

SQL-NoSQL-NewSQL

数据库架构演变

张甦 (su)

关于我

张甦，

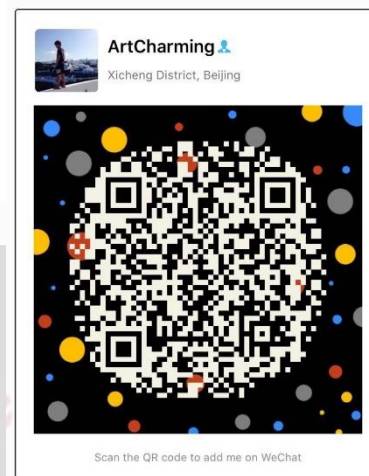
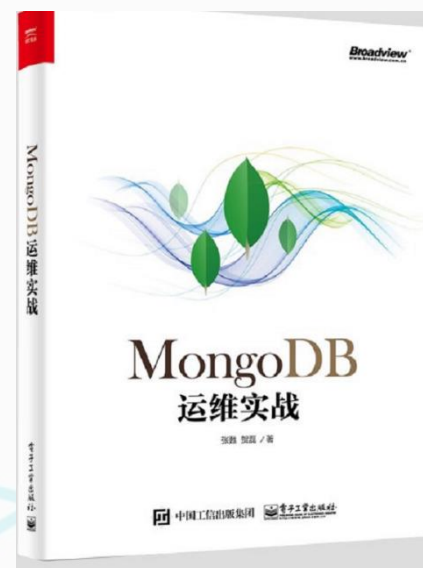
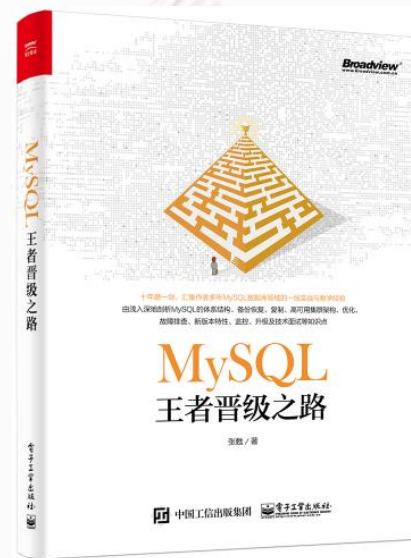
数据库专家兼专家讲师

51CTO专家博主

《MySQL王者晋级之路》

《MongoDB运维实战》作者。

近10年互联网线上处理及培训经验，专注于MySQL,pg,MongoDB，以及巨杉，TiDB等NewSQL数据库有深入研究。曾就职于数据库服务公司，某大型电商平台，及汽车类网站等大型互联网公司。麾下的学员遍布各大企业。



