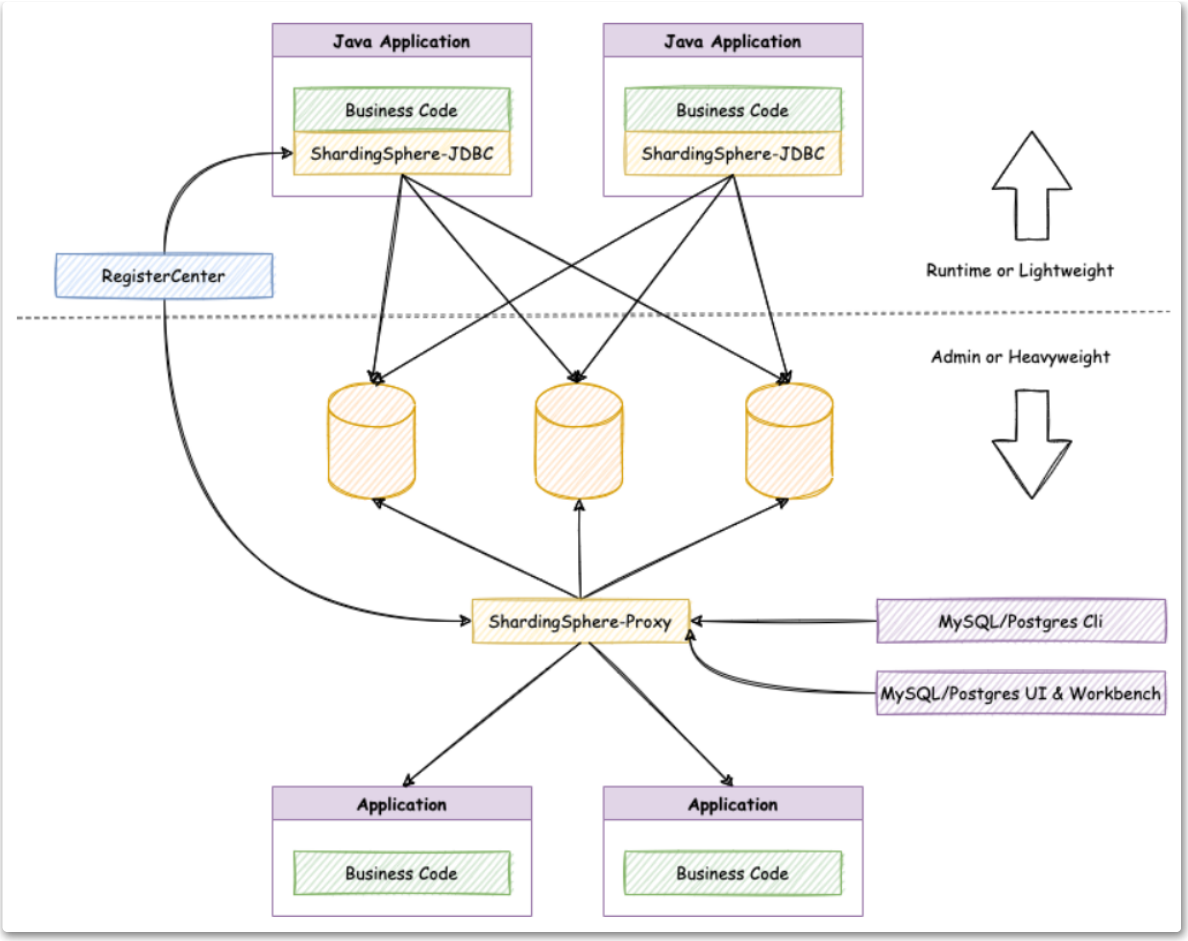
一、整体理解新版本

ShardingSphere在2021年十月份推出了5.0的第一个发布版本，并在2022年一月份推出了5.1版本。从整体来看，ShardingSphere5.x将自己的功能定位从数据库中间件升级到了DataBase Plus，数据库功能增强。核心产品定位的变化，必然会带来非常多的改变。不过从功能方面来看，目前5.X版本还只是做了一些功能增强，但是核心功能并没有太大的变动。很多规划的重大特性，目前也都没有在开源版本中支持。例如对于分布式数据库主要性能指标的可视化方案，以及对应的监控告警方案等。所以对于新版本，我们在学习阶段，没有必要去一一深究，可以先了解一下他的未来走向，如果出现了自己感兴趣的新特性，以后再持续跟踪。

