14 | 高性能数据库集群：读写分离

2018-05-29 李运华

14 | 高性能数据库集群：读写分离
朗读人：黄洲君　09′24″ | 4.31M

“从 0 开始学架构”专栏已经更新了 13 期，从各个方面阐述了架构设计相关的理论和流程，包括架构设计起源、架构设计的目的、常见架构复杂度分析、架构设计原则、架构设计流程等，掌握这些知识是做好架构设计的基础。

在具体的实践过程中，为了更快、更好地设计出优秀的架构，除了掌握这些基础知识外，还需要掌握业界已经成熟的各种架构模式。大部分情况下，我们做架构设计主要都是基于已有的成熟模式，结合业务和团队的具体情况，进行一定的优化或者调整；即使少部分情况我们需要进行较大的创新，前提也是需要对已有的各种架构模式和技术非常熟悉。

接下来，我将逐一介绍最常见的“高性能架构模式”“高可用架构模式”“可扩展架构模式”，这些模式可能你之前大概了解过，但其实每个方案里面都有很多细节，只有深入的理解这些细节才能理解常见的架构模式，进而设计出优秀的架构。

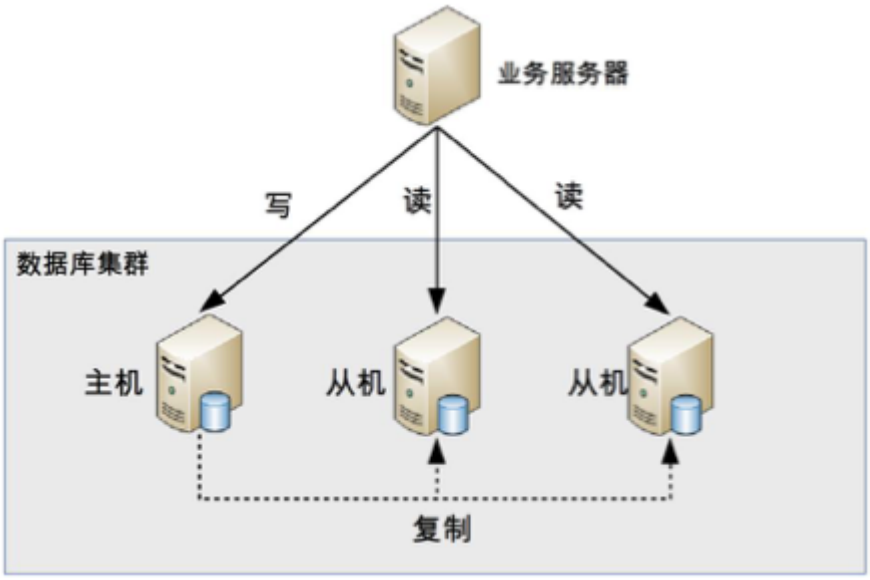
虽然近十年来各种存储技术飞速发展，但关系数据库由于其 ACID 的特性和功能强大的 SQL 查询，目前还是各种业务系统中关键和核心的存储系统，很多场景下高性能的设计最核心的部分就是关系数据库的设计。

不管是为了满足业务发展的需要，还是为了提升自己的竞争力，关系数据库厂商（Oracle、DB2、MySQL 等）在优化和提升单个数据库服务器的性能方面也做了非常多的技术优化和改进。但业务发展速度和数据增长速度，远远超出数据库厂商的优化速度，尤其是互联网业务兴起之后，海量用户加上海量数据的特点，单个数据库服务器已经难以满足业务需要，必须考虑数据库集群的方式来提升性能。

从今天开始，我会分几期来介绍高性能数据库集群。高性能数据库集群的第一种方式是“读写分离”，其本质是将访问压力分散到集群中的多个节点，但是没有分散存储压力；第二种方式是“分库分表”，既可以分散访问压力，又可以分散存储压力。先来看看“读写分离”，下一期我再介绍“分库分表”。

读写分离原理

读写分离的基本原理是将数据库读写操作分散到不同的节点上，下面是其基本架构图。



读写分离的基本实现是：

- 数据库服务器搭建主从集群，一主一从、一主多从都可以。
- 数据库主机负责读写操作，从机只负责读操作。
- 数据库主机通过复制将数据同步到从机，每台数据库服务器都存储了所有的业务数据。
- 业务服务器将写操作发给数据库主机，将读操作发给数据库从机。

