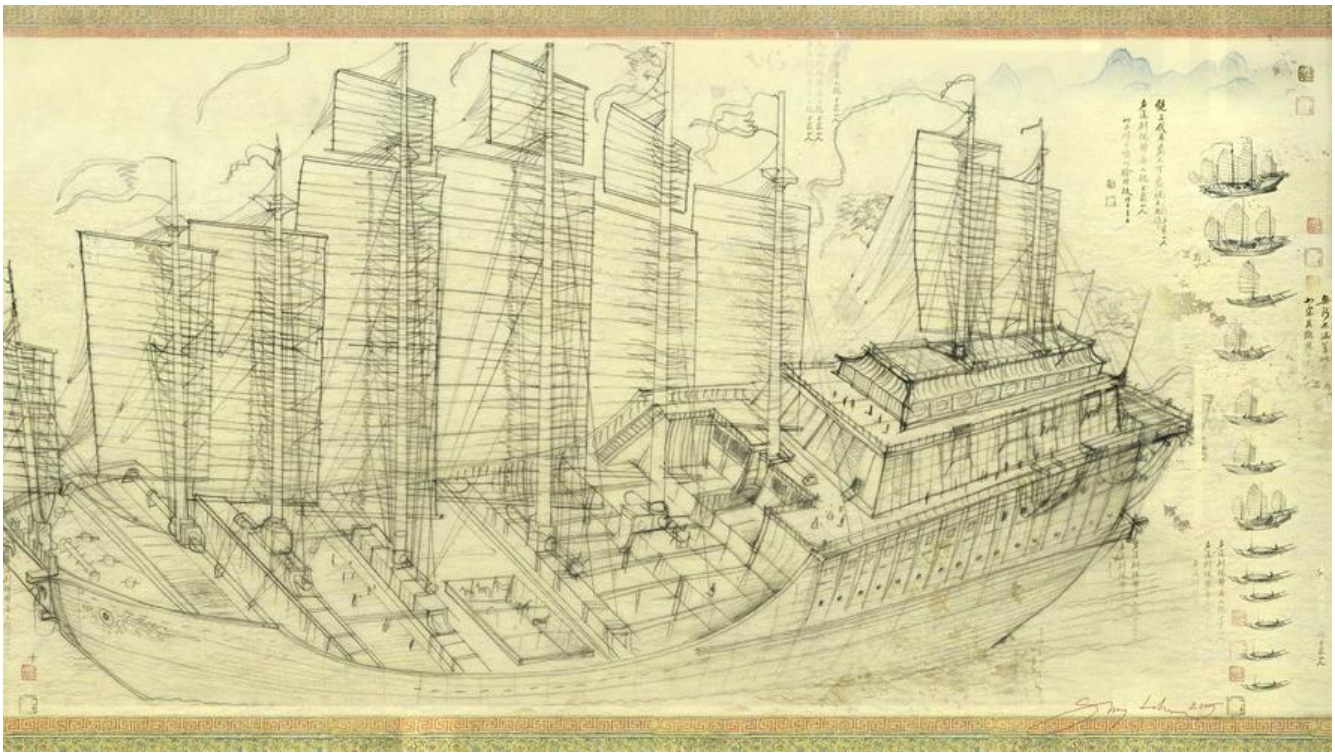
链滴

存储（七）—— Replication

作者：xinzhongtianxia
原文链接：https://ld246.com/article/1564019719401
来源网站：链滴
许可协议：署名-相同方式共享 4.0 国际 (CC BY-SA 4.0)



The major difference between a thing that might go wrong and that cannot possibly go wron is that when a thing that cannot possibly go wrong goes wrong it usually turns out to be im ossible to get at or repair.