> 由于目前新版本功能蓝图还有很多在设计阶段，并且现在用新版本的企业也都比较少。所以，以下的一些分享仅仅代表我个人目前的一些观点。

在整体功能架构上，ShardingSphere 5.X提出了一个混合架构，可以将ShardingJDBC和ShardingProxy两个产品更紧密的结合在一起。



图中，箭头往上的 Runtime or Lightweight部分，主要是针对程序员的部分，而箭头往下的 Admin or Heavyweight部分，主要是针对运维的部分。在这个规划当中，通过图中的注册中心，将以往比较割裂的ShardingJDBC和ShardingProxy结合到一起，这样可以让架构师能够更加自由地调整适合于当前业务的最佳系统架构。在我看来，这其中更大的意义在于ShardingJDBC和ShardingProxy的功能对齐，减少技术的复杂度对运维工作的影响。从而让ShardingSphere生态环境能够运行得更稳定。

关于注册中心在ShardingSphere中的作用，之前ShardingProxy的课程中已经做了一部分演示。而ShardingJDBC在4.X版本中还不支持注册中心。在5.X版本中，已经增加了ShardingJDBC的注册中心支持。这样，ShardingJDBC和ShardingProxy就可以基于注册中心直接同步复杂的分库分表配置。

不过关于注册中心，目前来看，与其说是架构调整，不如说是易用性的增强。通过注册中心，提供了一种可能，我们可以使用ShardingJDBC进行分库分表的功能开发，然后基于注册中心可以快速部署一个具有相同分库分表配置的ShardingProxy来进行数据库层的测试。至于说真的将两个产品共同用于支撑业务架构，还有比较多的事情要做。例如大家可以思考一下，自定义的一些分库分表的算法，要在ShardingJDBC和ShardingProxy之间同步，还是少不了开发和运维的共同参与。所

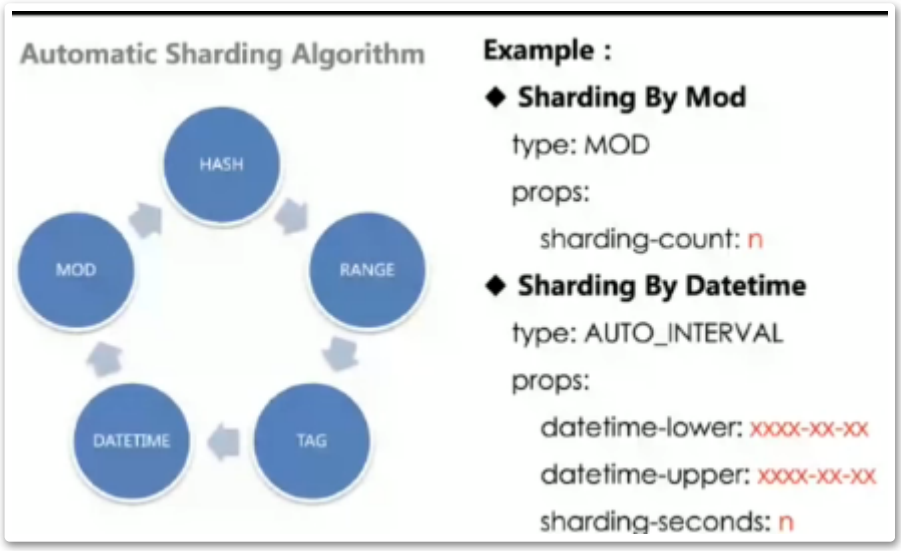
以，在这种模式下，技术的复杂度依然还是会扩散到运维层面，这对于项目的稳定运行，依然是一个不小的风险。

二、5.X部分新特性

5.X版本是ShardingSphere非常重视的一个大的架构升级版本，5.X版本从功能构思到发布测试版本，再到最终发布成熟版本，中间经过了超过十个月的时间，这对于开源项目，是一个非常重大的资源投入了。所以，5.X的新特性还是挺多的，这里我们就不一一分享了，主要说说三个跟开发和使用比较相关的几个大的特性。

1、DistSQL

4.X版本中，ShardingSphere支持了四种灵活的可定制的分片算法，但是这些分片算法，对于开发挺友好，对于运维则不太友好。毕竟你或许可以强行向运维同学解释一个inline表达式是什么意思，但是你基本不可能让一个运维同学去理解你写的Hint分片算法的Java代码是什么意思。所以，在5.X版本中，ShardingSphere提出了一些定制化的分片算法。

