# MySQL 分库分表及其平滑扩容方案

2018年07月26日 07:38:52 [kefeng-wang](https://me.csdn.net/kefengwang" \t "https://blog.csdn.net/kefengwang/article/details/_blank) 阅读数：1793更多

所属专栏： [Java存储相关中间件](https://blog.csdn.net/column/details/25370.html" \t "https://blog.csdn.net/kefengwang/article/details/_blank)

 版权声明：【自由转载-非商用-非衍生-保持署名】－转载请标明作者和出处。 https://blog.csdn.net/kefengwang/article/details/81213050

[众所周知](https://www.baidu.com/s?wd=%E4%BC%97%E6%89%80%E5%91%A8%E7%9F%A5&tn=24004469_oem_dg&rsv_dl=gh_pl_sl_csd" \t "https://blog.csdn.net/kefengwang/article/details/_blank)，数据库很容易成为应用系统的瓶颈。单机数据库的资源和处理能力有限，在高并发的分布式系统中，可采用分库分表突破单机局限。本文总结了分库分表的相关概念、全局ID的生成策略、分片策略、平滑扩容方案、以及流行的方案。   
  
****作者：****王克锋   
****出处：****[https://kefeng.wang/2018/07/22/mysql-sharding/](https://kefeng.wang/2018/07/22/mysql-sharding/" \t "https://blog.csdn.net/kefengwang/article/details/_blank)   
****版权：****[自由转载-非商用-非衍生-保持署名](https://creativecommons.org/licenses/by-nc-nd/3.0/deed.zh" \t "https://blog.csdn.net/kefengwang/article/details/_blank)，转载请标明作者和出处。

## **1 分库分表概述**

在业务量不大时，单库单表即可支撑。   
当数据量过大存储不下、或者并发量过大负荷不起时，就要考虑分库分表。

### **1.1 分库分表相关术语**

* 读写分离: 不同的数据库，同步相同的数据，分别只负责数据的读和写；
* 分区: 指定分区列表达式，把记录拆分到不同的区域中(必须是同一[服务器](https://www.baidu.com/s?wd=%E6%9C%8D%E5%8A%A1%E5%99%A8&tn=24004469_oem_dg&rsv_dl=gh_pl_sl_csd" \t "https://blog.csdn.net/kefengwang/article/details/_blank)，可以是不同硬盘)，应用看来还是同一张表，没有变化；
* 分库：一个系统的多张数据表，存储到多个数据库实例中；
* 分表: 对于一张多行(记录)多列(字段)的二维数据表，又分两种情形：   
  (1) 垂直分表: 竖向切分，不同分表存储不同的字段，可以把不常用或者大容量、或者不同业务的字段拆分出去；   
  (2) 水平分表(最复杂): 横向切分，按照特定分片算法，不同分表存储不同的记录。

### **1.2 真的要采用分库分表？**

需要注意的是，分库分表会为数据库维护和业务逻辑带来一系列复杂性和性能损耗，除非预估的业务量大到[万不得已](https://www.baidu.com/s?wd=%E4%B8%87%E4%B8%8D%E5%BE%97%E5%B7%B2&tn=24004469_oem_dg&rsv_dl=gh_pl_sl_csd" \t "https://blog.csdn.net/kefengwang/article/details/_blank)，切莫过度设计、过早优化。   
规划期内的数据量和性能问题，尝试能否用下列方式解决：

* 当前数据量：如果没有达到几百万，通常无需分库分表；
* 数据量问题：增加磁盘、增加分库(不同的业务功能表，整表拆分至不同的数据库)；
* 性能问题：升级CPU/内存、读写分离、优化数据库系统配置、优化数据表/索引、优化 SQL、分区、数据表的垂直切分；
* 如果仍未能奏效，才考虑最复杂的方案：数据表的水平切分。

## **2 全局ID生成策略**

### **2.1 自动增长列**

优点：数据库自带功能，有序，性能佳。   
缺点：单库单表无妨，分库分表时如果没有规划，ID可能重复。解决方案：

#### **2.1.1 设置自增偏移和步长**

## 假设总共有 10 个分表

## 级别可选: SESSION(会话级), GLOBAL(全局)SET @@SESSION.auto\_increment\_offset = 1; ## 起始值, 分别取值为 1~10SET @@SESSION.auto\_increment\_increment = 10; ## 步长增量

* 1
* 2
* 3
* 4

如果采用该方案，在扩容时需要迁移已有数据至新的所属分片。

#### **2.1.2 全局ID映射表**

在全局 Redis 中为每张数据表创建一个 ID 的键，记录该表当前最大 ID；   
每次申请 ID 时，都自增 1 并返回给应用；   
Redis 要定期持久至全局数据库。