需要注意的是，这里用的是“主从集群”，而不是“主备集群”。“从机”的“从”可以理解为“仆从”，仆从是要帮主人干活的，“从机”是需要提供读数据的功能的；而“备机”一般被认为仅仅提供备份功能，不提供访问功能。所以使用“主从”还是“主备”，是要看场景的，这两个词并不是完全等同的。

读写分离的实现逻辑并不复杂，但有两个细节点将引入设计复杂度：主从复制延迟和分配机制。

复制延迟

以 MySQL 为例，主从复制延迟可能达到 1 秒，如果有大量数据同步，延迟 1 分钟也是有可能的。主从复制延迟会带来一个问题：如果业务服务器将数据写入到数据库主服务器后立刻（1 秒内）进行读取，此时读操作访问的是从机，主机还没有将数据复制过来，到从机读取数据是读不到最新数据的，业务上就可能出现问题。例如，用户刚注册完后立刻登录，业务服务器会提示他“你还没有注册”，而用户明明刚才已经注册成功了。

解决主从复制延迟有几种常见的方法：

1. 写操作后的读操作指定发给数据库主服务器

例如，注册账号完成后，登录时读取账号的读操作也发给数据库主服务器。这种方式和业务强绑定，对业务的侵入和影响较大，如果哪个新来的程序员不知道这样写代码，就会导致一个bug。

2. 读从机失败后再读一次主机

这就是通常所说的“二次读取”，二次读取和业务无绑定，只需要对底层数据库访问的 API 进行封装即可，实现代价较小，不足之处在于如果有很多二次读取，将大大增加主机的读操作压力。例如，黑客暴力破解账号，会导致大量的二次读取操作，主机可能顶不住读操作的压力从而崩溃。

3. 关键业务读写操作全部指向主机，非关键业务采用读写分离

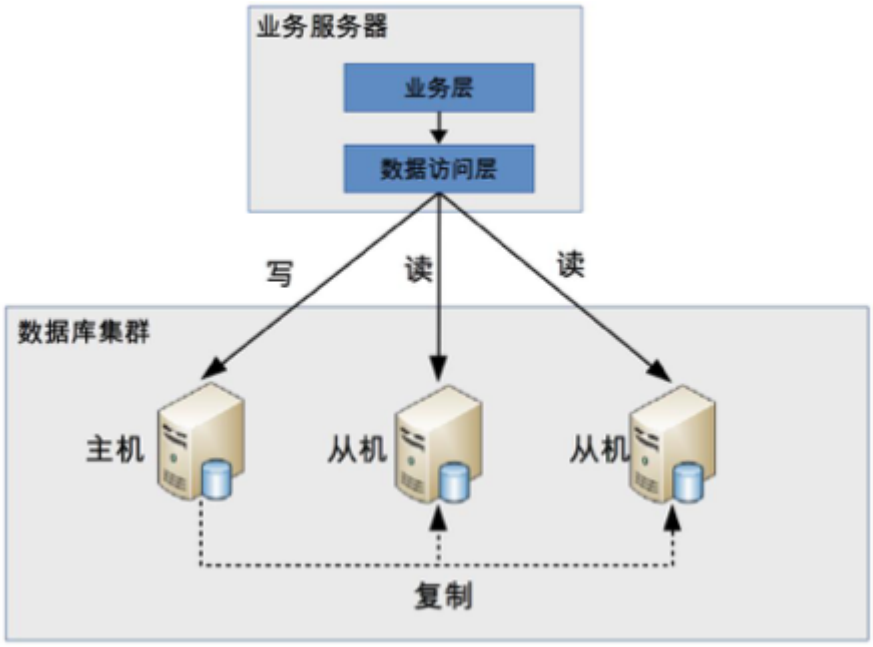
例如，对于一个用户管理系统来说，注册 + 登录的业务读写操作全部访问主机，用户的介绍、爱好、等级等业务，可以采用读写分离，因为即使用户改了自己的自我介绍，在查询时却看到了自我介绍还是旧的，业务影响与不能登录相比就小很多，还可以忍受。