目录

CONTENTS

A

写书的初衷

B

数据库架构演变

01

写书的初衷

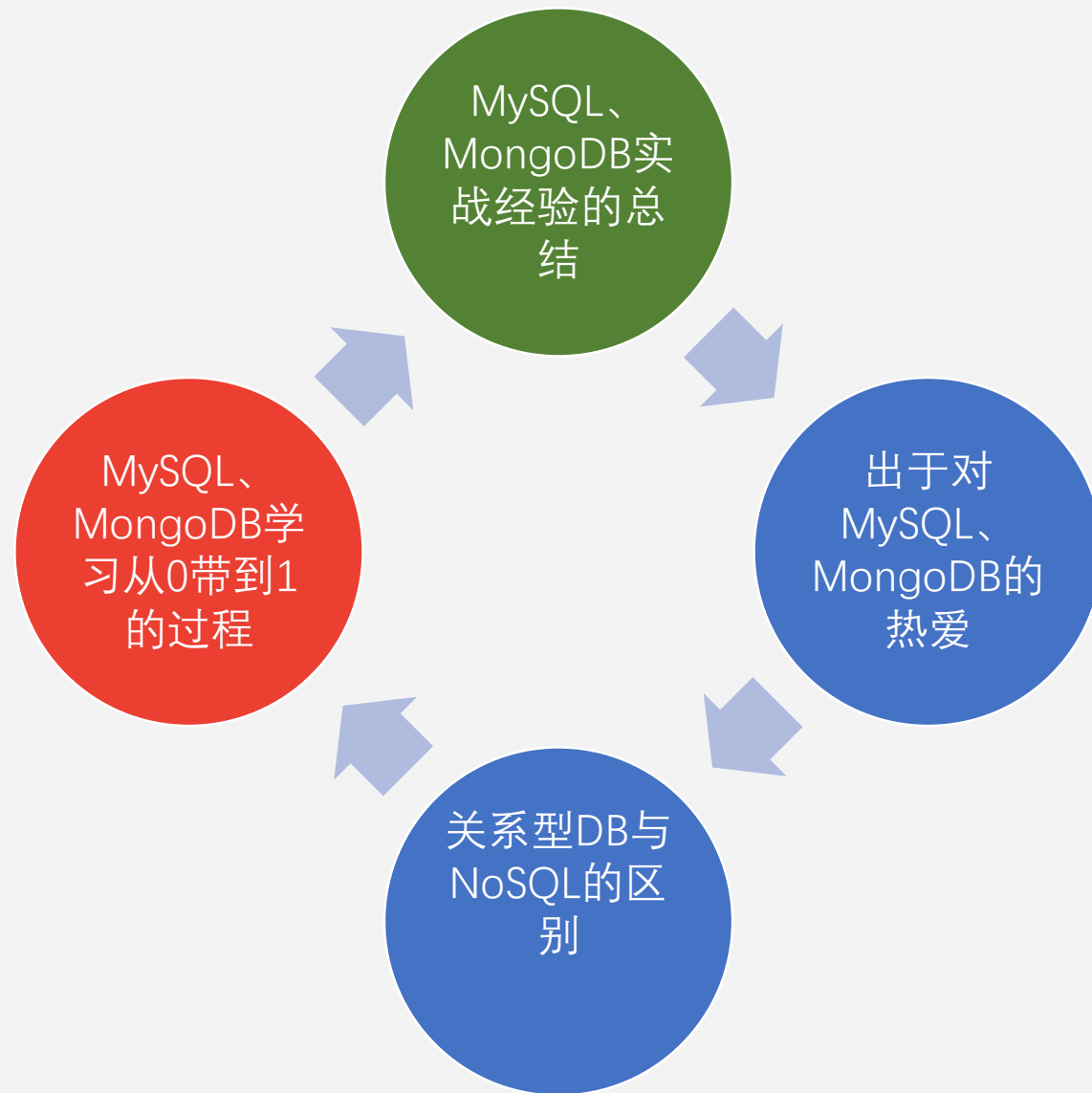
MongoDB全书核心脉络



MySQL全书核心脉络



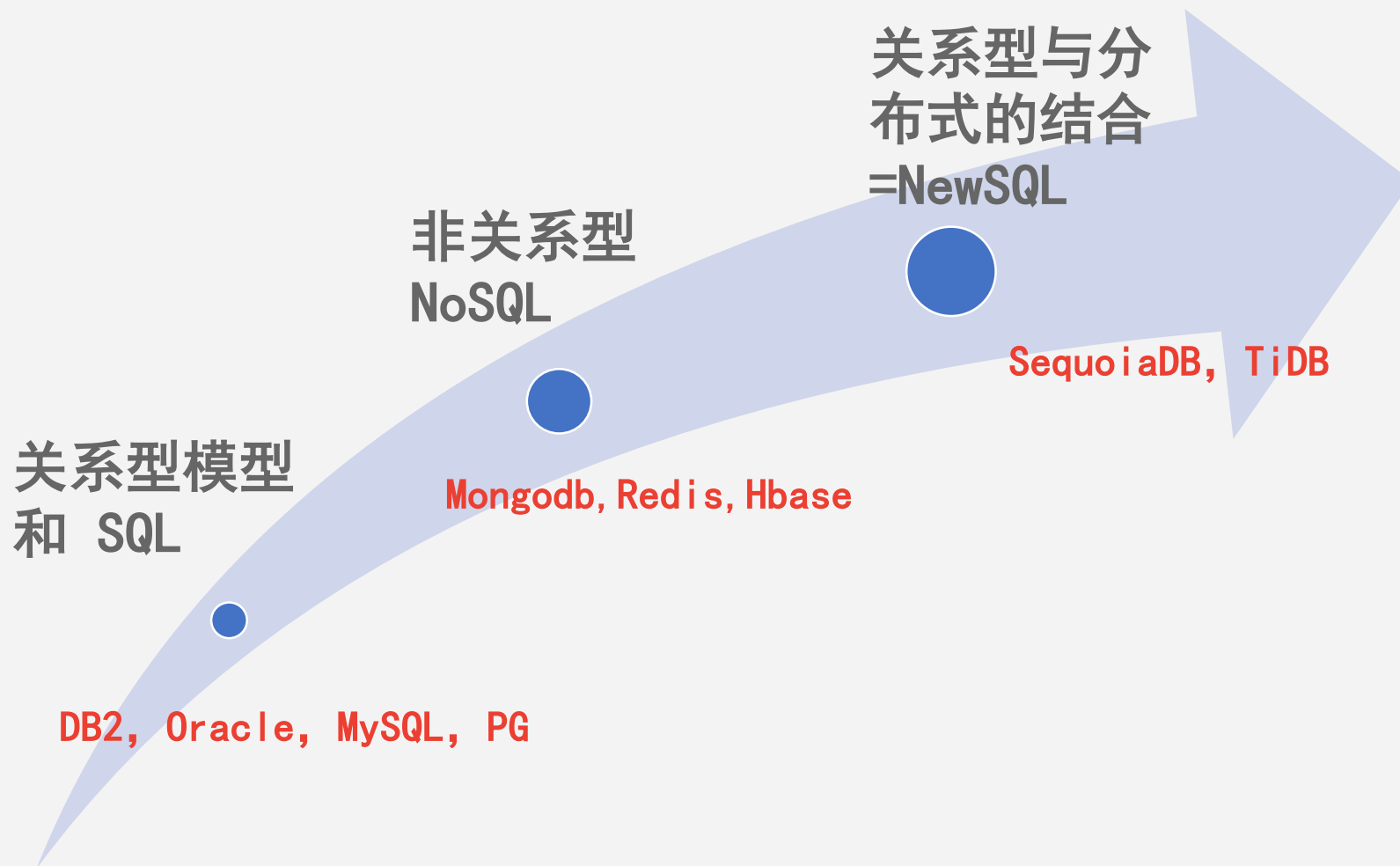
初衷——四因素






























数据库架构演变

数据库发展



DB-Engines-Ranking

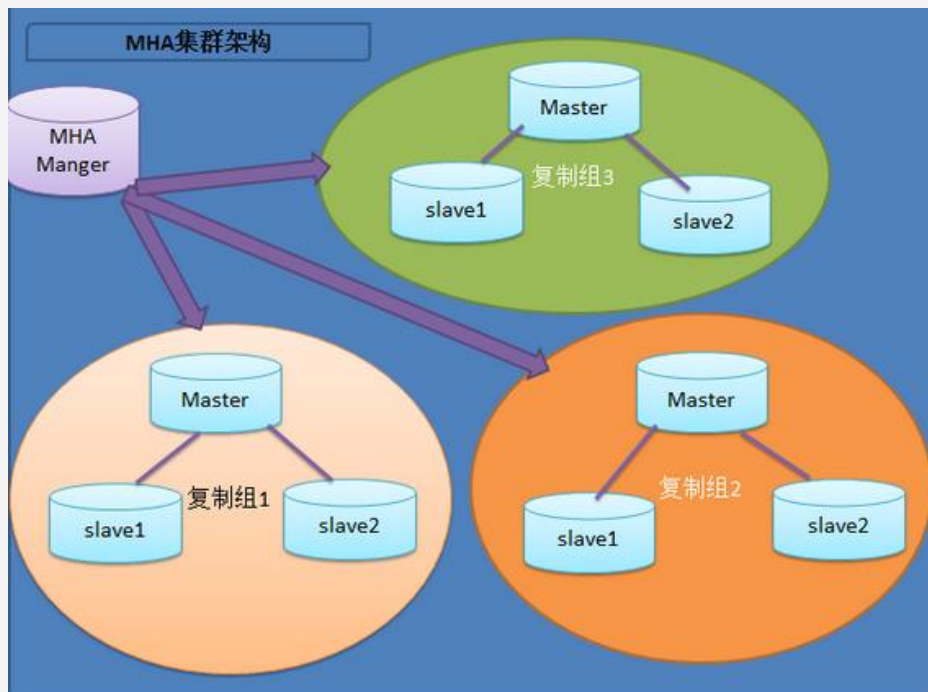
Rank			DBMS	Database Model	Score		
Jul 2019	Jun 2019	Jul 2018			Jul 2019	Jun 2019	Jul 2018
1.	1.	1.	Oracle 	Relational, Multi-model 	1321.26	+22.04	+43.47
2.	2.	2.	MySQL 	Relational, Multi-model 	1229.52	+5.89	+33.45
3.	3.	3.	Microsoft SQL Server 	Relational, Multi-model 	1090.83	+3.07	+37.42
4.	4.	4.	PostgreSQL 	Relational, Multi-model 	483.28	+6.65	+77.47
5.	5.	5.	MongoDB 	Document	409.93	+6.03	+59.60
6.	6.	6.	IBM Db2 	Relational, Multi-model 	174.14	+1.94	-12.06
7.	7.	 8.	Elasticsearch 	Search engine, Multi-model 	148.81	-0.01	+12.59
8.	8.	 7.	Redis 	Key-value, Multi-model 	144.26	-1.86	+4.35
9.	9.	9.	Microsoft Access	Relational	137.31	-3.70	+4.73
10.	10.	10.	Cassandra 	Wide column	127.00	+1.82	+5.95
11.	11.	11.	SQLite 	Relational	124.63	-0.26	+9.35
12.	 13.	 13.	Splunk	Search engine	85.49	+0.87	+16.25
13.	 12.	 14.	MariaDB 	Relational, Multi-model 	84.44	-0.76	+16.92

分久必合



MySQL高可用集群

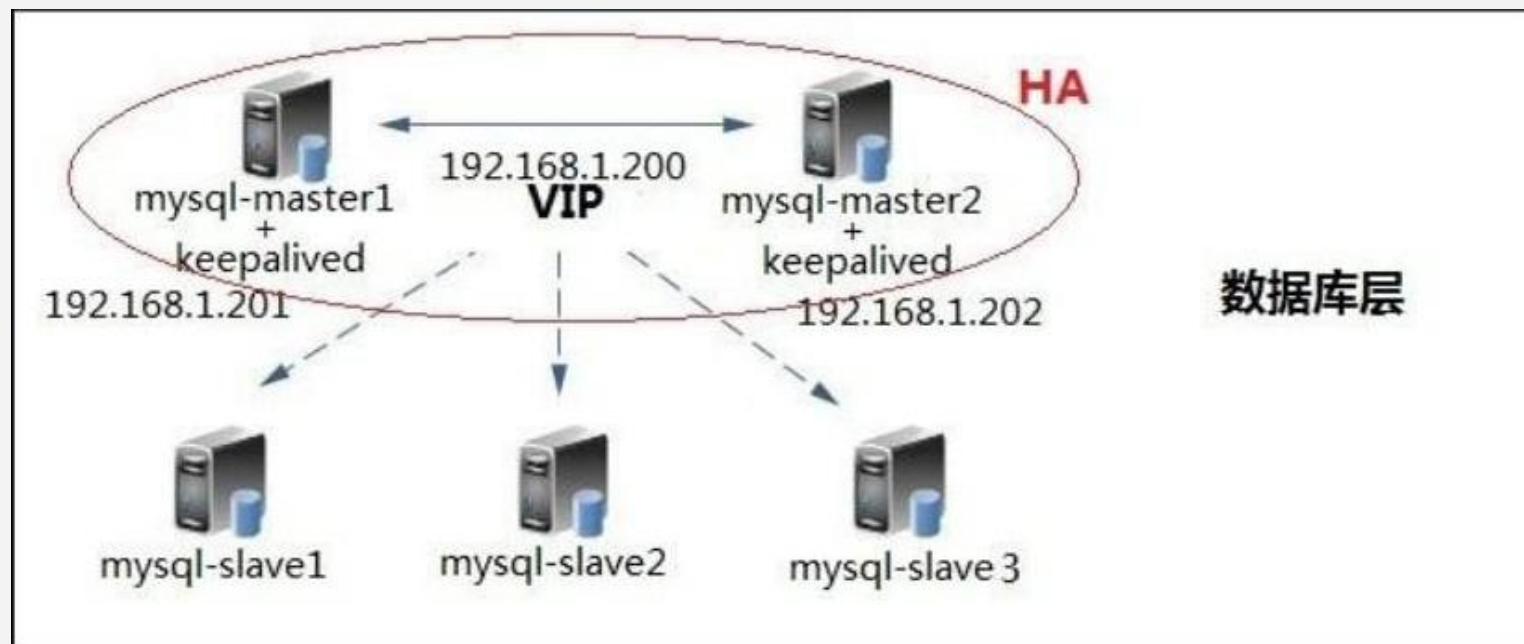
基于主从架构的MHA



MHA的目的在于维持MySQL Replication中master库的高可用性，其最大特点是可以修复多个slave之间的差异日志，最终使所有slave保持数据一致，然后从中选择一个充当新的master，并将其他slave指向它。当master出现故障时，可以通过对比slave之间I/O thread 读取主库binlog的position号，选取最接近的slave作为备选主库（备胎）。其他的从库可以通过与备选主库对比生成差异的中继日志。在备选主库上应用从原来master保存的binlog，同时将备选主库提升为master。最后在其他slave上应用相应的差异中继日志并从新的master开始复制。