### **2.2 UUID(128位)**

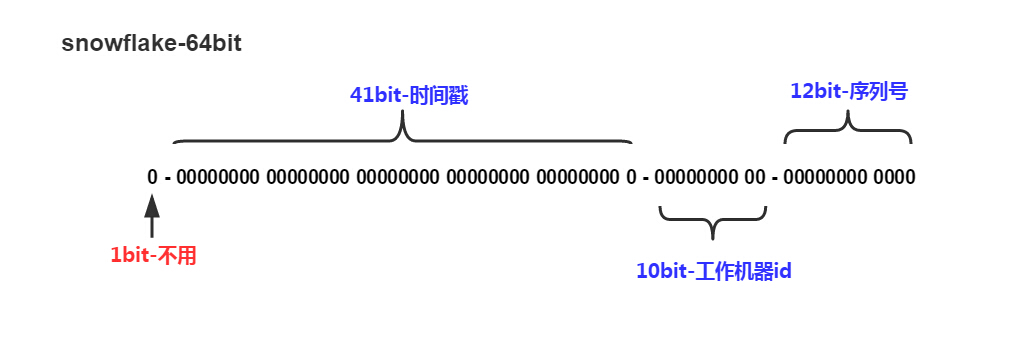
在一台机器上生成的数字，它保证对在同一时空中的所有机器都是唯一的。通常平台会提供生成UUID的API。   
UUID 由4个连字号(-)将32个字节长的字符串分隔后生成的字符串，总共36个字节长。形如：550e8400-e29b-41d4-a716-446655440000。   
UUID 的计算因子包括：以太网卡地址、纳秒级时间、芯片ID码和许多可能的数字。   
UUID 是个标准，其实现有几种，最常用的是微软的 GUID(Globals Unique Identifiers)。

优点：简单，全球唯一；   
缺点：存储和传输空间大，无序，性能欠佳。

### **2.3 COMB(组合)**

参考资料：[The Cost of GUIDs as Primary Keys](http://www.informit.com/articles/article.aspx?p=25862" \t "https://blog.csdn.net/kefengwang/article/details/_blank)   
组合 GUID(10字节) 和时间(6字节)，达到有序的效果，提高索引性能。

### **2.4 Snowflake(雪花) 算法**

参考资料：[twitter/snowflake](https://github.com/twitter/snowflake/tree/snowflake-2010" \t "https://blog.csdn.net/kefengwang/article/details/_blank)，[Snowflake 算法详解](https://www.lanindex.com/twitter-snowflake%EF%BC%8C64%E4%BD%8D%E8%87%AA%E5%A2%9Eid%E7%AE%97%E6%B3%95%E8%AF%A6%E8%A7%A3/" \t "https://blog.csdn.net/kefengwang/article/details/_blank)   
Snowflake 是 Twitter 开源的分布式 ID 生成算法，其结果为 long(64bit) 的数值。   
其特性是各节点无需协调、按时间大致有序、且整个集群各节点单不重复。   
该数值的默认组成如下(符号位之外的三部分允许个性化调整)：   


* 1bit: 符号位，总是 0(为了保证数值是正数)。
* 41bit: 毫秒数(可用 69 年)；
* 10bit: 节点ID(5bit数据中心 + 5bit节点ID，支持 32 \* 32 = 1024 个节点)
* 12bit: 流水号(每个节点每毫秒内支持 4096 个 ID，相当于 409万的 QPS，相同时间内如 ID 遇翻转，则等待至下一毫秒)

## **3 分片策略**

### **3.1 连续分片**

根据特定字段(比如用户ID、订单时间)的范围，值在该区间的，划分到特定节点。   
优点：集群扩容后，指定新的范围落在新节点即可，无需进行数据迁移。   
缺点：如果按时间划分，数据热点分布不均(历史数冷当前数据热)，导致节点负荷不均。

### **3.3 ID取模分片**

缺点：扩容后需要迁移数据。

### **3.2 一致性Hash算法**

优点：扩容后无需迁移数据。

### **3.4 Snowflake 分片**

优点：扩容后无需迁移数据。

## **4 分库分表引入的问题**

### **4.1 分布式事务**

参见 [分布式事务的解决方案](https://kefeng.wang/2018/03/01/distributed-transaction/" \t "https://blog.csdn.net/kefengwang/article/details/_blank)   
由于两阶段/三阶段提交对性能损耗大，可改用事务补偿机制。

### **4.2 跨节点 JOIN**

对于单库 JOIN，MySQL 原生就支持；   
对于多库，出于性能考虑，不建议使用 MySQL 自带的 JOIN，可以用以下方案避免跨节点 JOIN：

* 全局表: 一些稳定的共用数据表，在各个数据库中都保存一份；
* 字段冗余: 一些常用的共用字段，在各个数据表中都保存一份；
* 应用组装：应用获取数据后再组装。

