09 | 可复用架构案例（二）：如何对现有系统做微服务改造？

2020-03-11 王庆友

架构实战案例解析

进入课程 >

讲述：王庆友

时长 17:03　大小 15.62M

你好，我是王庆友。在上一讲中，我以订单服务为例，和你一起讨论了如何从头开始，设计一个共享服务。今天我们再来聊一聊：**如何对现有系统做微服务化改造**。

很多早期的互联网公司都有巨大的单体应用，底层的数据表集中放在一个数据库里，这些表加起来可能有几百张。对于这样的应用系统和数据库，我们往往需要对它们进行拆分，通过微服务化改造，保证系统能够不断地扩展和复用。

相比从头开始落地服务，对现有系统做微服务化改造，这会面临更多的挑战。

首先，应用和数据表紧密耦合在一起，代码模块和表是多对多的依赖关系。一个模块会访问多张表，多个模块也会对同一张表进行访问，而且由于表都在一个数据库里，开发人员往往

会随意对表做关联，有时候甚至 Join 5~6 张表以上。这样，代码模块和表之间的关系是剪不断，理还乱，我们很难清晰地划分代码和数据表的边界，也就很难把它们封装成独立的微服务。

还有，系统现在已经在运行了，我们的改造不能影响业务的稳定性。那微服务落地后，现有的系统要怎么对接微服务，数据要怎么迁移，才能保证系统的平滑过渡呢？

所以，要想应对这些挑战，一方面，我们要保证比较合理的服务设计，才能达到优化系统架构的目的；另一方面，我们要做到整个过程对现有系统的影响比较小，才能达到系统改造顺利落地的目的。

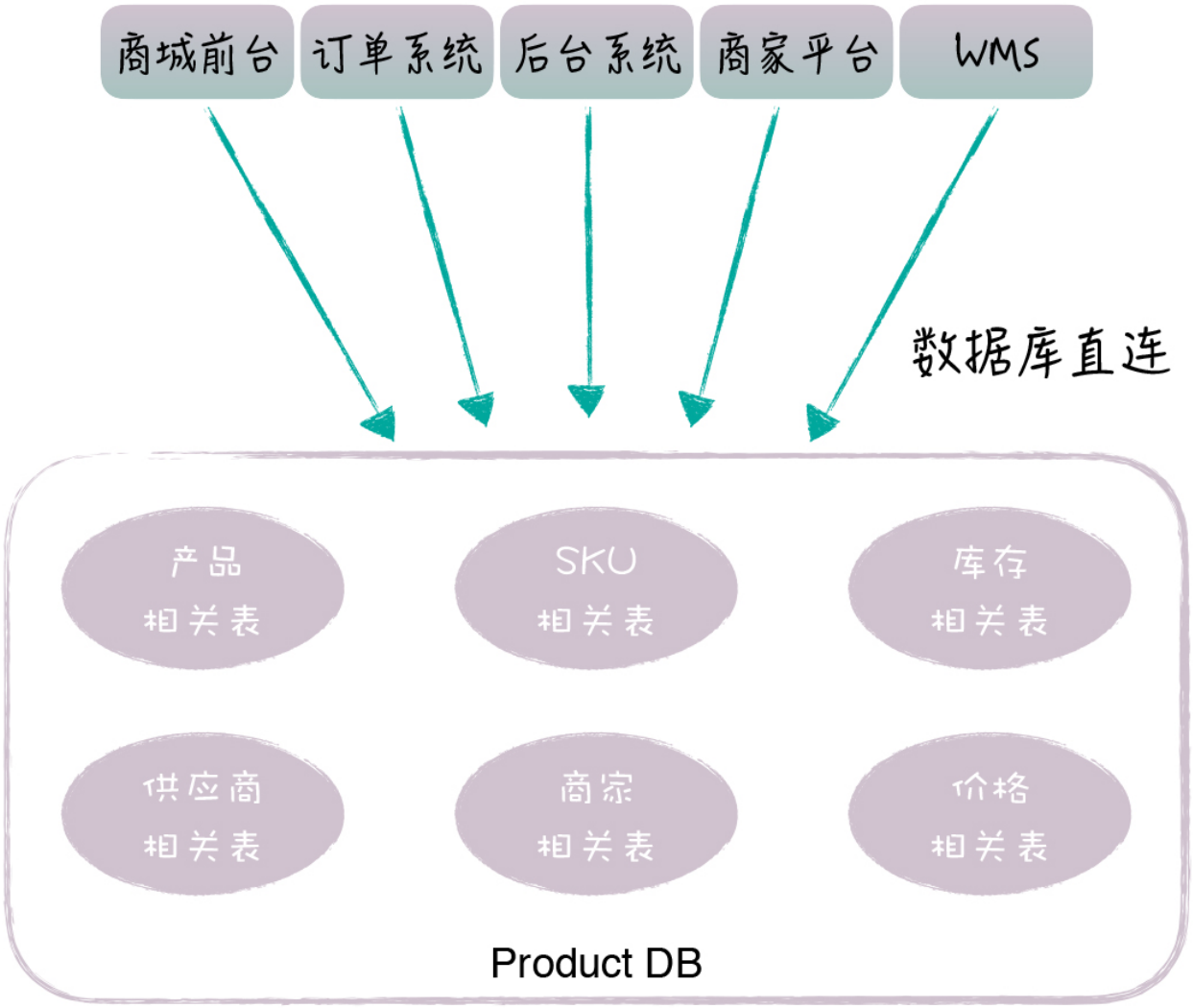
接下来，我就以 1 号店库存服务化改造为例，让你深入理解，我们是如何把库存相关的功能和数据表，从现有系统里剥离出来，最终构建独立的库存服务，并实现和业务系统平滑对接的。