MySQL高可用集群

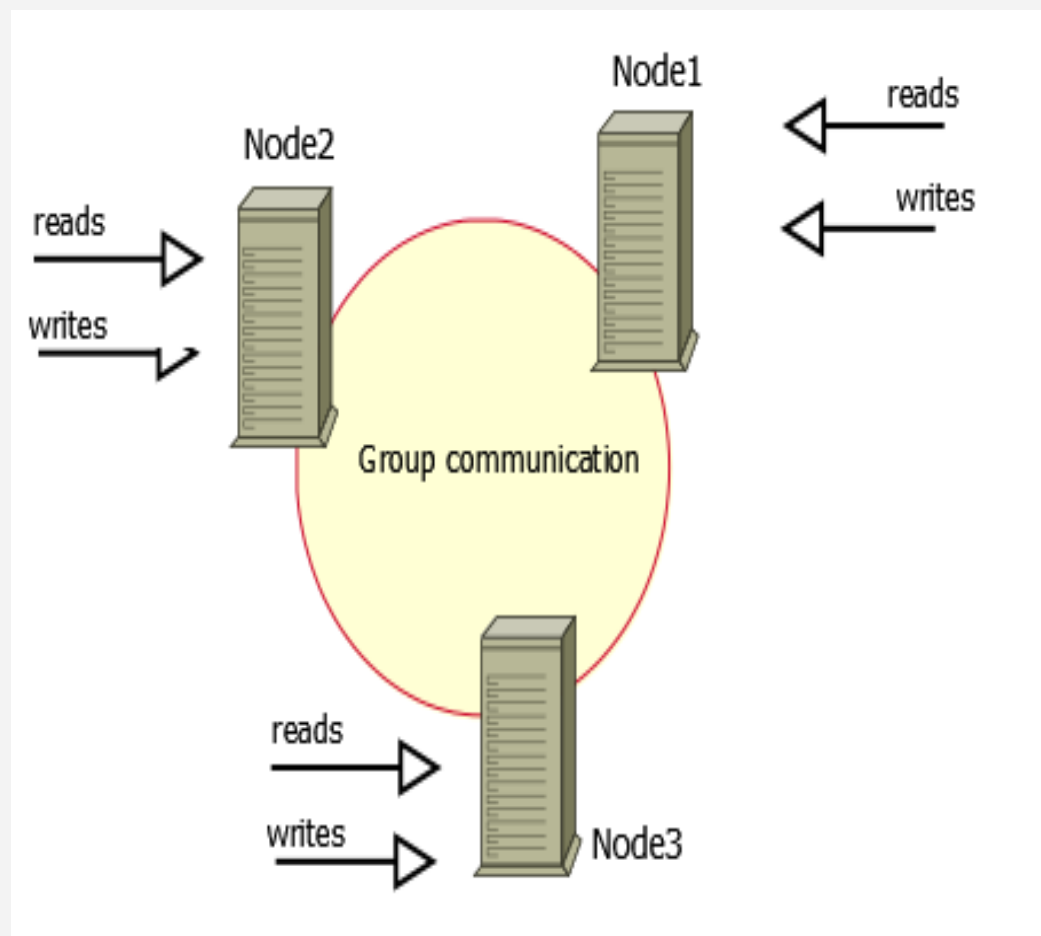
基于主从架构的双节点+keepalived



中小型规模的时候，采用这种架构是最省事的。两个节点可以采用简单的一主一从模式，或者双主模式，并且放置于同一个VLAN中，在master节点发生故障后，利用keepalived/heartbeat的高可用机制实现快速切换到slave节点。

MySQL高可用集群

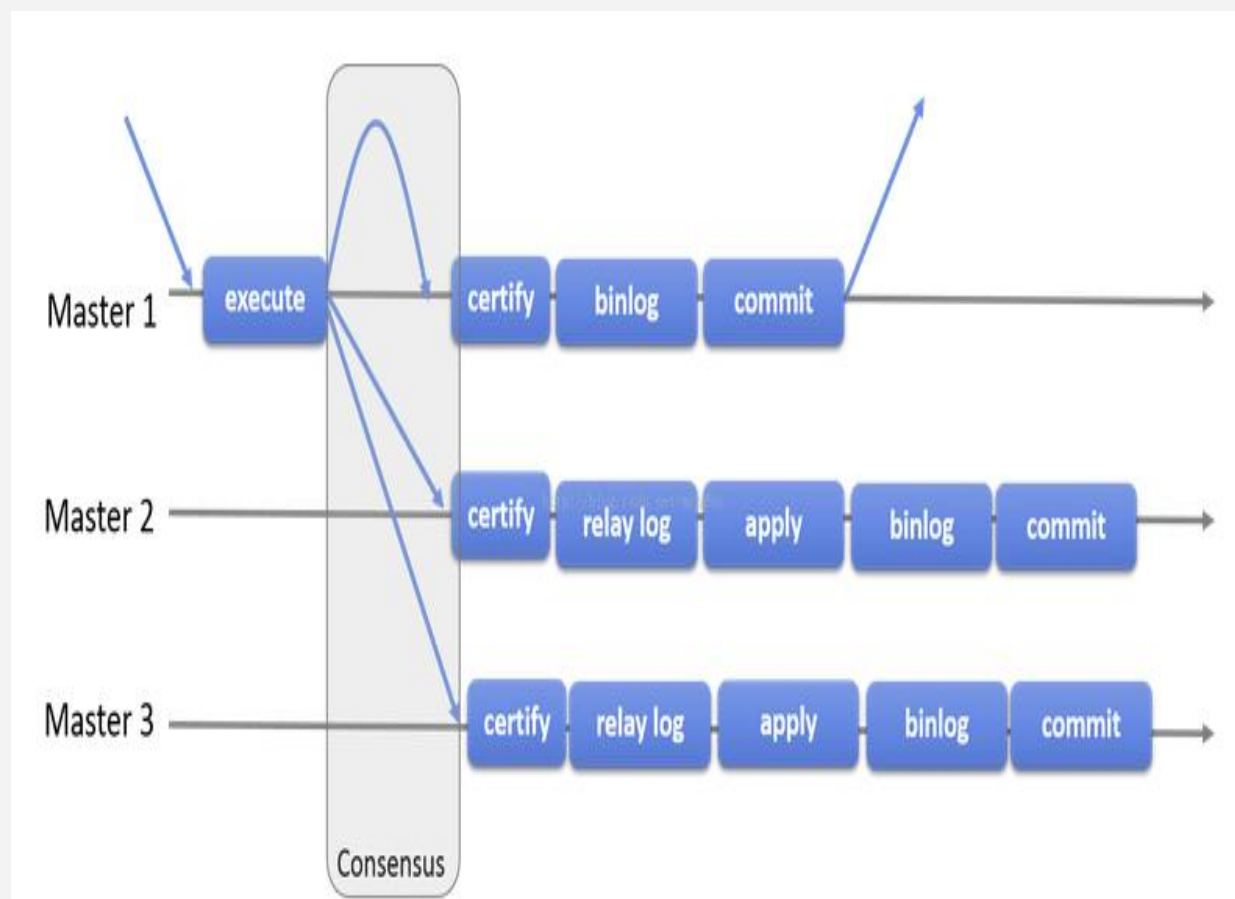
PXC架构



PXC是基于Galera协议的MySQL高可用集群架构。Galera产品是以Galera Cluster方式为MySQL提供高可用集群解决方案的。Galera Cluster就是集成了Galera插件的MySQL集群。Galera replication是Codership提供的MySQL数据同步方案，具有高可用性，方便扩展，并且可以实现多个MySQL节点间的数据同步复制与读写，可保障数据库的服务高可用及数据强一致性。