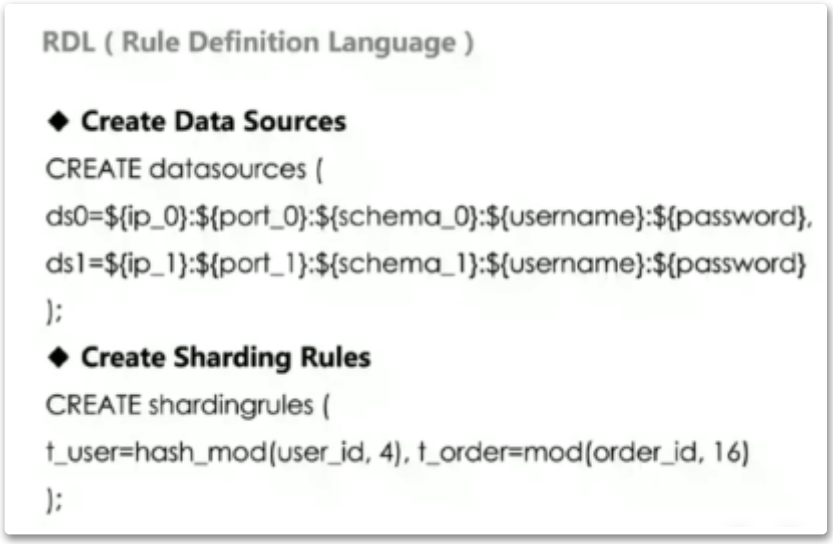


例如，对于常用的取模算法，我们通过添加一个MOD的分片type，以及该type对应的分片个数这么一个参数，就足够定义一个取模分片的算法逻辑，相比于inline表达式，这显然更利于运维同学理解。而图中下面对于按时间范围分片的规则定义方式，也显然会更为清晰。未来基于这种type的方式进行分片规则的封装，就能够很容易的实现企业内共享的分库分表规则库。

而在此基础上，ShardingSphere还扩展出了一组完整的规则定义、描述、维护的语言，称为DistSQL。DistSQL并没有对功能进行增强，不过可以在ShardingSphere运行过程当中，以SQL的方式来动态维护ShardingSphere中的规则定义，而不必在应用启动之初就全部维护好所有的规则定义。这可以使ShardingSphere用起来更像一个独立的数据库，而不是依附在实际数据库之上的一个数据库中间件。不过，目前对于DistSQL，还只在ShardingProxy当中支持。

DistSQL由三个部分组成，RDL(Resource & Rule Definition Language 规则定义语句)，RQL(Resource & Rule Query Language 规则查询语句)和RAL(Resource & Rule Administration Language 规则管理语句)。

其中RDL是用来定义资源和资源的语法，包括我们之前在开发过程中自己定义的数据源、分表规则、分库规则、广播规则等。



而RQL就是用来查询资源和规则定于的语法。例如：

```
SHOW SHARDING TABLE RULES; --查看定义的所有表的分片规则
SHOW SHARDING TABLE RULE tableName; --查看表tanleName的分片规则
SHOW SHARDING ALGORITHMS; --查看定义的所有的分片算法
SHOW UNUSED SHARDING ALGORITHMS; --查看未使用的分片算法
SHOW SHARDING KEY GENERATORS; --查看定义的主键生成器
SHOW DEFAULT SHARDING STRATEGY ; --查看默认的分片策略
```

RAL则跟之前举的RDL通过Type定义分片规则类似，对于一些常见的路由规则、事务类型、分片执行计划等进行固化，然后通过RAL来配置相关的属性。例如hint强制路由，readwrite_splitting读写分离等