另外，某个 ID 的用户信息在哪个节点，他的关联数据(比如订单)也在哪个节点，可以避免分布式查询。

### **4.3 跨节点聚合**

只能在应用程序端完成。   
但对于分页查询，每次大量聚合后再分页，性能欠佳。

### **4.4 节点扩容**

节点扩容后，新的分片规则导致数据所属分片有变，因而需要迁移数据。

## **5 节点扩容方案**

相关资料: [数据库秒级平滑扩容架构方案](http://zhuanlan.51cto.com/art/201702/530555.htm" \t "https://blog.csdn.net/kefengwang/article/details/_blank)

### **5.1 常规方案**

如果增加的节点数和扩容操作没有规划，那么绝大部分数据所属的分片都有变化，需要在分片间迁移：

* 预估迁移耗时，发布停服公告；
* 停服(用户无法使用服务)，使用事先准备的迁移脚本，进行数据迁移；
* 修改为新的分片规则；
* 启动服务器。

### **5.2 免迁移扩容**

采用双倍扩容策略，避免数据迁移。扩容前每个节点的数据，有一半要迁移至一个新增节点中，对应关系比较简单。   
具体操作如下(假设已有 2 个节点 A/B，要双倍扩容至 A/A2/B/B2 这 4 个节点)：

* 无需停止应用服务器；
* 新增两个数据库 A2/B2 作为从库，设置主从同步关系为：A=>A2、B=>B2，直至主从数据同步完毕(早期数据可手工同步)；
* 调整分片规则并使之生效：   
  原 ID%2=0 => A 改为 ID%4=0 => A, ID%4=2 => A2；   
  原 ID%2=1 => B 改为 ID%4=1 => B, ID%4=3 => B2。
* 解除数据库实例的主从同步关系，并使之生效；
* 此时，四个节点的数据都已完整，只是有冗余(多存了和自己配对的节点的那部分数据)，择机清除即可(过后随时进行，不影响业务)。

## **6 分库分表方案**

### **6.1 代理层方式**

部署一台代理服务器伪装成 MySQL 服务器，代理服务器负责与真实 MySQL 节点的对接，应用程序只和代理服务器对接。对应用程序是透明的。   
比如 MyCAT，[官网](http://www.mycat.io/" \t "https://blog.csdn.net/kefengwang/article/details/_blank)，[源码](https://github.com/MyCATApache/Mycat-Server" \t "https://blog.csdn.net/kefengwang/article/details/_blank)，参考文档：[MyCAT+MySQL 读写分离部署](https://kefeng.wang/2016/12/22/mycat-mysql/" \t "https://blog.csdn.net/kefengwang/article/details/_blank)   
MyCAT 后端可以支持 MySQL, SQL Server, Oracle, DB2, PostgreSQL等主流数据库，也支持MongoDB这种新型NoSQL方式的存储，未来还会支持更多类型的存储。   
MyCAT 不仅仅可以用作读写分离，以及分表分库、容灾管理，而且可以用于多租户应用开发、云平台基础设施，让你的架构具备很强的适应性和灵活性。

### **6.2 应用层方式**

处于业务层和 JDBC 层中间，是以 JAR 包方式提供给应用调用，对代码有侵入性。主要方案有：   
(1)淘宝网的 [TDDL](https://github.com/alibaba/tb_tddl" \t "https://blog.csdn.net/kefengwang/article/details/_blank): 已于 2012 年关闭了维护通道，建议不要使用。   
(2)当当网的 [Sharding-JDBC](https://github.com/sharding-sphere/sharding-sphere" \t "https://blog.csdn.net/kefengwang/article/details/_blank): 仍在活跃维护中：   
是当当应用框架 ddframe 中，从关系型数据库模块 dd-rdb 中分离出来的数据库水平分片框架，实现透明化数据库分库分表访问，实现了 Snowflake 分片算法；   
Sharding-JDBC定位为轻量Java框架，使用客户端直连数据库，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分片策略灵活，可支持等号、between、in等多维度分片，也可支持多分片键。   
SQL解析功能完善，支持聚合、分组、排序、limit、or等查询，并支持Binding Table以及笛卡尔积表查询。

Sharding-JDBC直接封装JDBC API，可以理解为增强版的JDBC驱动，旧代码迁移成本几乎为零：

* 可适用于任何基于Java的ORM框架，如JPA、Hibernate、Mybatis、Spring JDBC Template或直接使用JDBC。
* 可基于任何第三方的数据库连接池，如DBCP、C3P0、 BoneCP、Druid等。
* 理论上可支持任意实现JDBC规范的数据库。虽然目前仅支持MySQL，但已有支持Oracle、SQLServer等数据库的计划。