—— Douglas Adams, Mostly Harmless (1992)

Replication

Replication是指把数据复制一份存储到其他机器上，这些机器之间通过网络进行连接。

在分布式系统中，你至少有三个理由使用Replication，并享受因此带来的好处和忍受因此带来的问题。

好处

1. 提高系统可用性

当某台机器故障后，因为相同的数据还存储在其他机器上，仍然可以被访问到，所以整个系统对外来仍然是可用的。

2. 提高系统性能

想象这样一个场景，我们只在北京有一个机房，于是我们发现，对于北方的用户我们的服务响应很快但是对于南方的用户请求，我们的服务似乎反应没有那么块了。**毕竟，在毫秒级的响应速度和上千公的传输距离面前，光速也并不是那么快了。**

此时，如果我们有北京和上海两个机房，通过Relicaiton保证两个机房中的数据相同，那么我们就可通过路由策略使北方的用户访问北京机房中的数据，南方的用户访问上海机房的数据。这样一来，全的用户的请求都可以快速响应了。

3. 提高系统吞吐量

我们知道每台机器的处理能力是有限的。如果一台机器只能承受1万个用户同时访问，但是现实中每钟同时访问我们请求的用户量是10万，怎么办？—— 把数据复制10份存储到10台机器上，10台机器时对外提供服务就行了（10这个数字在实际场景中并不严谨，这里意会即可）。

问题

世上的事，但凡得到好处时，肯定要付出些什么，Replication也不例外。

Replication遇到的问题，也许比我们想象中还要严重一些，永远记住一点，软件设计越简单越好，能足需求时，Replicaiton能不用就不用。

1. 存储成本

Replication的存储成本和数据被复制的份数成正比，现在存储还是很贵的，尤其是SSD和内存。想想果有几T（这个时代，数据轻轻松松就能上T）数据，在阿里云上存一年得多少钱，如果再把这些数据制三份呢？真是恐怖。有时候更让人心疼的是，有时候昂贵的存储仅仅是为了备份，为了预防那0.00%的机器挂掉的概率。是对于中小企业和个人开发者来说，可用性和成本之间的权衡，是一个永恒的题。

2. 数据传输延迟

存储成本的问题，用钱就能解决。但是这个数据传输延迟的问题，已经涉及到宇宙基本定律了，远远是钱能搞定的。如前面suo'shuo

Replicaiton是的数据传输延迟问题，是分布式领域最最头疼的问题之一，解决办法也是很多种，效果很难完美。

举个例子：

好比买火车票吧，假设12306的数据中心分别在北京和上海，sky同学在东北苏家屯，moon同学在香铜锣湾，sky看到余票还有2张，这时sky买了一张，再看余票时是1张，但是同一时刻，moon看到的票仍然是2张，于是moon胸有成竹的跟担心买不到票的女朋友说：“放心，还有两张票，一切尽在握之中！”，结果悲催的是他只买到了一张。。。于是乎，女友diss了他一晚上手速太慢，手速太慢。。

3. 数据一致性

在replication的过程中，几乎一定会有数据的丢失，很多场景下，都需要对replication的上下游进行据核对，然后通过补偿机制进行补救。而数据不一致期间，产生的问题也是很常见的，和上面提到的据延迟问题差不多。

除了数据丢失引起的数据不一致之外，如果采用多个leader同时写的方案（Mysql的主-主，Dynamo B，异地多活多个数据中心之间的数据同步等），也会因为数据冲突的问题，造成数据不一致问题。