语句	说明	示例
set readwrite_splitting	针对当前连接，设置读写分离的路	set readwrite_splitting

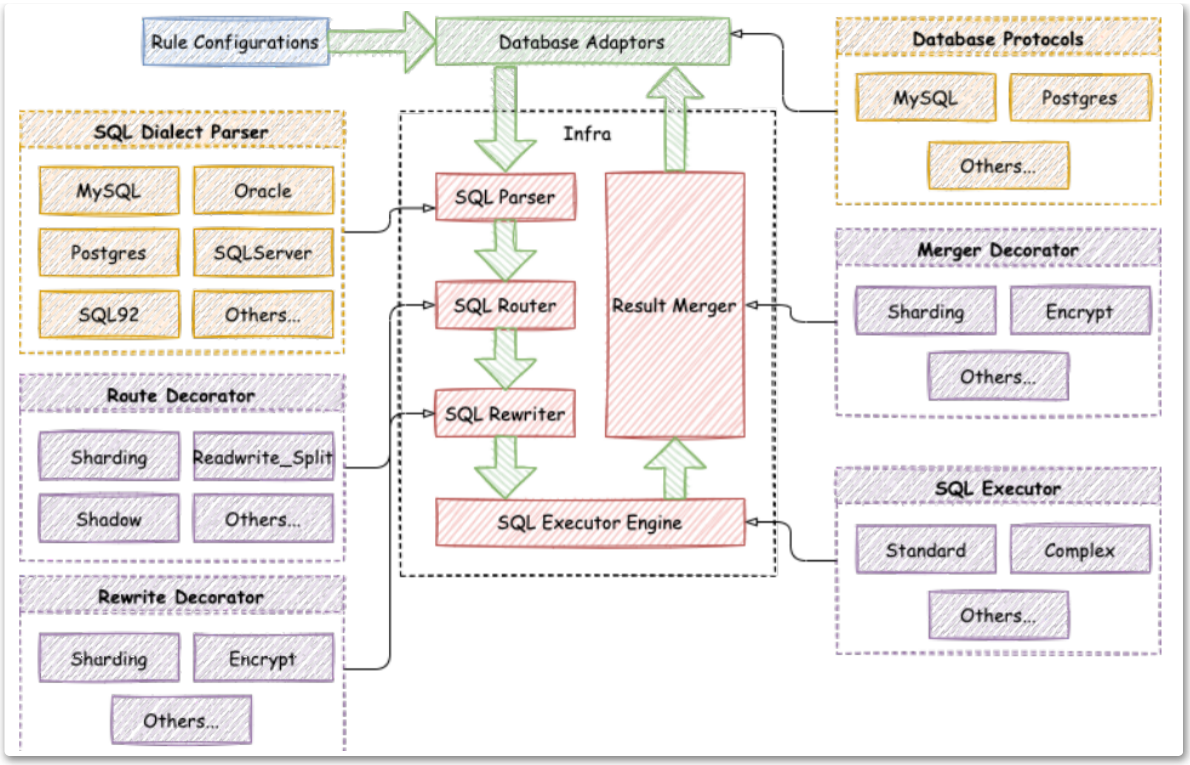
语句	说明	示例
hint source=[auto/write]	由策略(自动路由或强制到写库)	hint source=write
set sharding hint database_value=yy	针对当前连接，设置hint仅对数据库分片有效，并设定分片值，yy:数据库分片值	set sharding hintdatabase_value = 100
add sharding hint database_value xx = yy	针对当前连接，为表 xx 添加分片值 yy，xx：逻辑表名称，yy：数据库分片值	add sharding hintdatabase_value t_order=100
add sharding hint ta-ble_value xx = yy	针对当前连接，为表 xx 添加分片值 yy，xx：逻辑表名称，yy：表分片值	add sharding hint ta-ble_value t_order =100
clear hint	针对当前连接，清除 hint 所有设置	clear hint
[enable / disable] readwrite_splitting read xxx[from schema]	启用 / 禁用读库	enable r eadwrite_splitting readresource_0
[enable / disable] instance [IP=xxx, PORT=xxx/ instanceId]	启 用 / 禁 用proxy 实例	disable instance 127.0.0.1@3307
show instance list	查询 proxy 实例信息	show instance list