分配机制

将读写操作区分开来，然后访问不同的数据库服务器，一般有两种方式：**程序代码封装**和**中间件封装**。

1. 程序代码封装

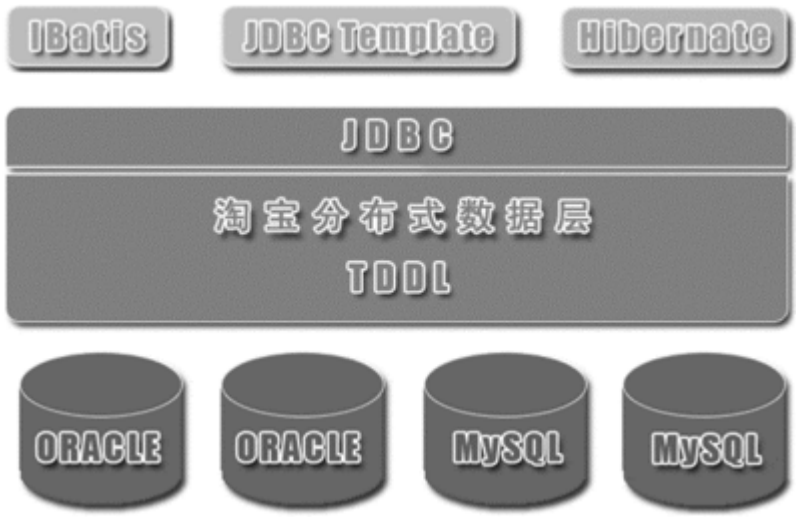
程序代码封装指在代码中抽象一个数据访问层（所以有的文章也称这种方式为“中间层封装”），实现读写操作分离和数据库服务器连接的管理。例如，基于 Hibernate 进行简单封装，就可以实现读写分离，基本架构是：



程序代码封装的方式具备几个特点：

- 实现简单，而且可以根据业务做较多定制化的功能。
- 每个编程语言都需要自己实现一次，无法通用，如果一个业务包含多个编程语言写的多个子系统，则重复开发的工作量比较大。
- 故障情况下，如果主从发生切换，则可能需要所有系统都修改配置并重启。

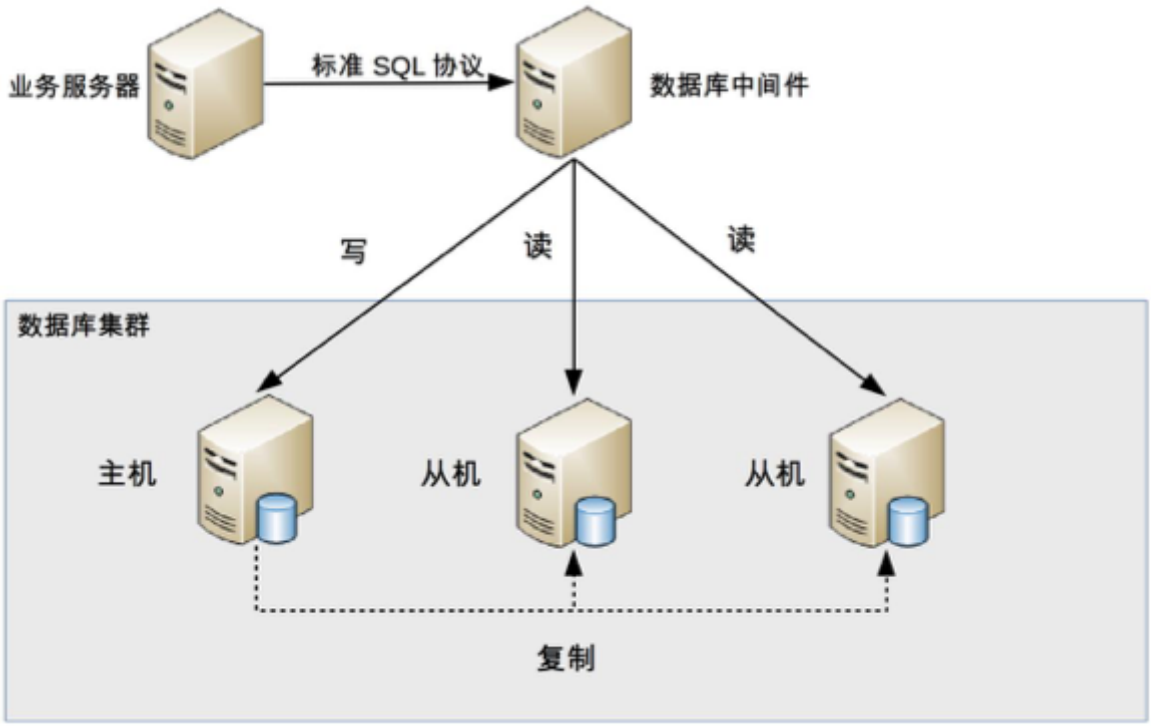
目前开源的实现方案中，淘宝的 TDDL（Taobao Distributed Data Layer，外号: 头都大了）是比较有名的。它是一个通用数据访问层，所有功能封装在 jar 包中提供给业务代码调用。其基本原理是一个基于集中式配置的 jdbc datasource 实现，具有主备、读写分离、动态数据库配置等功能，基本架构是：



