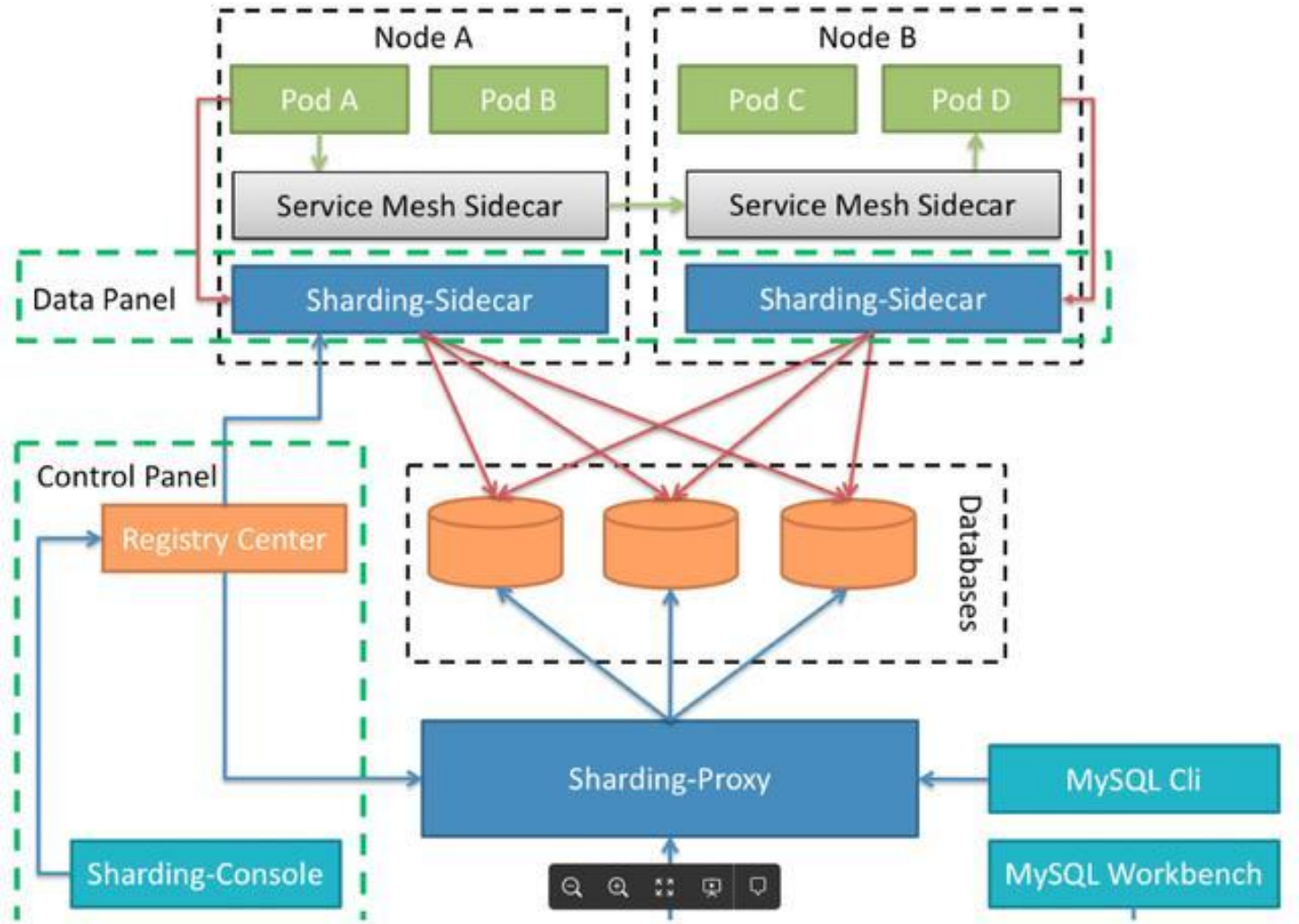


数据库中间件漫谈

阿丸
微信公众号：阿丸笔记

关注他

9 人赞同了该文章

1.前言

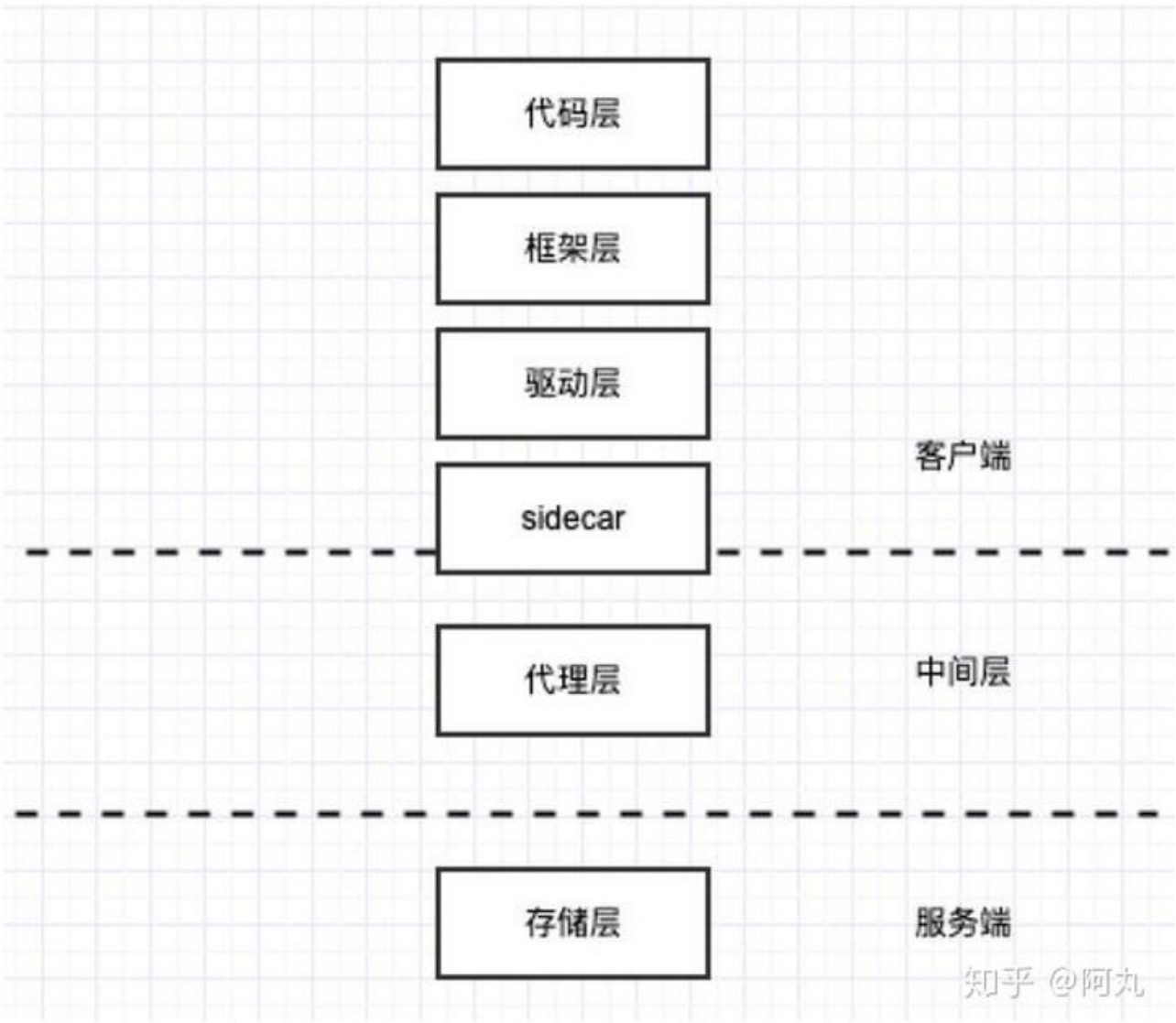
随着业务的发展，MySQL数据库中的表会越来越多，表中的数据量也会越来越大，相应地，数据操作的开销也会越来越大；另外，无论怎样升级硬件资源，单台服务器的资源（CPU、磁盘、内存、网络IO、事务数、连接数）总是有限的，最终数据库所能承载的数据量、数据处理能力都将遭遇瓶颈。