这里就不一一列举了，等后面新版本成熟了，大家可以再去详细了解。而且，个人觉得，这些功能对开发并没有太多帮助。

> 很多同学翘首期盼的实际表动态建表功能并没有排上日程。

2、可插拔内核

在5.X版本中，还提出了一个可拔插内核。目前来看，更像一个程序员理想化的重构。

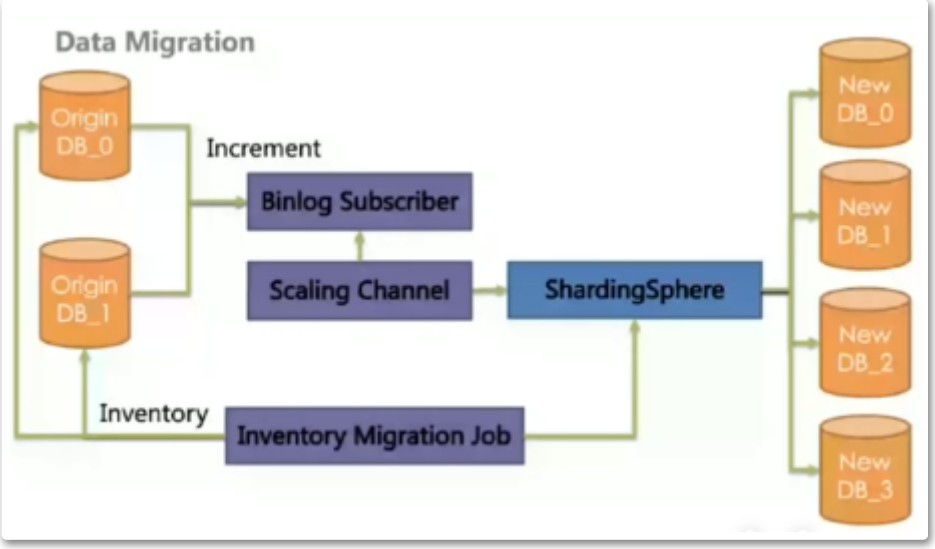


所谓可拔插架构，就是希望形成与具体业务无关的内核。图中红色的部分，就是分库分表的核心流程，跟4.x版本，其实变化并不大，就少了一个SQL改写的功能，这个一并放到了SQL解析中。而可拔插内核就是希望红色这一块功能与任何具体的业务无关，包括配置的规则。然后，让所有的功能组件都以插件的方式集成到内核当中。而内核并不区分哪些是ShardingSphere社区开发的功能，哪些是开发者自己扩扎的功能。这样，ShardingSphere社区成了一个跟所有其他开发者一样的插件提供者，从而让开发者可以更自由的构建自己的分库分表应用。比如说，对于SQL解析器，现在ShardingSphere实现了对于MySQL和PostGreSQL的支持，那以后只要按照内核中SQL解析器的要求，开发一个插件，就可以支持Oracle，支持SQLServer，甚至于支持Rocksdb这样的非关系型数据库。而对于执行引擎中的分布式事务问题，ShardingSphere已经集成了Seata和XA两种分布式事务引擎，未来还会规划实现一种全新的事务引擎，更好的支持分库分表这一特定的应用场景。

这些内核插件的实现方式，也还是使用SPI机制进行梳理，这跟4.x版本的开发体系是一样的。所以，目前看来，这更像是对已有代码的一次梳理重构，而对于内核，既然绑定了分库分表这样的业务场景，又怎么可能真正做到与具体业务无关呢？这中间要如何进行平衡和取舍，只能等待未来再观察了。