MySQL高可用集群

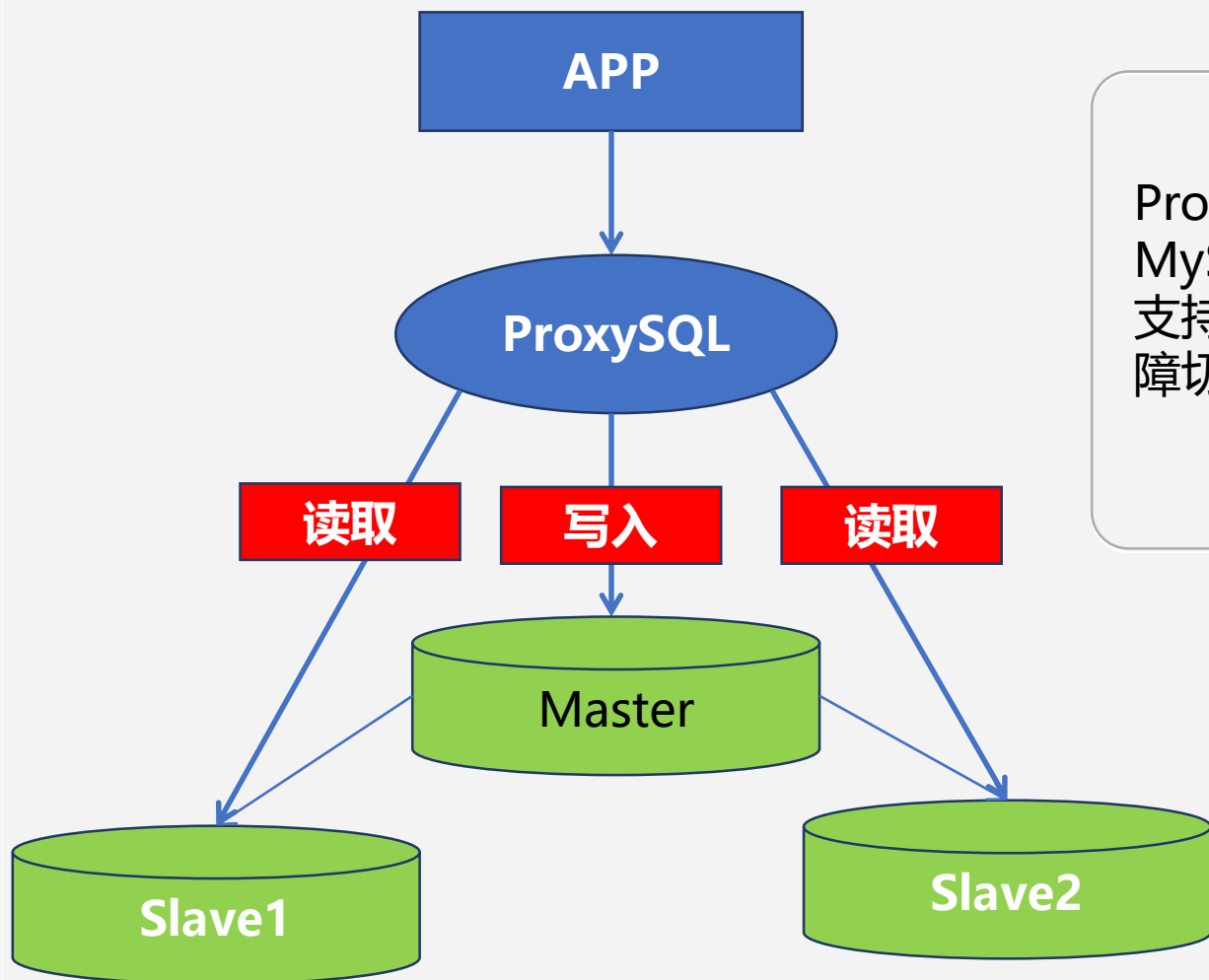
MGR架构



MySQL官方在5.7.17版本正式推出组复制（MySQL Group Replication，简称MGR）。master1, master2, master3, 所有成员独立完成各自的事务。当客户端先发起一个更新事务，该事务先在本地执行，执行完成之后就要发起对事务的提交操作了。在还没有真正提交之前需要将产生的复制写集广播出去，复制到其他成员。如果冲突检测成功，组内决定该事务可以提交，其他成员可以应用，否则就回滚。最终，这意味着所有组内成员以相同的顺序接收同一组事务。因此组内成员以相同的顺序应用相同的修改，保证组内数据强一致性。

强大的MySQL高可用集群

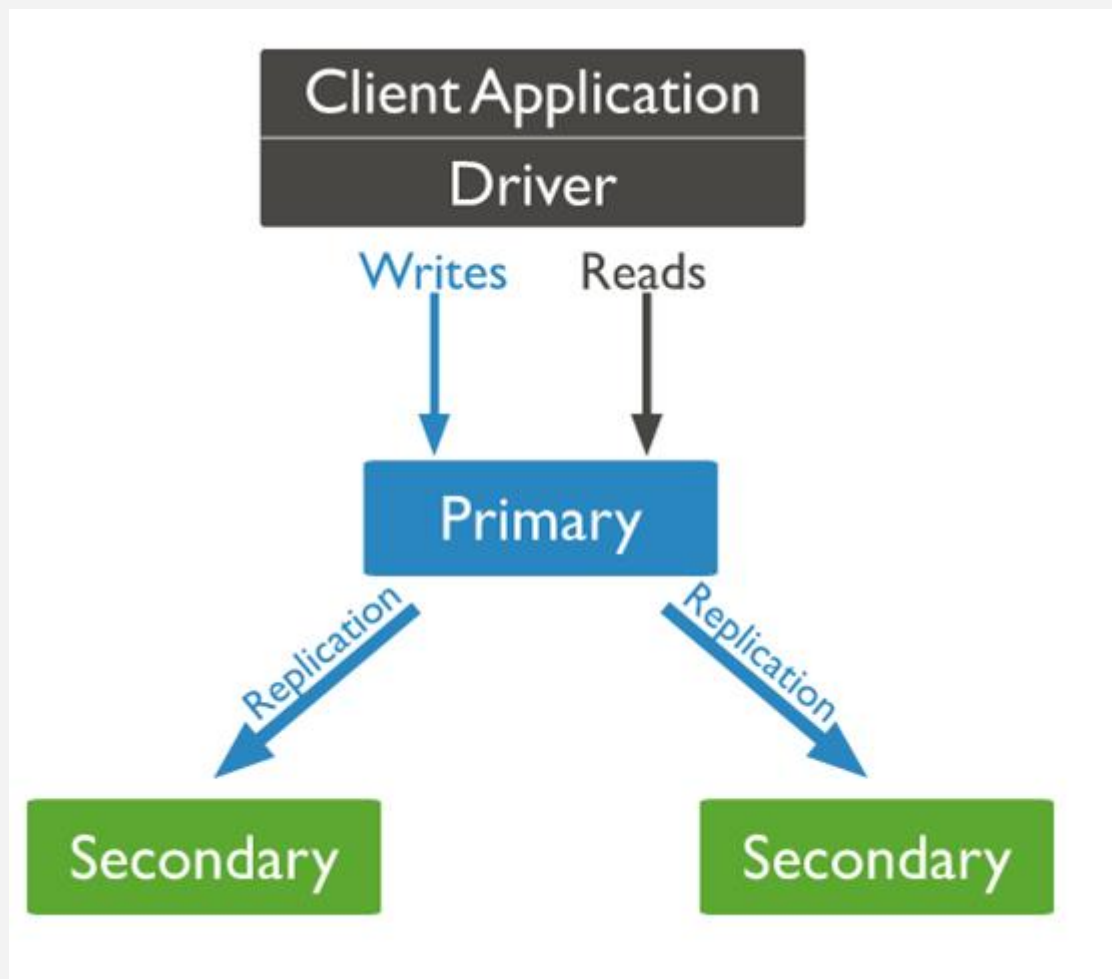
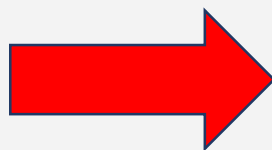
读写分离的实现 ProxySQL