分表、分库和读写分离可以有效地减小单台数据库的压力。

本文主要针对业界主流的数据库中间件的实现、功能、成本等方面进行对比，总结数据库中间件的实现方式，并展望未来的可能发展。

2. 实现方式

一般来说，对于数据库中间件，可以在以下六个层次做切入。



2.1 代码层

在同一个项目中创建多个数据源，采用if else的方式，直接根据条件在代码中路由。

Spring中有动态切换数据源的抽象类，具体参见AbstractRoutingDataSource。

如果项目不是很庞大，使用这种方式能够快速的进行分库。但缺点也是显而易见的，这种海量的代码侵入是绝不能被接受的。

而且当查询结果返回时，需要对跨库、聚合等查询结果进行归并，开发工作量非常巨大。

这种方式了解一下即可，一般不会去使用。

2.2 框架层

主要是修改或增强现有ORM框架的功能，在SQL中增加一些自定义原语或者hint来实现。

常见的比如实现一些拦截器（比如Mybatis的Interceptor接口），增加一些自定义解析来控制数据的流向，效果虽然较好，但会改变一些现有的编程经验。

这种情况适合公司ORM框架统一的情况，但在很多情况下不太现实。

而且大部分情况下要修改框架源码，因此，也不推荐。

2.3 驱动层

无论是从代码层还是框架层做处理，都是高侵入、难维护的。

因此，常见的数据库中间件，至少需要从驱动层开始，我们可以理解为一个smart-client。

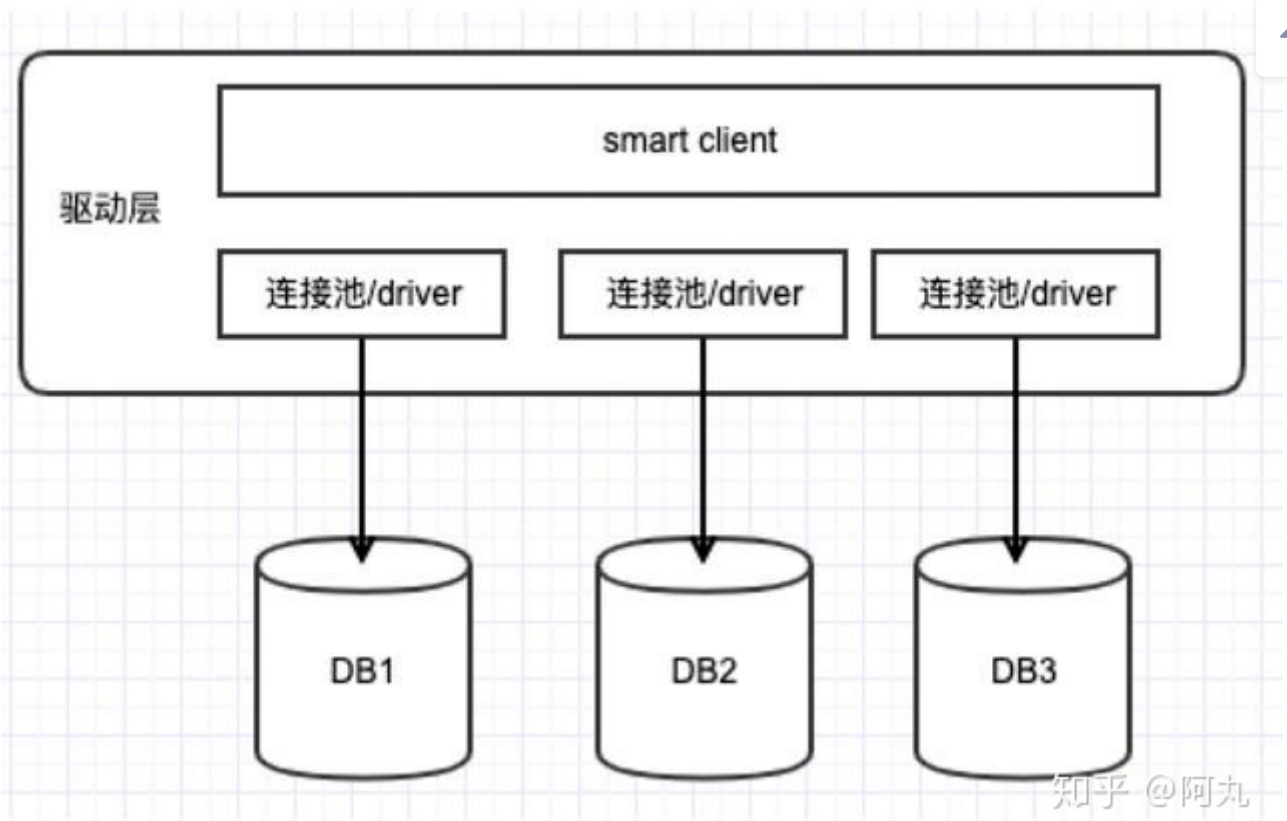
什么意思是smart-client呢?