问题的解决

1. 存储成本

如果不是必要，可以不使用replication，或者数据复制时，少保留几个副本。

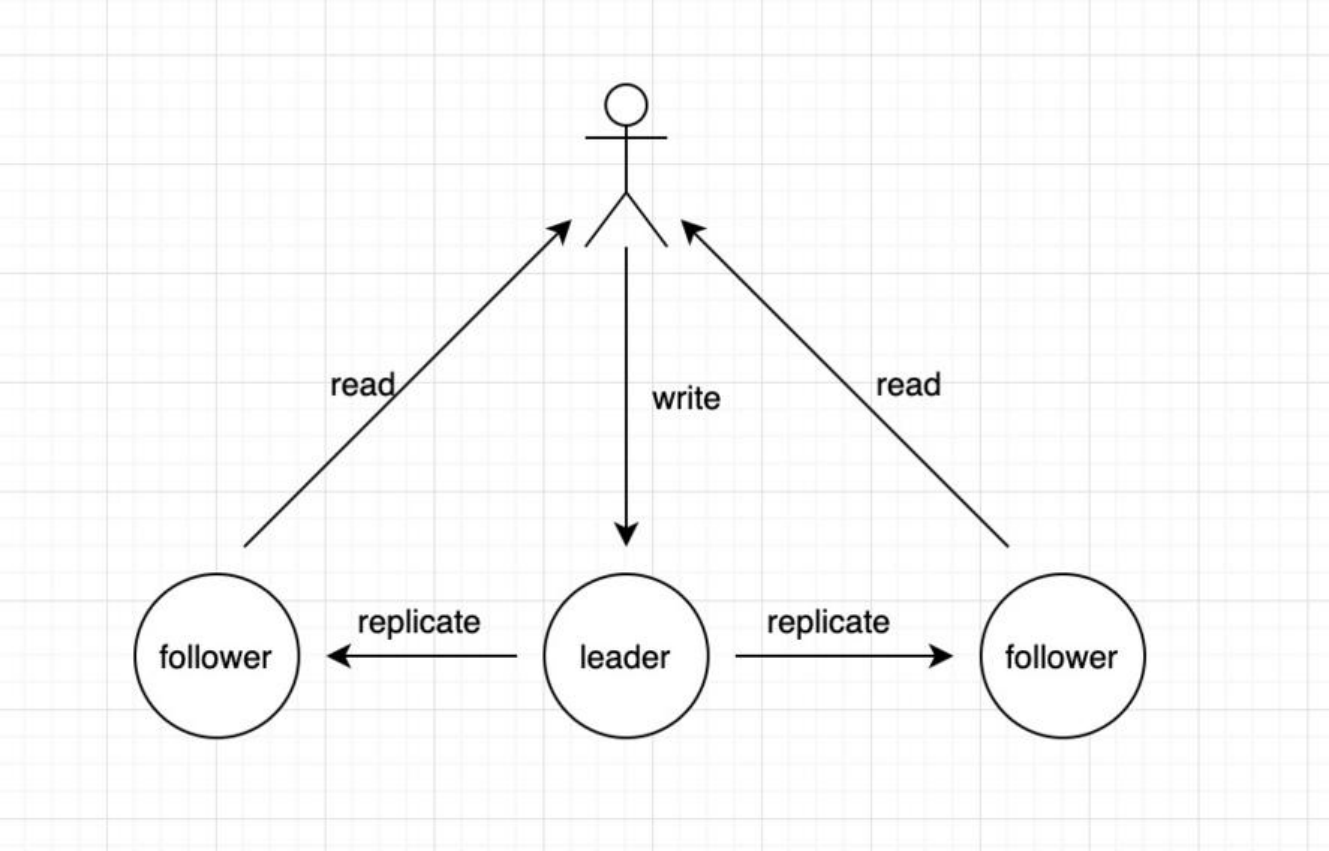
2. 数据延迟

数据延迟的问题解决方案上主要有三个方面：

1.