ProxySQL是MySQL的一款中间件的产品，是灵活强大的MySQL代理层。可以实现读写分离，支持Query路由功能，支持动态指定某个SQL进行cache，支持动态加载配置、故障切换和一些SQL的过滤功能

MongoDB复制集

三副本架构是最基础的复制集的架构，一主两备模式。主节点接受外界的阅读请求，向备节点进行数据同步。当主节点宕掉，会自动切换到备节点，不影响线上业务，防止单点故障。

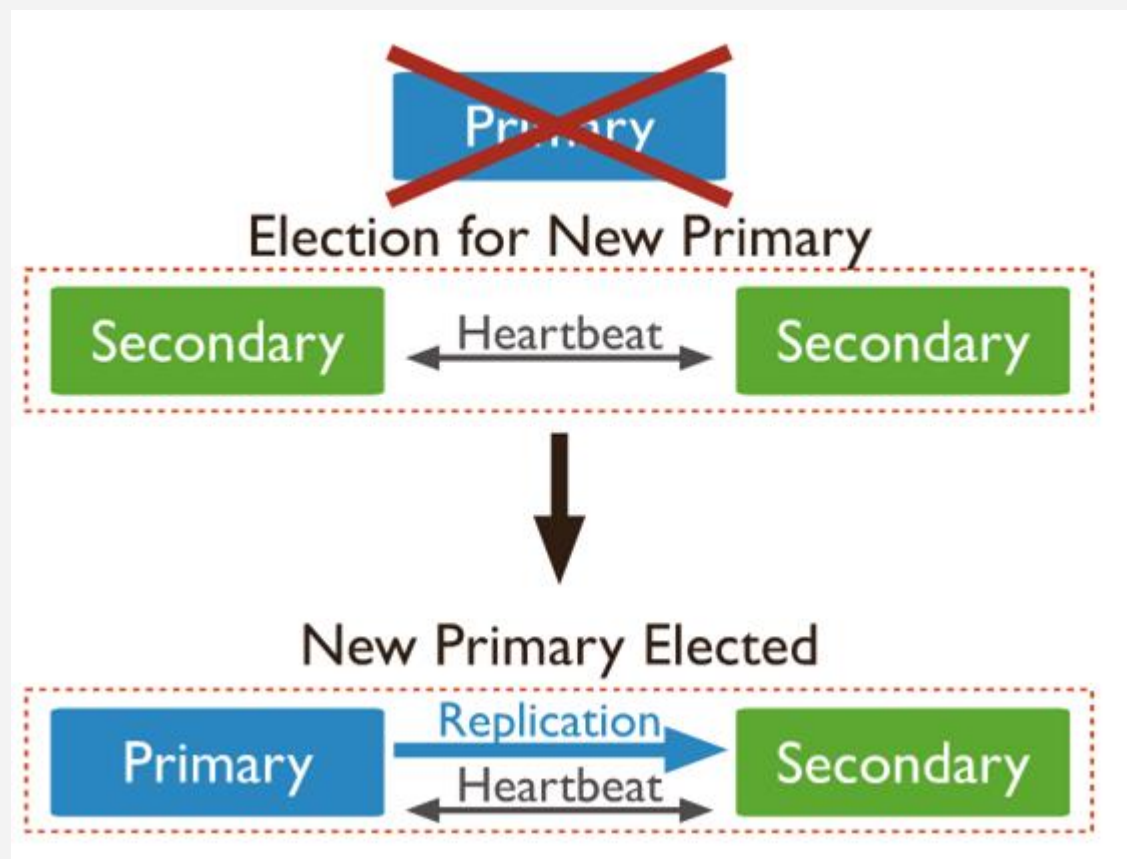


MongoDB复制集自动切换

副本集的所有成员都可以接受读取操作。但是，默认情况下，应用程序将其读取操作指向primary。

副本集可以有至多一个primary节点，primary节点宕机后，集群会触发选举以选出新的primary节点

在以下三成员节点副本集架构中，primary宕机后，触发了一次选举，从剩下的两个secondary节点里，选举出了一个新的primary节点。



MongoDB复制集读写分离设置

read preference 决定MongoDB客户端从哪个节点上读取数据。

默认情况下，应用程序将其读取操作指向副本集中的**primary**节点。

指定**read preference** 选项时要注意：因为使用异步复制，复制延迟会导致**secondary**上的数据可能不是最新的。

默认情况下，复制集的所有读请求都发到**Primary**，**Driver**可通过设置**Read Preference**来将读请求路由到其他的节点。

primary：默认规则，所有读请求发到**Primary**

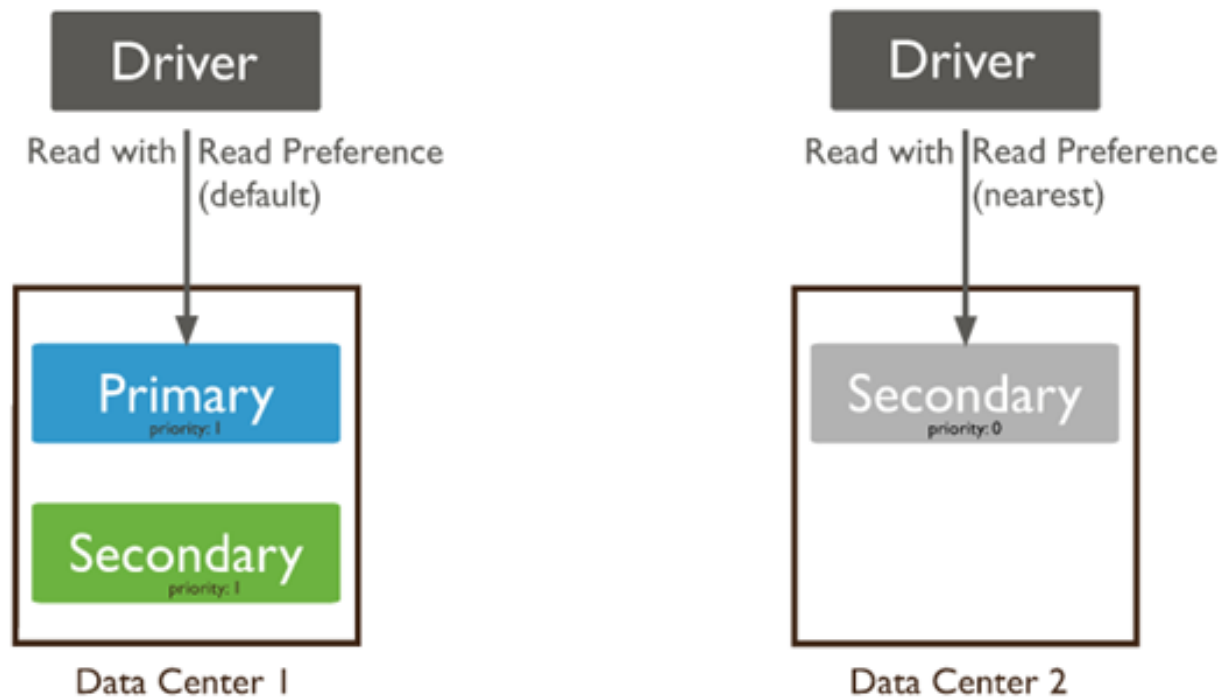
primaryPreferred： **Primary**优先，如果**Primary**不可达，请求**Secondary**

