限的情况下，长度短且访问频度高的属性，内存能够load更多的数据，命中率会更高，磁盘IO会减少，数据库的性能会提升。

垂直切分有什么特点？

答：垂直切分和水平切有相似的地方，又不太相同：

- 多个实例之间也不直接产生联系，即没有binlog同步。
- 多个实例数据库结构，都不一样。
- 多个实例存储的数据之间至少有一列交集，一般来说是业务主键，所有实例间数据并集构成全局数据。

垂直切分解决什么问题？

答：垂直切分即可以降低单库的数据量，还可以降低磁盘IO从而提升吞吐量，但它与业务结合比较紧密，并不是所有业务都能够进行垂直切分的。

总结

业务场景决定数据库架构：

- 业务初期用单库。
- 读压力大，读高可用，用分组。
- 数据量大，写线性扩容，用分片。
- 属性短，访问频度高的属性，垂直拆分到一起。

还有哪些未尽事宜？

本文以“用户中心”为例，对常见数据库架构设计进行了简要梳理与总结，但实际数据库架构设计远比此复杂，特别是水平切分的架构设计，不同业务场景的切分方式不尽相同。

本专题后续将要详细介绍，覆盖90%互联网业务特性的四类业务：

- 用户中心（“单KEY”类业务）
- 帖子中心（“1对多”类业务）
- 好友关系（“多对多”类业务）
- 订单中心（“多KEY”类业务）

分别应该如何实施水平切分，专题期望达到的效果是，从今之后，不管什么业务场景，数据量大水平切分的技术问题从此就不用再担心了，尽请期待。

有任何疑问，欢迎评论提问，后续第2到第5场的Chat会逐一解答。

希望对得起这1块钱，帮忙转发一下哟。

Chat专题预订通道：

沈剑教你一次搞定数据库水平切分