通常smart-client是在连接池或者driver的基础上进行了一层封装。

这个smart-client内部可以与不同的数据库建立连接。

服务需要查询的sql，就交给smart-client进行解析、优化，然后发送给具体的数据库进行操作。

例如在读写分离情况下，smart-client会选择sql走从库还是主库；在分库分表的情况下，进行sql解析、sql改写等操作，然后路由到不同的分库，将得到的结果进行合并，返回给应用。



我们熟知的TDDL、Sharding-JDBC等，都是在此层切入。

优点：

1）实现方便，业务无入侵。smart-client不需要实现客户端通信协议，只需要在数据数据库厂商提供的不同语言的数据库驱动上做封装即可。例如mysql针对java语言提供了mysql-connector-java驱动，针对python提供了mysql-connector-python驱动。

2）天然去中心化。smart-client以sdk的方式被应用引入，然后部署到不同的服务节点上，不需要有代理层proxy。因此相较于代理方式而言，不需要考虑高可用的问题。只要应用的节点没有全部宕机，就可以访问数据库。(这里的高可用是相比代理层proxy而言，数据库本身的高可用还是需要保证的)

缺点：

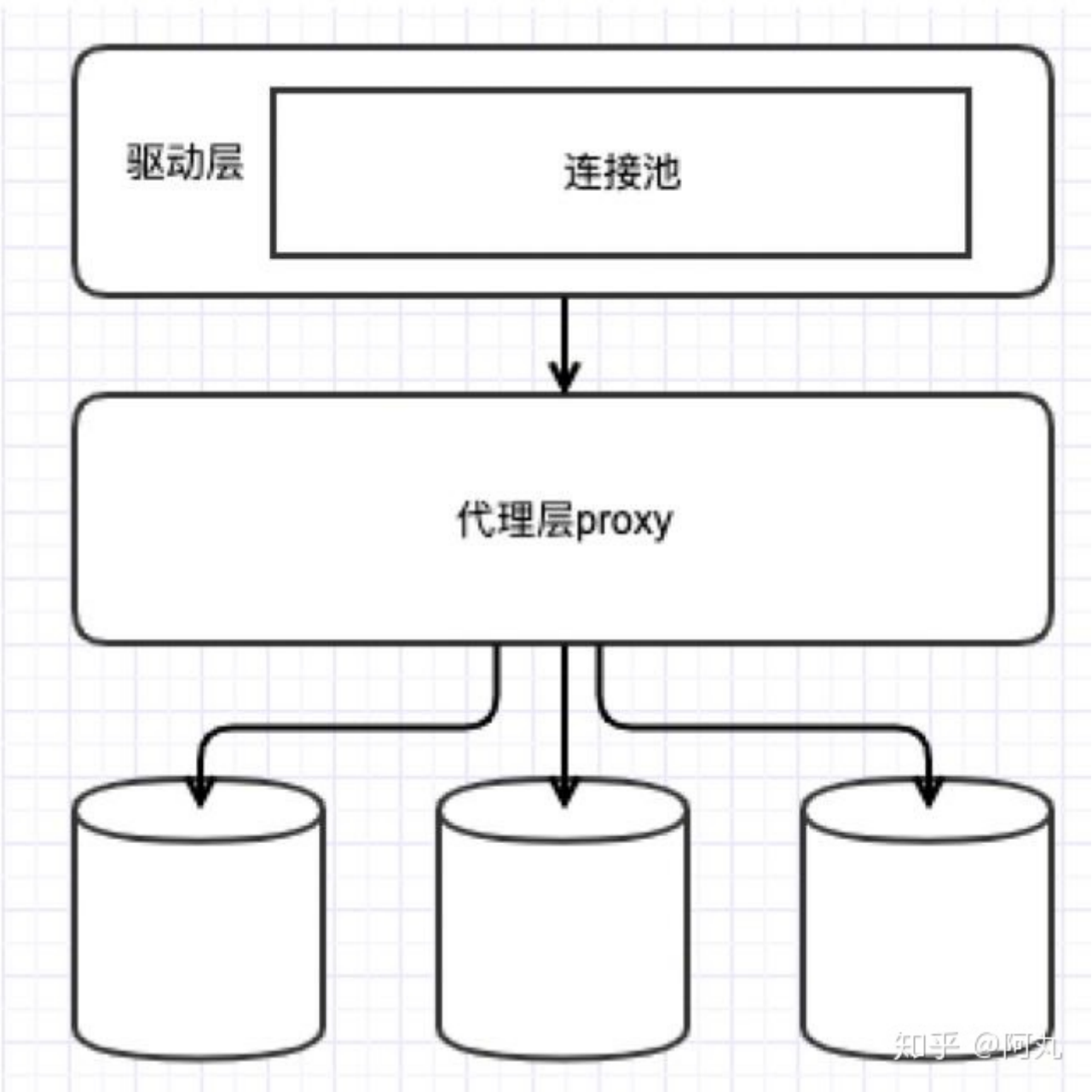