Replication的方式

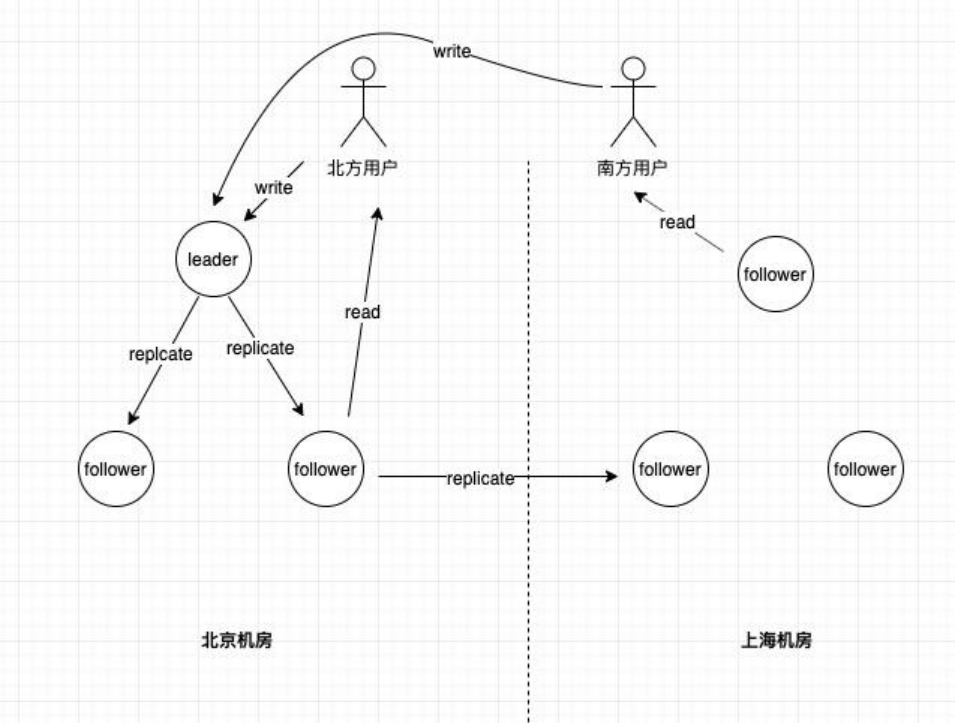
1. Leader And Followers（主从）



主从模式，这应该是最常见的replication方式了。所谓主从，就是保证写入数据是只写入到一个节点主节点写入后，把数据复制到其他从节点。读取数据时可以从所有节点读取。

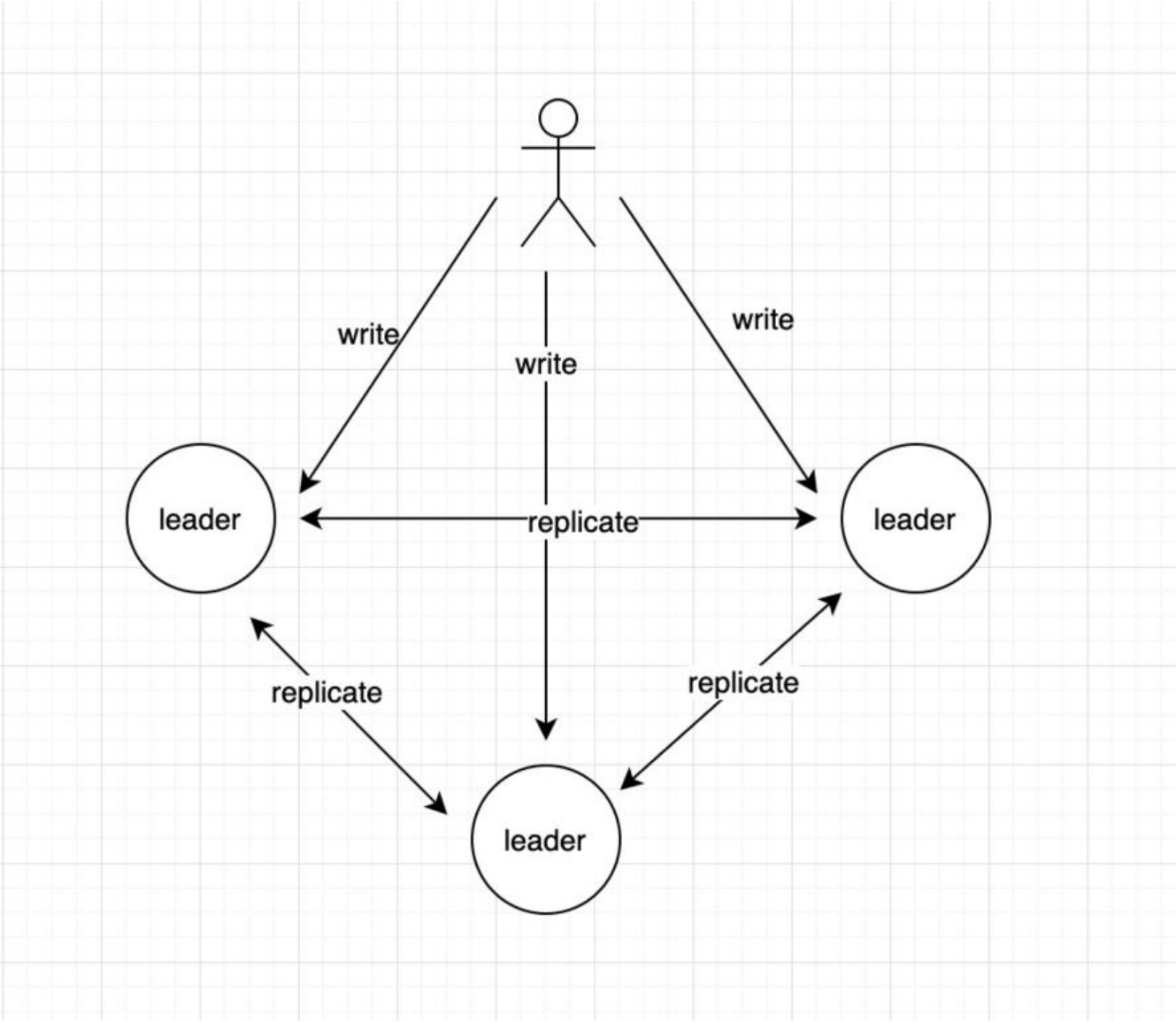
mysql主从模式，hbase，hdfs，zookeeper，等等，都是通过这种方式进行数据同步的。