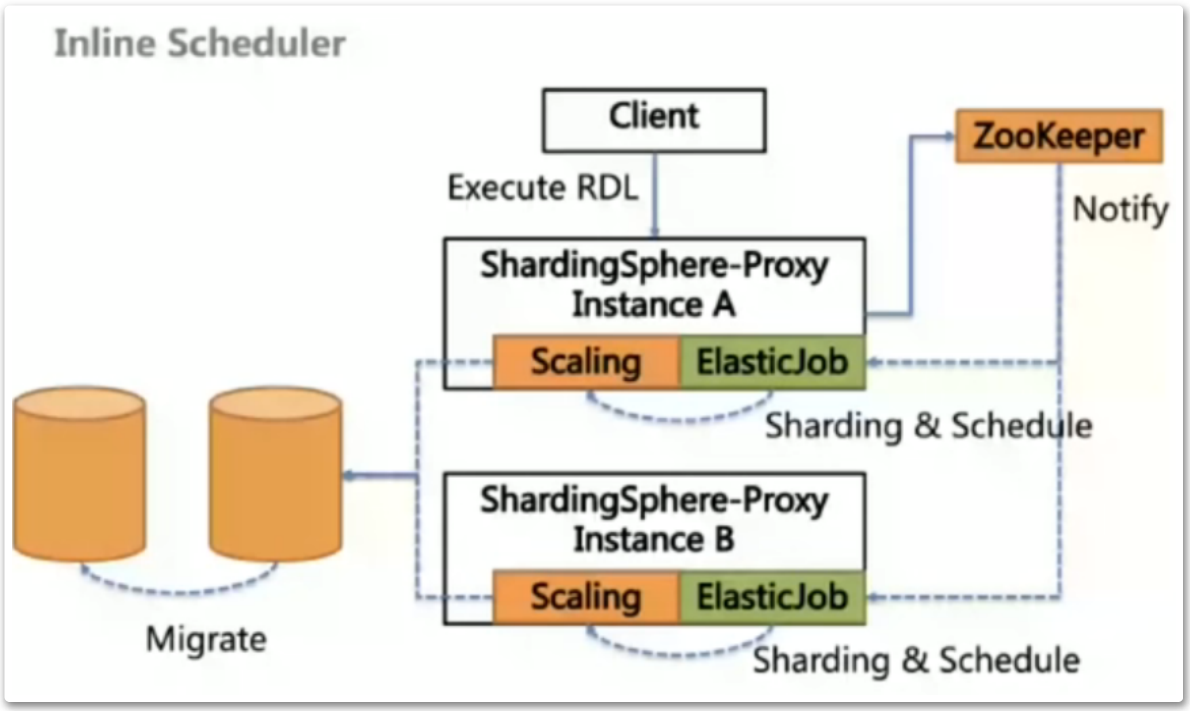
3、数据迁移

ShardingSphere的下一个目标就是实现弹性数据迁移，从而支持分片策略的灵活变更。



对于分库分表来说，数据迁移一直都是一个很头疼的事情，例如采用取模分片时，随着数据量越来越大，可能需要将原来两个分片的数据要扩展到四个分片，这就必然要对已有数据进行迁移。这是一个很重的操作，即要对大量的老数据进行全量迁移，又要对新产生的数据进行重新分片，同时还要尽量不影响数据的完整性，其中的难度可想而知。所以，很多技术团队使用分库分表都会极力避免数据迁移。但是，如果不迁移，分库分表规则无法更新，那就无法适应更大的数据量，这又会严重影响ShardingSphere产品的灵活性。所以，ShardingSphere在新版本中，也试图在数据迁移方面提供指导。

ShardingSphere 5.X版本会规划使用Scalling组件以及ElasticJob(ShardingSphere的任务调度子项目)形成一系列标准的数据迁移指导方案。



比如对于存量数据，使用ElasticJob对迁移任务进行切分，以分布式的方式进行大量存量数据的迁移。而对于新增的数据，通过注册中心来感知ShardingProxy上的数据变化，及时更新到新的数据分片当中。

三、全部内容总结

在张亮的技术分享过程中，我们看到，ShardingSphere这样一个产品，将不再仅仅作为一个Sharding分库分表的解决方案，而会更多关注Sphere生态的发展。比如张亮对于ShardingSphere弹性扩缩容设计的分享，加上规划中的SideCar功能组件，你是不是会想到，这就是在为了未来数据库上云做准备？虽然数据库上云这事，在目前看来是挺不靠谱的，但是MVC架构、微服务架构、消息驱动、大数据、云原生等等，哪种技术场景不是从不可能，不靠谱发展起来的呢？

分库分表的思想其实很简单，就是当数据量大了之后，化整为零，进行分布式存储。但是具体实施时，又是非常自由的，有非常多的思路和实现方式，而像ShardingSphere、MyCat等这样的组件，其实也只是一种算是经过验证后，比较靠谱的实现方式而已。其实同样的事情，很多的组件，像ES、Hadoop、Clickhouse等等，也都在做。在任何一个特定的项目中，对解决方案进行取舍的过程其实也是对分布式存储这事的不同思考。因此，在学习这些组件的过程中，同学们应该更注重于对于分布式存储发现的问题以及解决思路进行总结和思考，而不要一味的关注于具体产品的实现过程。毕竟，这些实现方式，临时查查就会，但是理解问题，解决问题的能力，可不是一下子就能学会的。让工具真正成为工具，而不是限制。

最终，我们将对ShardingSphere的全部内容总结成一句话就是：分库分表，能不分就不分。 当然这句话的意思并不是敷衍，而是工程化的总结。分库分表不像微服务、MQ等技术，有个大概的了解，百度一下，马上就可以上手了。有问题，大不了以后再做架构调整。而分库分表由于与大量的数据是绑定的，所以每一步都需要格外小心。如果没有考虑清楚就贸然分库分表，那么数据层的复杂逻辑就必然要蔓延到应用层，这在架构层面来说是非常不科学的，会给业务和数据带来强耦合。而这种耦合在日后项目演进过程中，会给整个项目带来非常多的困难。所以，与其说是能不分就不分，还不如换做另一句老话， 一不做，二不休 更妥当。

> 有道云笔记链接地址

> 文档：VIP06-ShardingSphere5.x新版本特性.md
> 链接：http://note.youdao.com/noteshare?id=0bca84d5d7262266bcb6f73280531036&sub=D12D866761E447CD83F66D82563EF6DC