1）通常仅支持某一种语言。例如tddl、zebra、sharding-jdbc都是使用java语言开发，因此对于使用其他语言的用户，就无法使用这些中间件。如果其他语言要使用，那么就要开发多语言客户端。

2）版本升级困难。因为应用使用数据源代理就是引入一个jar包的依赖，在有多个应用都对某个版本的jar包产生依赖时，一旦这个版本有bug，所有的应用都需要升级。而数据库代理升级则相对容易，因为服务是单独部署的，只要升级这个代理服务器，所有连接到这个代理的应用自然也就相当于都升级了。

3）去中心化的缺点，比如无法做全局的sql限流

2.4 代理层

在应用中，我们通过一个普通的数据源(c3p0、druid、dbcp等)与代理服务器建立连接，所有的sql操作语句都是发送给这个代理，由这个代理去操作底层数据库，得到结果并返回给应用。在这种方案下，分库分表和读写分离的逻辑对开发人员是完全透明的。



像MySQL Router、MyCat、ShardingSphere（proxy模式）等，都是在此层切入。

优点：

1）多语言支持。也就是说，不论你用的php、java或是其他语言，都可以支持。以mysql数据库为例，如果proxy本身实现了mysql的通信协议，那么你可以就将其看成一个mysql 服务器，因此不同