secondary： 所有的读请求都发到**secondary**

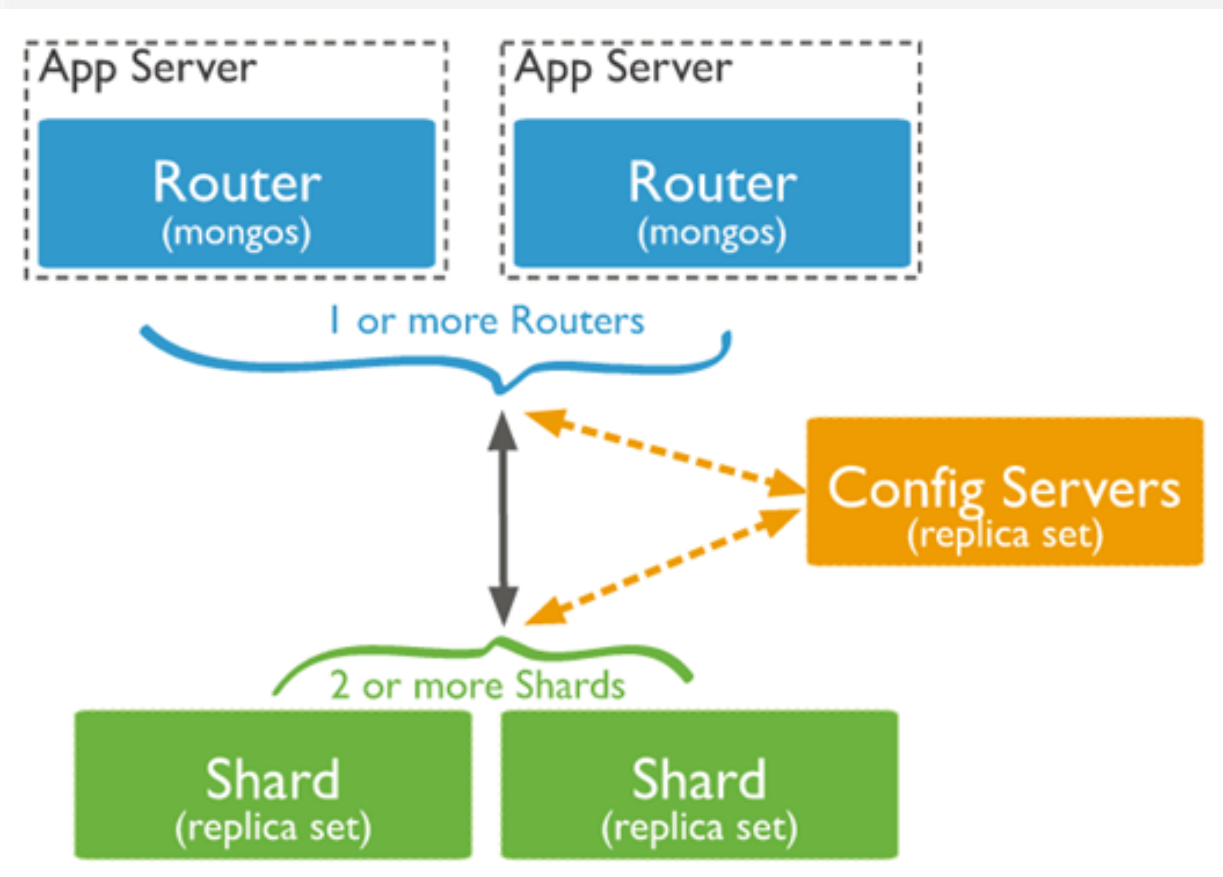
secondaryPreferred： **Secondary**优先，当所有

Secondary不可达时，请求**Primary**

nearest： 读请求发送到最近的可达节点上（通过ping探测得出最近的节点）



MongoDB分片架构



分片是一种在多台机器上分配数据的方法。MongoDB使用分片架构有助于您去管理非常大数量的数据集和高吞吐量操作的集群。大数据量和高吞吐量的业务情况对单台服务器来讲是具备很大的挑战性的。例如，高查询率可能耗尽服务器的CPU容量。工作集大小超过系统内存，那么压力则会给到磁盘上，这对IO来讲不是我们所希望看到的。

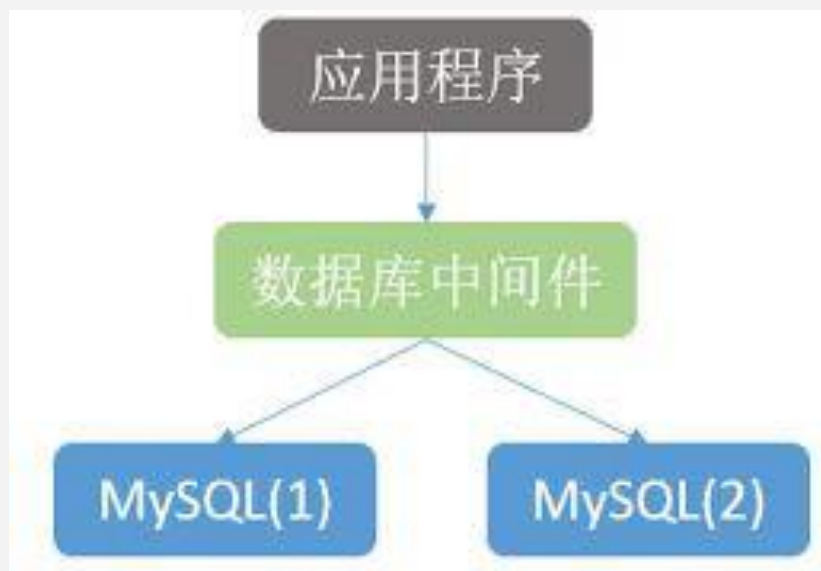
MongoDB支持通过分片进行水平缩放。

数据量爆发式增长, 数据存不下了 咋办?



SQL模型-中间件应用

具体中间件产品：MyCAT/ProxySQL/Cobar/Kingshard/Vitess



优势

简单，相对成熟

开源方案

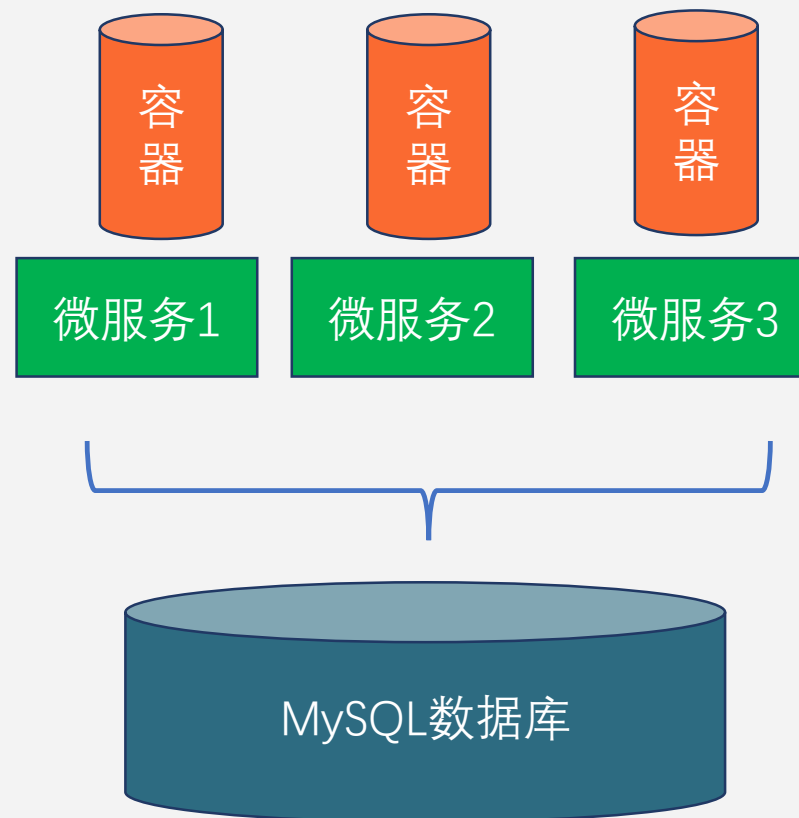
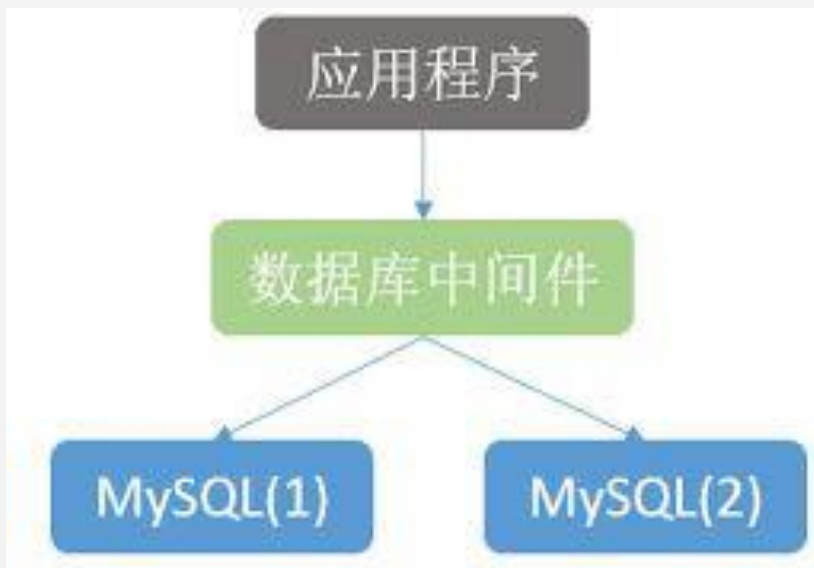
劣势