（http://1.im.guokr.com/0Y5YjfjQ8eGOzeskpen2mlNIYA_b7DBLbGT0YHyUiLFZAgAAgwEAAFBO.png）

2. 中间件封装

中间件封装指的是独立一套系统出来，实现读写操作分离和数据库服务器连接的管理。中间件对业务服务器提供 SQL 兼容的协议，业务服务器无须自己进行读写分离。对于业务服务器来说，访问中间件和访问数据库没有区别，事实上在业务服务器看来，中间件就是一个数据库服务器。其基本架构是：

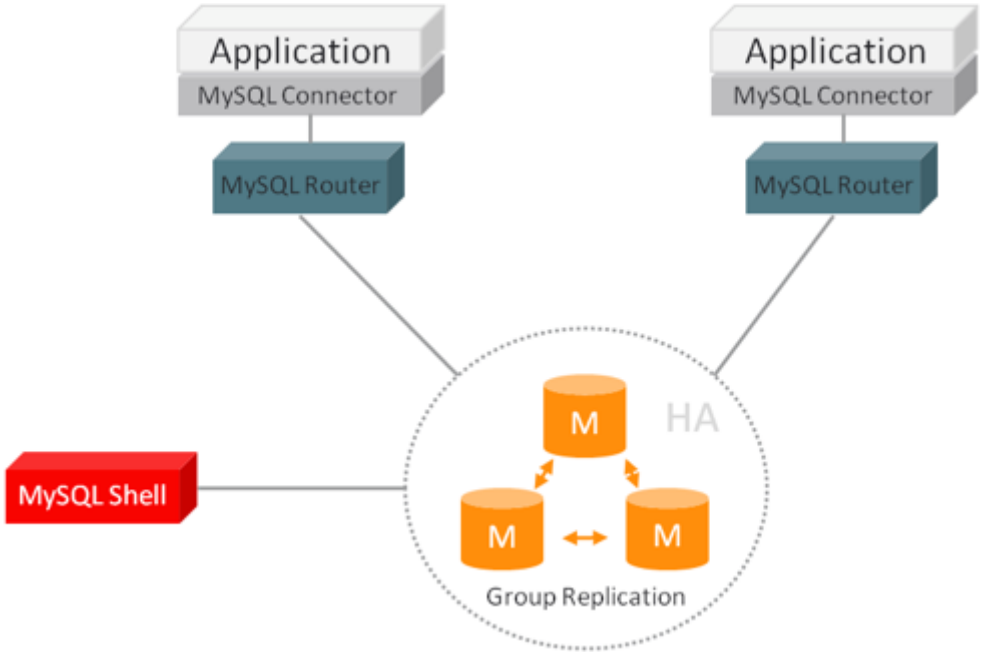


数据库中间件的方式具备的特点是：

- 能够支持多种编程语言，因为数据库中间件对业务服务器提供的是标准 SQL 接口。
- 数据库中间件要支持完整的 SQL 语法和数据库服务器的协议（例如，MySQL 客户端和服务器的连接协议），实现比较复杂，细节特别多，很容易出现 bug，需要较长的时间才能稳定。
- 数据库中间件自己不执行真正的读写操作，但所有的数据库操作请求都要经过中间件，中间件的性能要求也很高。
- 数据库主从切换对业务服务器无感知，数据库中间件可以探测数据库服务器的主从状态。例如，向某个测试表写入一条数据，成功的就是主机，失败的就是从机。