语言的开发者都可以使用mysql官方提供的对应的驱动来与这个代理服务器建通信。

2）对业务开发同学透明。由于可以把proxy当成mysql服务器，理论上业务同学不需要进行太多代码改造，既可以完成接入。

缺点：

1）实现复杂。因为proxy需要实现被代理的数据库server端的通信协议，实现难度较大。

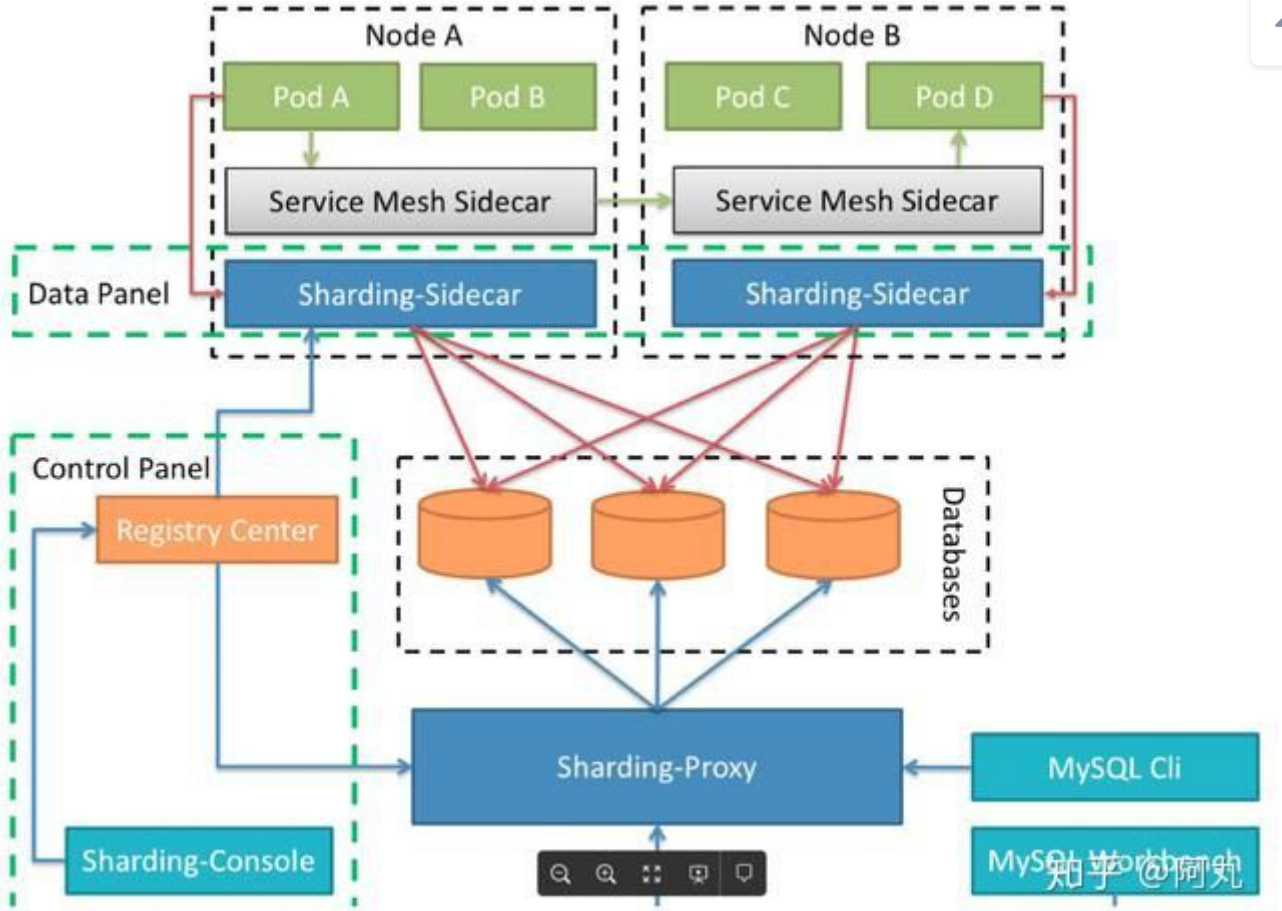
2）proxy本身需要保证高可用。由于应用本来是直接访问数据库，现在改成了访问proxy，意味着proxy必须保证高可用。否则，数据库没有宕机，proxy挂了，导致数据库无法正常访问，就尴尬了。