影响应用吞吐量

扩容受限

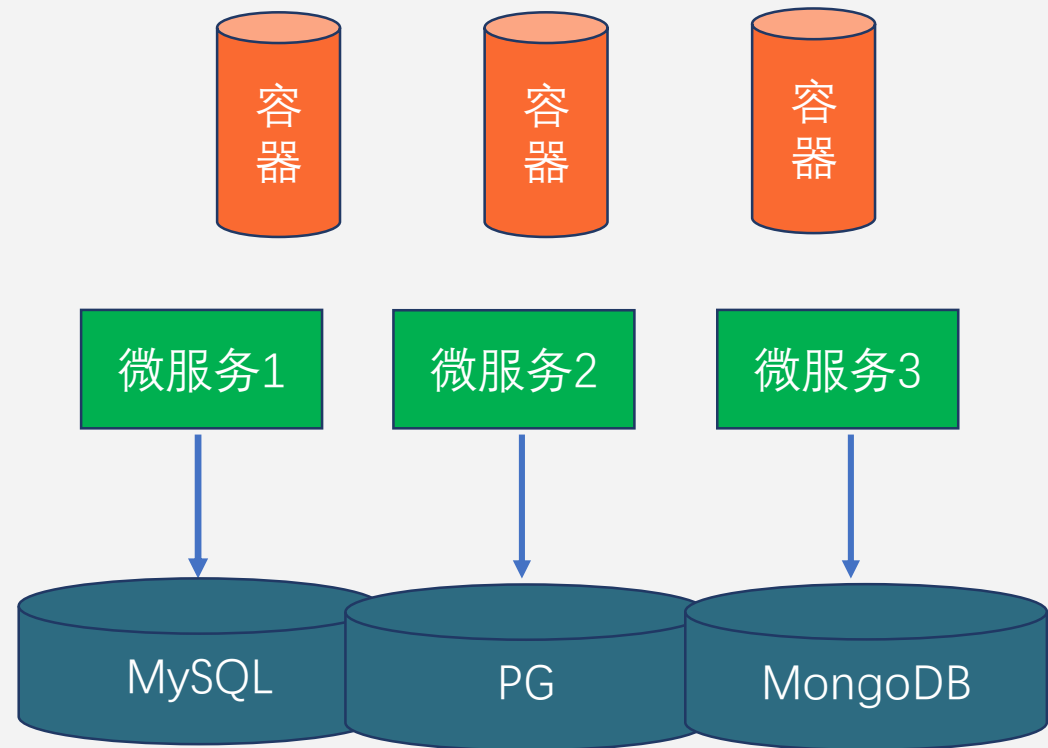
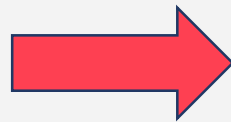
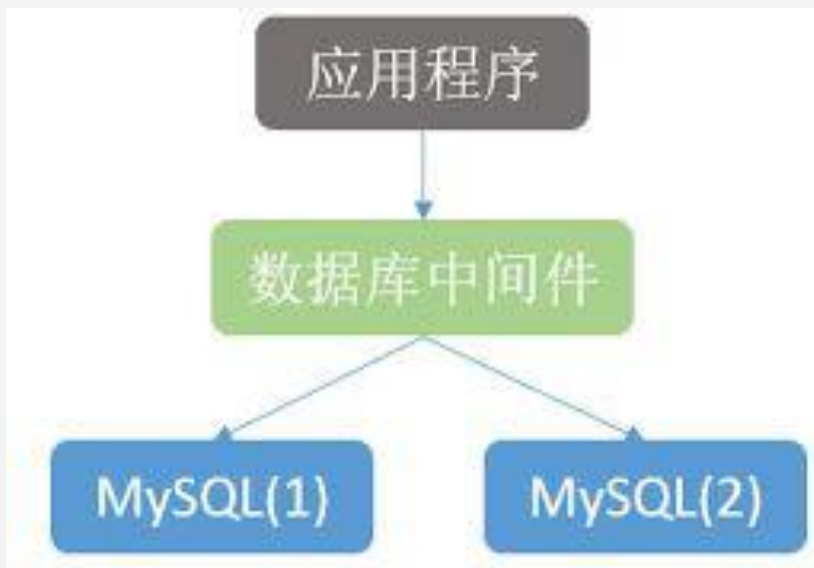
跨库join，跨库的一致性很难保证