即便是在跨地域的一些场景中，人们也倾向于一主多从，一写多读的模式。比如有北京、上海两个机，那么华北的用户读写自然都是北京机房，华东和华南的用户写数据时会跨越上千公里的电缆将数据入北京机房，然后再跨越上千公里把数据同步回上海机房，供华东和华南地区的用户读取。这种数据步方式的特点是可以避免数据冲突，随之而来的代价是数据跨地域写以及同步的延迟，有兴趣的同学以算算延迟有多少，相对论有言：但凡携带信息者，必低于光速，示意图如下：



2. Multi-Leader（多主）

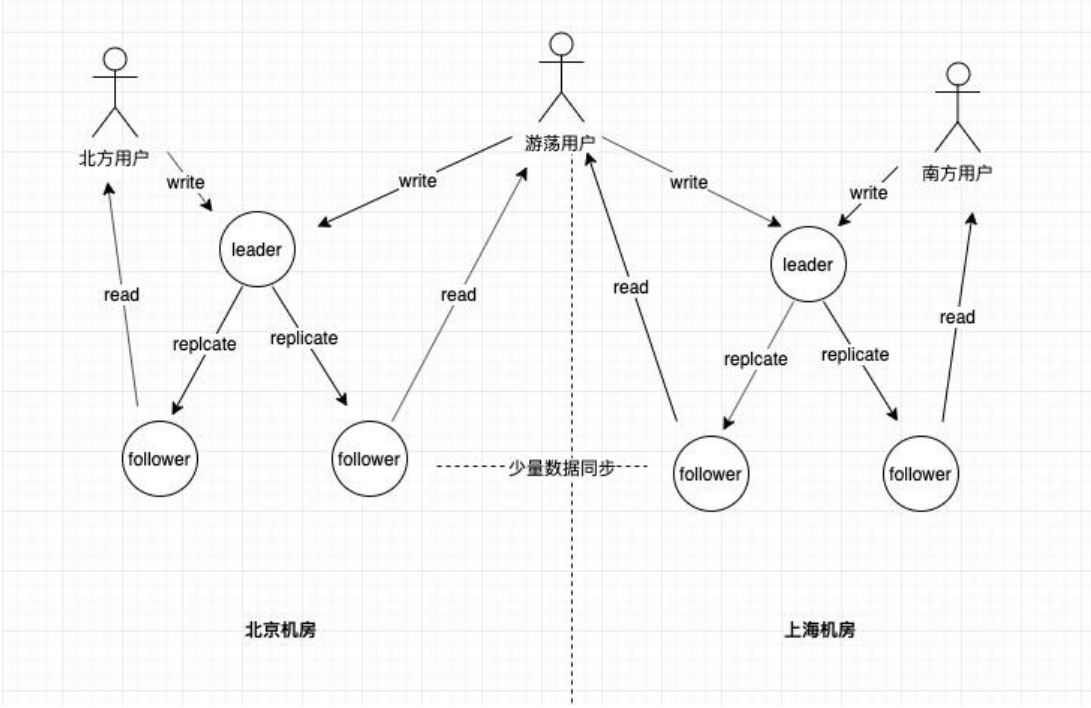
为避免跨地域带来的延迟，还有这样一种结构



所谓多主，就是很多个节点都可以写入数据，节点之间互为主从，互相同步数据。因为存在多个节点写入，所以很容易造成数据冲突。比如往leader1节点写入a=1，同时往leader2节点写入a=2，leade 1和leader2进行数据同步时就会出现冲突，无法确定最终a的值是1还是2。