3）租户隔离。可能有多个应用访问proxy代理的底层数据库，必然会对proxy自身的内存、网络、cpu等产生资源竞争，proxy需要需要具备隔离的能力。

2.5 Sidecar

Sharding-Sidecar是ShardingSphere的第三个产品，目前仍然在规划中。 定位为Kubernetes或Mesos的云原生数据库代理，以DaemonSet的形式代理所有对数据库的访问。

通过无中心、零侵入的方案提供与数据库交互的的啮合层，即Database Mesh，又可称数据网格。Database Mesh的关注重点在于如何将分布式的数据访问应用与数据库有机串联起来，它更加关注的是交互，是将杂乱无章的应用与数据库之间的交互有效的梳理。使用Database Mesh，访问数据库的应用和数据库终将形成一个巨大的网格体系，应用和数据库只需在网格体系中对号入座即可，它们都是被啮合层所治理的对象。



优点：

分布式云原生的数据库中间件模式，集成了jdbc和proxy各自的优点，能满足高可用、跨语言、无感知升级等多种优势特性

缺点：

需要整体架构支持云原生体系

目前还没正式上线。

2.6 存储层

这个层次实际上不应该叫数据库中间件了，需要更换存储。

比如Aurora、polardb、tidb等分布式数据库，通过计算节点和存储节点分离，计算节点scale up，存储节点scale out的理念将公有云的关系数据库产品推向了一个新的高度。

这样一来，实际上已经不再需要传统的数据库中间件了，一切问题天然就不存在了。

3. 功能对比

从上文可以了解到，目前最主流的数据库中间件主要是从驱动层smart-client和代理层proxy切的。

下面，我们来了解下业界主流的中间件产品在这两个层次的站队情况与实现的功能对比。

实现方式	中间件产品
smart-client	tddl（阿里，未开源）、zebra（美团）、sharding-JDBC(shardingsphere的子项目)、
proxy	cobar(阿里开源)、mycat（社区项目，cobar改良版）、drds(阿里云产品)、sharding-proxy(shardingsphere的子项目)、Gaea（小米开源）

其他还有比如：

Atlas、Kingshard、DBProxy、mysql router、MaxScale、58 Oceanus、ArkProxy、Ctrip DAL、Tsharding、Youtube vitess、网易DDB、Heisenberg、proxysql、Mango、DDAL、Datahekr、MTAtlas、

我们可以看到，基本各个大厂都撸过一遍自己的中间件产品。不过目前开源而且比较火的已经不多了，主要还是以shardingsphere为主。

我们从功能维度，来对比一下几个产品。

	TDDL	sharding-jdbc	zebra	Gaea
分库分表	•	•	•	•
读写分离	•	•	•	•
分布式事务		•		
强制路由	•	•		
读写流量控制	•			
按sql限流			•	
数据脱敏		•		
熔断、禁用实例	•	•		

4. 展望

从上文的分析可以看出，尽管目前主流的数据库中间件还是在smart-client和proxy两个层面进行处理的，但是，已经能看到未来的方向了。云原生的到来，估计会做进一步的降维打击。

一方面是作为sidecar的模式，可能会有一个新的阶段，比如shardingsphere推出sidecar模式后。

而另一方面，云数据库通过全新的计算存储分离的架构方式，打破传统关系型数据库的性能瓶颈，传统数据库中间件将不再需要关注，一切都将以数据库基础设施的形式提供给使用者。

> 都看到最后了，原创不易，点个关注，点个赞吧～
> 知识碎片重新梳理，构建Java知识图谱：github.com/saigu/JavaKn...（历史文章查阅非常方便）
> 扫码关注我的公众号“阿丸笔记”，第一时间获取最新更新。同时能免费获取海量Java技术栈电子书、各个大厂面试题哦。

发布于 2020-03-14

数据库　数据库技术　中间件

文章被以下专栏收录

阿丸笔记　关注专栏

还没有评论

写下你的评论...