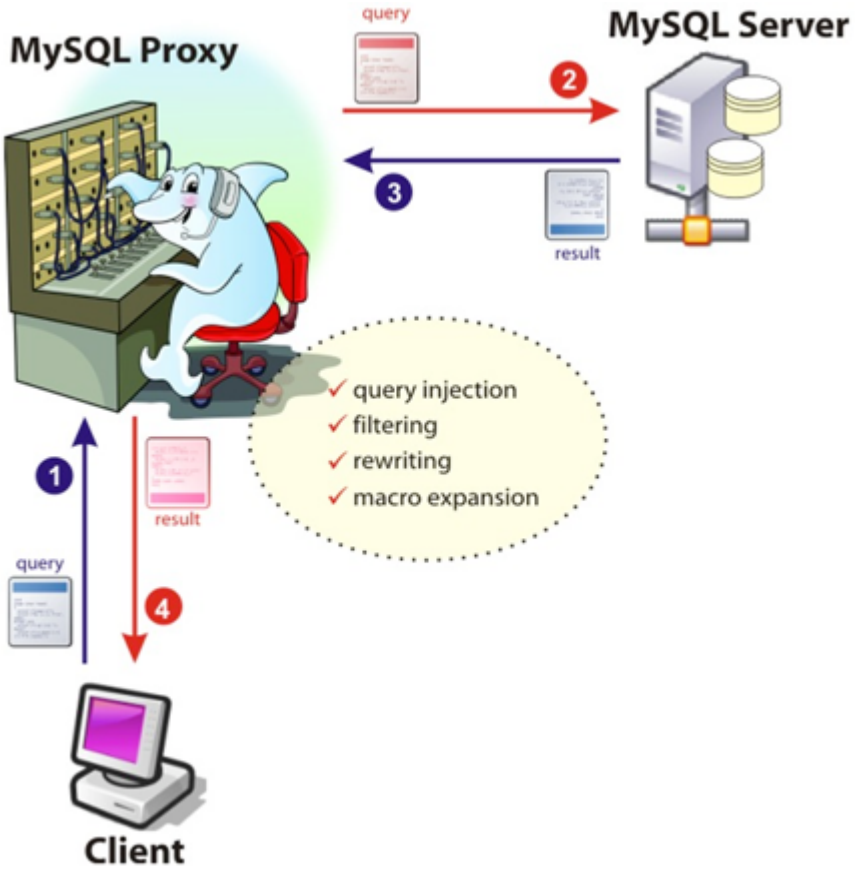
由于数据库中间件的复杂度要比程序代码封装高出一个数量级，一般情况下建议采用程序语言封装的方式，或者使用成熟的开源数据库中间件。如果是大公司，可以投入人力去实现数据库中间件，因为这个系统一旦做好，接入的业务系统越多，节省的程序开发投入就越多，价值也越大。

目前的开源数据库中间件方案中，MySQL 官方先是提供了 MySQL Proxy，但 MySQL Proxy 一直没有正式 GA，现在 MySQL 官方推荐 MySQL Router。MySQL Router 的主要功能有读写分离、故障自动切换、负载均衡、连接池等，其基本架构如下：



(https://dev.mysql.com/doc/mysql-router/2.1/en/images/mysql-router-positioning.png)

奇虎 360 公司也开源了自己的数据库中间件 Atlas，Atlas 是基于 MySQL Proxy 实现的，基本架构如下：



(https://camo.githubusercontent.com/42c01a1245183948ba8c61e5572d3aa9c3e8a08e/687474703a2f2f7777332e73696e61696d672e636e2f6c617267652f366535373035613556a7731656271353169336668716a32306a69306a6a7767392e6a7067)

以下是官方介绍，更多内容你可以参考这里。

> Atlas 是一个位于应用程序与 MySQL 之间中间件。在后端 DB 看来，Atlas 相当于连接它的客户端，在前端应用看来，Atlas 相当于一个 DB。Atlas 作为服务端与应用程序通信，它实现了 MySQL 的客户端和服务端协议，同时作为客户端与 MySQL 通信。它对应用程序屏蔽了 DB 的细节，同时为了降低 MySQL 负担，它还维护了连接池。