为了解决冲突，有一个比较好用的办法：就是数据路由。拿用户来说吧，一个用户大多数时间活动区是固定的，比如本人，99%刷知乎的时间都是在北方。所以可以给用户打上一个地域的标签，如果是方的用户，那么读写都路由到北方的机房，如果不幸，这个用户出差到南方了，那么算他倒霉，仍然他路由到北方的机房读写，这时候体验可能会差一点点，但毕竟小概率。这种结构南北方互相独立，间可以不需要互相同步数据。如果有必要，可以只针对少量全国游荡、居无定所的用户进行跨地域的据同步。

整个示意图如下：



3. Leaderless（无中心）

还有一种无中心的结构，这种结构彻底抛弃了leader的概念，集群中所有节点的地位都是同等的，客端向集群写数据时，可以向任意节点写入，从集群读取数据时，可以从任意节点读取，不过有个限制是W+R>N。下面来解释一下这个公式是什么意思以及为什么有这个限制。

W+R>N

W：每次写入数据时，同时写入的节点数。

R：每次读取时，同时读取的节点数。

N：集群中的节点总数。

是不是很懵，为啥有这个么规定？

举个例子就知道了：

比如集群中有a、b、c三个节点，此时N=3，写入数据时，往b节点写入了，此时W=1，读取数据时怎样才能保证能读到刚才写入的数据呢？显然同时从a、b、c三个节点读取，然后比较数据的版本（入数据时带上版本号），发现b节点的数据最新，于是最终取b节点的数据作为最新的数据。

额，等等，这其中好像并没有replication啊。

确实，这种模式本身似乎并不需要replication也能跑的通，如果引入节点间的replication就会遇到上多主结构的问题：数据冲突。

这种无中心的结构，在亚马逊的DynamoDB中发挥的淋漓尽致。应用这种结构可以最大限度的保证据的写入，哪怕写入的数据本身存在冲突也无所谓，所有的冲突最终都会由客户端解决，好比读取时客户端从不同节点读到了同一个版本的不同数值，那么客户端自行决定用哪个好了。

客户端还有另一个任务：修复数据。当客户端获得了最新版本的数据后，会把这份数据写入到其他没这份最新数据的节点中，从这点看，似乎客户端把replication的事情做了。

为啥要修复数据呢？按上面W+R>N的理论，不修也能正常运行啊。

那么，如果可怜的W节点挂了呢？所以，还是要有relication，所以要修数据。

除此之外，这种W+R>N还有一个开挂搬的好处就是：在满足公式的情况下，W和R的数量可以根据要调节！当时度dynamo的论文时，着实被这种操作惊艳到了。

可以调节W和R意味着什么？

这意味着，对于写入要求很高的场景，把W调低，可以减少一次要写入的节点数，提高写入速度。同对于读取要求很高的场景，可以调低R的数值。

不过，这种无中心的结构在工程上实现起来很是繁琐麻烦，所以业内用的很少。

你看paxos理论上很好，但是用的最广泛的还是zab和raft。

总结

relication的结构无论怎样变来变去，最终的目的就是在享受其带来的好处时，应对其引发的问题。

无论哪种结构，其背后的思想都值得我们学习和思考，记住这些没用，真正有用的是将思想融会贯通运用到实际场景中去。

光有replication还不够，replication只是解决了容灾和访问延迟的一部分问题。剩下的问题，比如数量很大时，replciation也是无能为力的。给你一张几亿行的mysql表，再怎么replicaiton，再怎么读分离也不行，这时候就需要分而治之了，也就是分库分表，或者用一种更通用的说法：且看下回

存储（八）—— Partition