SQL模型-转换为微服务 集中化存储



总结：数据高度耦合，如果增加微服务数量，底层数据库无法扩展，那整体性能还是会受到限制

SQL模型-转换为微服务 分散化存储



总结：运维成本高，底层数据库还是会存在容量上限问题，还需要考虑扩展问题

NoSQL模型

为了达到水平扩展能力，解决数据存不下问题；考虑利用更简单的存储模型

放弃标准的关系模型

放弃SQL子集

放弃join

放弃SQL

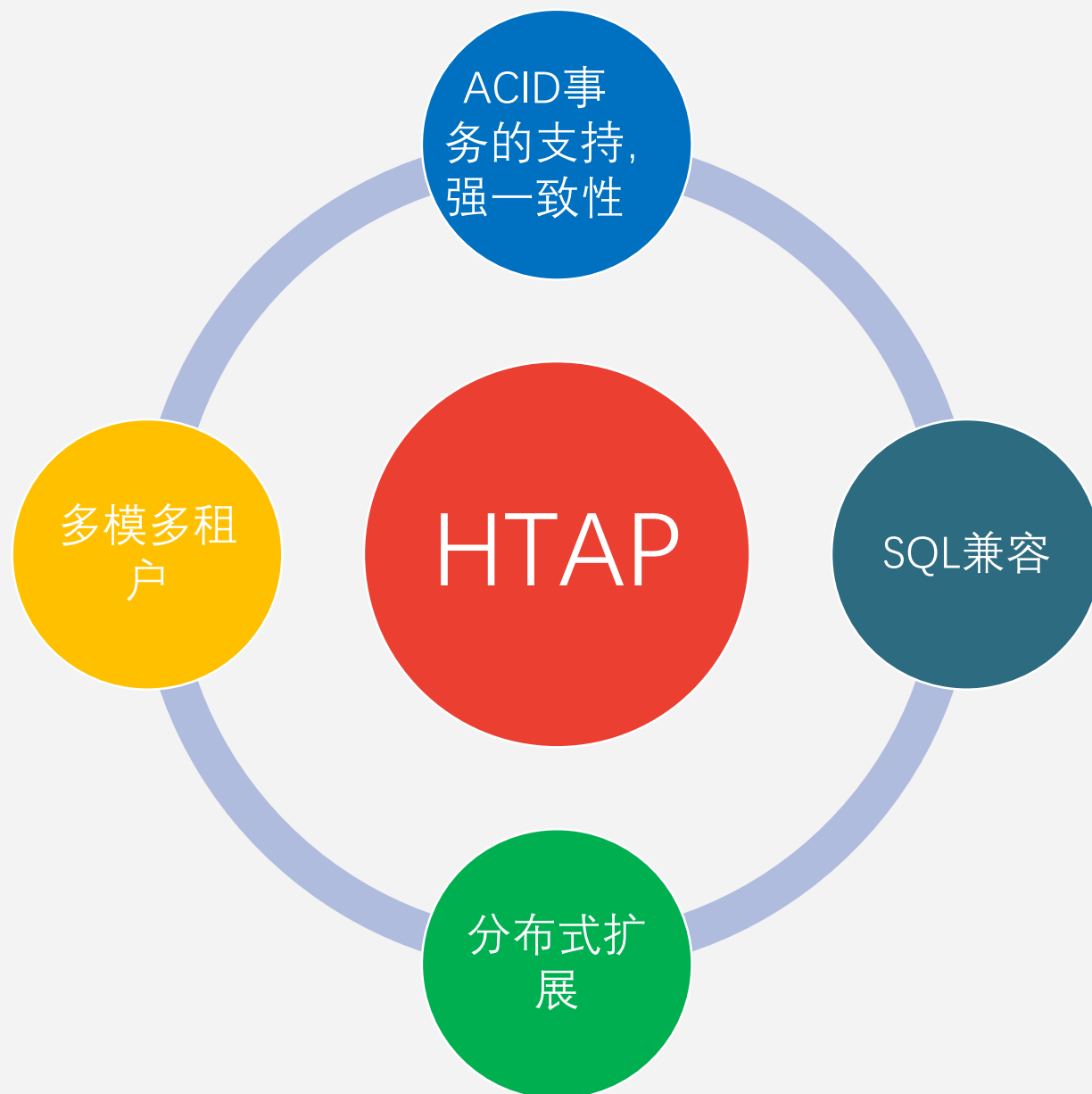
放弃事务

弱一致性

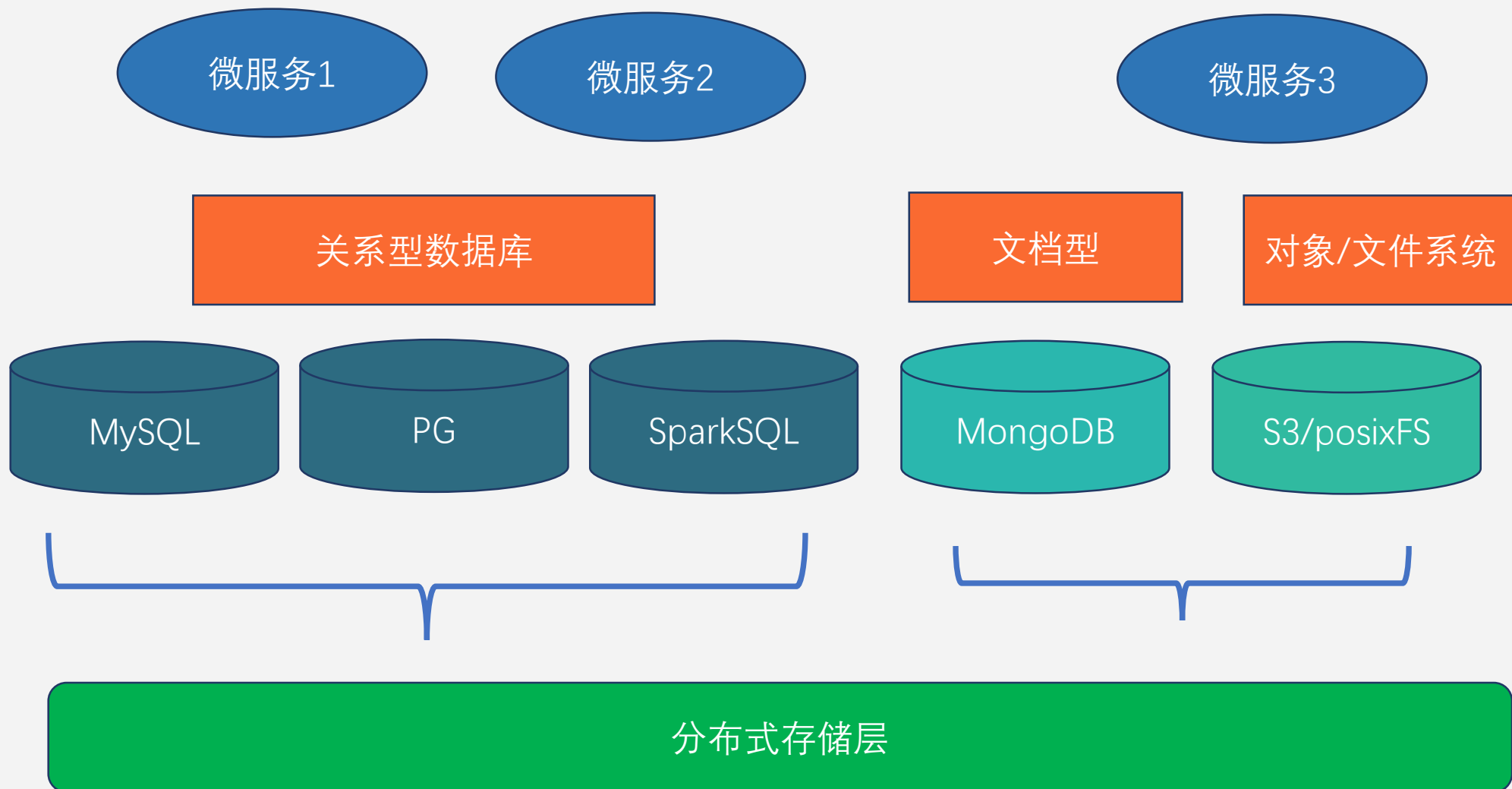
业务逻辑来考虑实现事务

事务的速度和精准性无法保证

NewSQL模型—特性

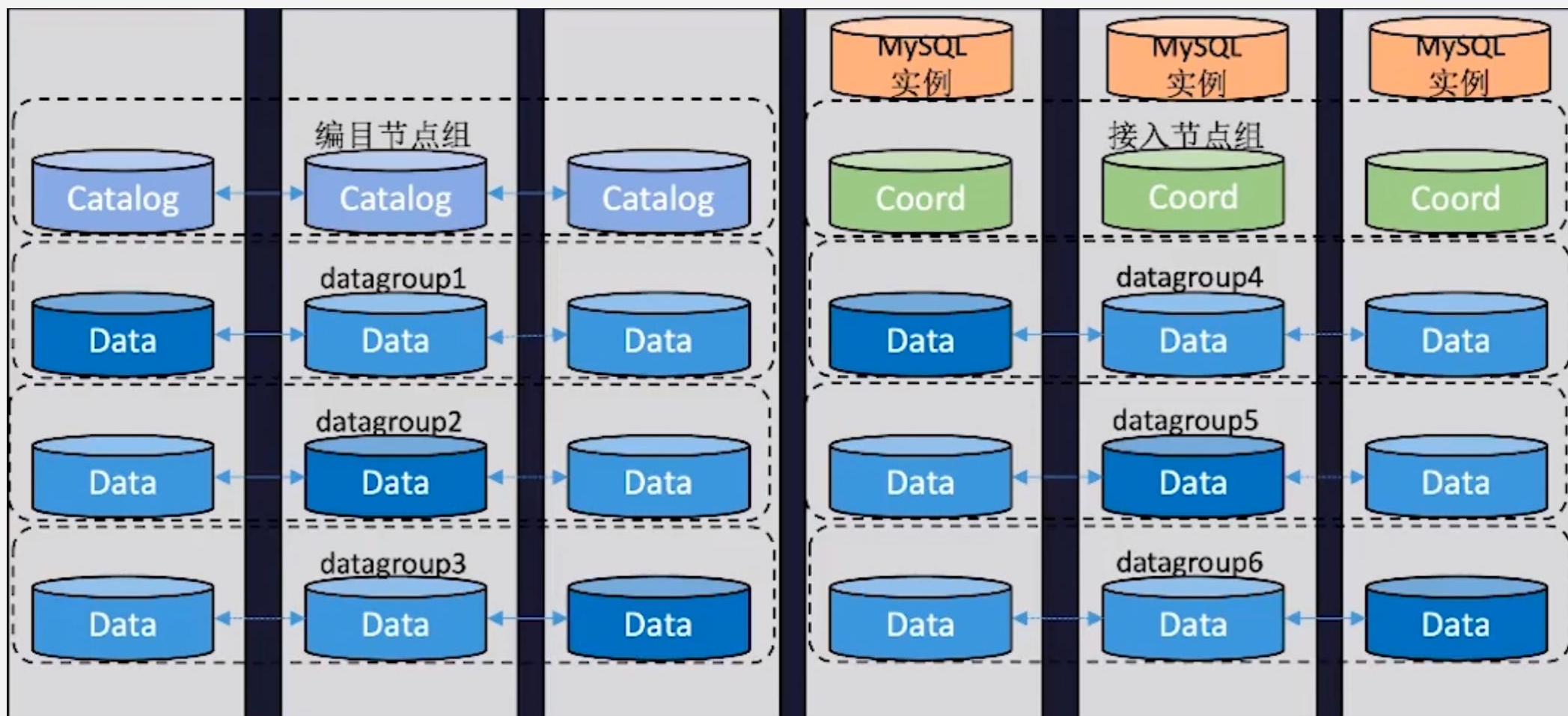


NewSQL模型—巨杉架构图



NewSQL模型—巨杉架构图

整体架构预览





**感谢
一路有你
SQL
NoSQL
NewSQL**