小结

今天我为你讲了读写分离方式的原理，以及两个设计复杂度：复制延迟和分配机制，希望对你有所帮助。

这就是今天的全部内容，留一道思考题给你吧，数据库读写分离一般应用于什么场景？能支撑多大的业务规模？

欢迎你把答案写到留言区，和我一起讨论。相信经过深度思考的回答，也会让你对知识的理解更加深刻。（编辑乱入：精彩的留言有机会获得丰厚福利哦！）

精选留言

海。 23

老师，您好

我个人的想法是可以加入缓存，例如注册后登录这种业务，可以在注册后加入数据库，并加入缓存，登录的时候先查缓存再查库表。

例如存入redis中并设置十分钟的过期时间。登录的时候先查redis，再查库表，如果redis中没有，说明就是过期的数据，这时候查从机就肯定存在了，希望能得到老师的点评，谢谢。

2018-05-30

作者回复

赞同，并不是说一有性能问题就上读写分离，而是应该先优化，例如优化慢查询，调整不合理的业务逻辑，引入缓存等，只有确定系统没有优化空间后，才考虑读写分离或者集群

2018-05-30

narry 9

个人感觉，读写分离适合读压力比写压力大很多的业务类型，最终的瓶颈应该是出现在承担写操作的主机上，最大规模和这台主机的iops等能力相关

2018-05-29

tangfengr 6

我认为读写分离适用单机并发无法支撑并且读的请求更多的情形。在单机数据库情况下，表上加索引一般对查询有优化作用却影响写入速度，读写分离后可以单独对读库进行优化，写库上减少索引，对读写的能力都有提升，且读的提升更多一些。
不适用的情况:1如果并发写入特别高，单机写入无法支撑，就不适合这种模式。
2 通过缓存技术或者程序优化能够满足要求

2018-07-19

作者回复

赞同👍

2018-07-20

侯海佳 6

我认为读写分离适合类似微博这种业务：读多写少

2018-05-29

wuxu 5

读写分离的前提是并发量大，单机已经不能处理该数量的并发请求了，想要解决问题就得做作拆分，于是有了读写分离，主库负责写，从库负责读，降低了同台机器并发请求，当读越来越多时，可扩充从库，写越来越多时，只好拆分业务或分库分表，如：注册功能，单独出来做一个注册的微服务，但还是会到达一个瓶颈，没做过，不知道能支持多少的并发？

2018-05-30

作者回复

要具体测试，不同业务复杂度不同

2018-05-30

彡工鸟 5

是否还应该加上一个，当单机写顶不住压力后，就可以做数据库拆分了，例如业务纵向拆分，连同数据库一起，就变成分布式服务，微服务了:)

2018-05-30

作者回复

说法没错，但具体实施的时候要注意，不要一有压力就上读者分离，因为很多时候其实是sql语句或者业务逻辑有问题，因此先优化，只有优化后也无法满足要求的时候才考虑读者分离或者集群

2018-05-30

Geek_8242cb 3

我们做网银系统，用redis存了一些不太重要的数据，比如数据字典信息，作为缓存。但是不太敢把用户权限，交易数据等重要信息存在缓存里，因为redis并不保证事务，我们担心一旦缓存服务器宕机或者失败会影响银行业务。所以缓存的作用也不是很大，还是把大部分读数据的压力放到了数据库上，您说我们这种担心有必要吗？如果单库后续扛不住压力，是否读写分离比加缓存更好一些？

2018-06-03

作者回复

交易型业务缓存应用不多，缓存一般总在查询类业务上，你们的担心有一定必要

2018-06-04

@漆~心endless 3

并非所有系统都需要进行读写分离，正如之前讲得架构三原则，其中根据“合适原则”的规定，先确认系统的业务量是否出现了数据库的性能问题。如果是，首先通过优化MySQL语句等，如果还是达不到要求的性能指标，则需进行读写分离。毕竟读写分离会引入一系列不可预知的问题，如数据不同步。

2018-05-31

合民 3

读写分离，主从架构，顾名思义，写不变，主要解决高性能读的问题，所以适用场景自然是读多写少的情况。比如类似于博客、中小型朋友圈，这种一般写数据库后基本不变，但是很多人会去访问，频繁的读。我认为如果缓存能支撑的话就没必要上读写分离，相对来说缓存更简单。

2018-05-31

刘岚乔月 3

请问 对于主从出现的数据同步延时问题 在实际生产落地 真的只有把重要的查询指向主吗 还有其他真正的落地方案吗

2018-05-29

作者回复

当然是真的呀，难道我还会骗你不成？😂

如果不想用这种方式，用缓存是可以规避这个问题的，但其实这时候的方案就不是读写分离了

2018-05-30

刘志刚 3

读写分离比较适用于类似消息记录，对于写和读业务的强实时性要求不到苛刻的地步的情况，而且做的时候这种跟业务量还是有比较大关系的，比如，业务量的订单量每年都不超过1

千万，整天去做分库分表倒不如好好优化下sql写法，如果订单量每天都超过好几百万，那这个必要性就很强了！

2018-05-29

作者回复

赞，先优化

2018-05-30

姜泮昌 3

读写分离适用于单服务器无法满足所有请求的场景，从请求类型的角度对服务器进行拆分，但这样在要求硬件资源能够支撑的同时，对代码实现也有更高的要求。

2018-05-29

作者回复

天下没有免费的午餐😀

2018-05-30

云学 2

相比于前面的几篇高大上文章，这篇更接地气

2018-05-29

haydenliu 2

老师，个人感觉，不是每个场景都需要读写分离，第一，对于那些及时性要求高的业务，是不合适的，第二，读多写少的业务，也没必要，反而增加了复杂度。不知道这样理解对不服?

2018-05-29

作者回复

1. 正确

2. 正好反过来了，读多写少就适合读写分离

2018-05-29

null 2

re: 写操作后的读操作指定发给数据库主服务器

后端无法知道本次请求是否为写操作之后的读，因此会依赖前端传递一个参数，如 target_db=master / slave，来决定目标数据库。
所以这种方式，需要在前后端代码实现相关逻辑，代码耦合较大。

这种解决方案是否只提供了一种思路，实际开发时很少使用这方案，不知道理解是否正确，谢谢！

2018-05-29

作者回复

是的，对代码逻辑有要求，

2018-05-29

richey 1

TDDL已经长期不更新了，老师怎么不提一下Mycat

2018-06-26

W_T 1

从数据库读写的角度分类，一共有四类：写多读多，写少读少，写多读少，写少读多。
写少读少的情况，不需要分离。
写多的情况，单个库写入会造成单点压力，分库写入，那其实就是分库分表了。
所以我认为，读写分离的设计适用于写少读多的情况。
至于业务量，读可以不断水平扩展，主要还是受写的限制。

2018-06-04

马广乐 1

加个缓存能解决写完立即读的场景吗，老师。

2018-05-29

作者回复

可以的，但那是另外一个方案了，很多场景就算用了缓存也要读写分离

2018-05-30

侯佳林 1

主服务器充当业务的写服务器、从服务器的读服务器，如果从服务器较多，单台写服务器的压力会很大

2018-05-29

Tom 1

读写分离一方面分离了读和写，另一方面读可通过多个从机再进一步分担读。
1.一主
一开始，读和写都在主机这条船上，随着读写越来越多，读和写相互影响，主机压力越来越大船快沉了。
2.一主一从
于是，读和写商量说分家吧，否则要抱团死了，写是数据源头留在主机，你读就自己找个从机新船吧；
便分成了一主一从，由主向从单向同步数据，有主推和从拉两种同步方式；读和写各自轻松了。
3.一主多从
随着业务开展，读写再次增长，特别是一般系统都是读的增长一般比写快很多，读一个从机也快沉船了，就多加了几个从主，变成了一主多从，数据由主机同步到所有从机，可以很方便地水平扩展。
4.多组
这块已经不算是读写分离了，要分离写了。
当写增长到一定程度，单个主机已经hold不住写了，要分库了。
按业务垂直拆分或水平拆分，拆分后的多库每库作为新的一组，每组是一个主从集群。

还有一种可能的过程是一主机直接到分库成多主机，某个主机读太多时进行读写分离。

能支撑多大规模就不了解了。
查了下mysql官网benchmarks(https://www.mysql.com/why-mysql/benchmarks/)，单机32users并发只读近40w+ qps，读写10w+ qps。
如果按这个数据，以32users算:
一主一从能支撑写10w+qps，读40w+;
一主三从能支撑写10w+qps，读120w+;

没压测过db，是不是太高了?
请指正，谢谢!

2018-05-29

作者回复

这个数据有误导性，要看具体的SQL语句和表结构，实际应用中一般不可能这么高

2018-05-30