改造背景和目标

我们先来看下这次架构改造的背景和目标。

1 号店作为一个网上超市，售卖的商品种类有数十万个，包括 1 号店自营和第三方商家的商品。由于历史原因，所有商品相关的表都存在产品库里面，这里面有产品的表（产品、分类、品牌、组合关系、属性等）、商品 SKU 的表、商家和供应商的表、库存和价格的表等等，这些表加起来，数量超过了上百张。

我们知道，商品是电商业务的核心，几乎所有的前后台系统都需要访问这个产品库，而这些系统的开发人员，早期的时候，只关心如何实现业务功能，对这些表的访问是怎么方便怎么来，有些 SQL 语句会对大量的表做 Join 关联。所以说，虽然系统是类似分布式的，但数据库是集中式的，如下图所示：



这样的方式，就给系统的维护带来了一系列的问题。

从**应用方面**来说，各个系统功能重复建设，比如很多系统都会直接访问库存相关的表，类似的库存逻辑散布在很多地方；另外，如果修改了库存表的某个字段，这些系统同时会受影响，正所谓牵一发而动全身。

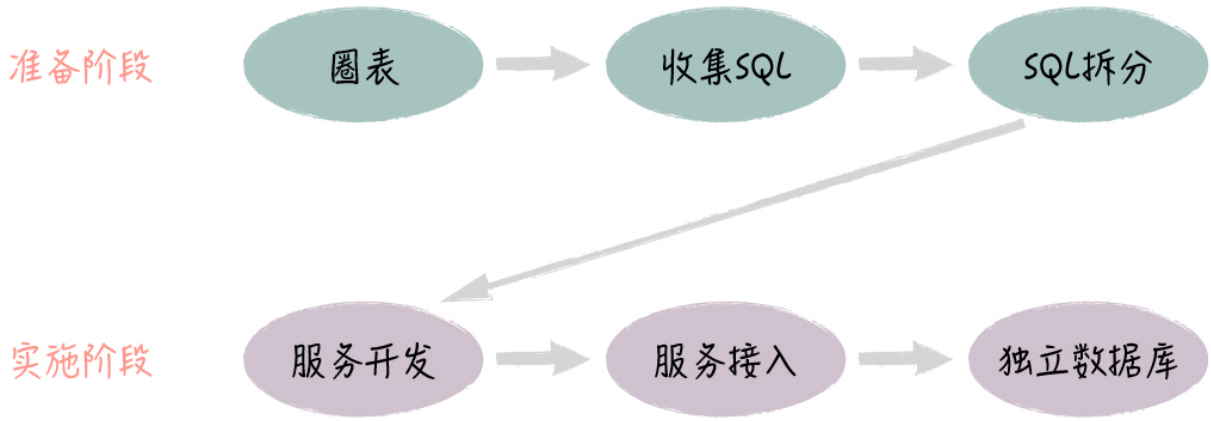
从**数据库方面**来说，数据库的可用性是比较差的，如果某个系统有慢查询，它就很可能拖垮整个产品数据库，导致它不可用；还有，这么多系统同时访问产品库，数据库的连接数也经常不够用。

所以，我们这次架构改造的目标，首先是对这个大数据库按照业务维度进行垂直拆分，比如分成产品数据库、库存数据库、价格数据库等等；然后基于这些拆分后的库，构建微服务，以接口的方式来支持数据库表的访问；最后将各个业务系统统一接入微服务，最终完成整个商品体系的微服务化改造。

微服务改造过程

你可以看到，这里涉及了多个微服务，如果同时进行服务化改造的话，牵扯太大，很难落地。于是，我们选择从**库存微服务**开始。一方面，库存的业务很重要，库存的规则也比较复杂，如果我们能够对库存逻辑进行优化，这会带来明显的业务价值；另一方面，电商的库存概念相对独立，涉及的表也比较少，我们可以相对容易地把它从现有体系中剥离出来。

整个改造过程，从确定库存相关的表开始，到最后把库存表从产品库迁移出来，落到单独的库存数据库为止，一共分为两个阶段，每个阶段包含了 3 个步骤，具体如下图所示：



准备阶段：这个阶段为微服务改造做好前期的准备工作，具体步骤包括了圈表、收集 SQL 和 SQL 拆分。

实施阶段：这个阶段实际落地微服务，具体步骤包括微服务开发、服务接入和数据库独立。

通过这些良好定义的步骤，我们就很好地保证了整个库存微服务改造的有序和可控。接下来，我就具体说明下改造的各个步骤，包括哪些人负责哪些事情、具体的挑战在什么地方，这样，你可以深入地理解整个改造过程。

准备阶段

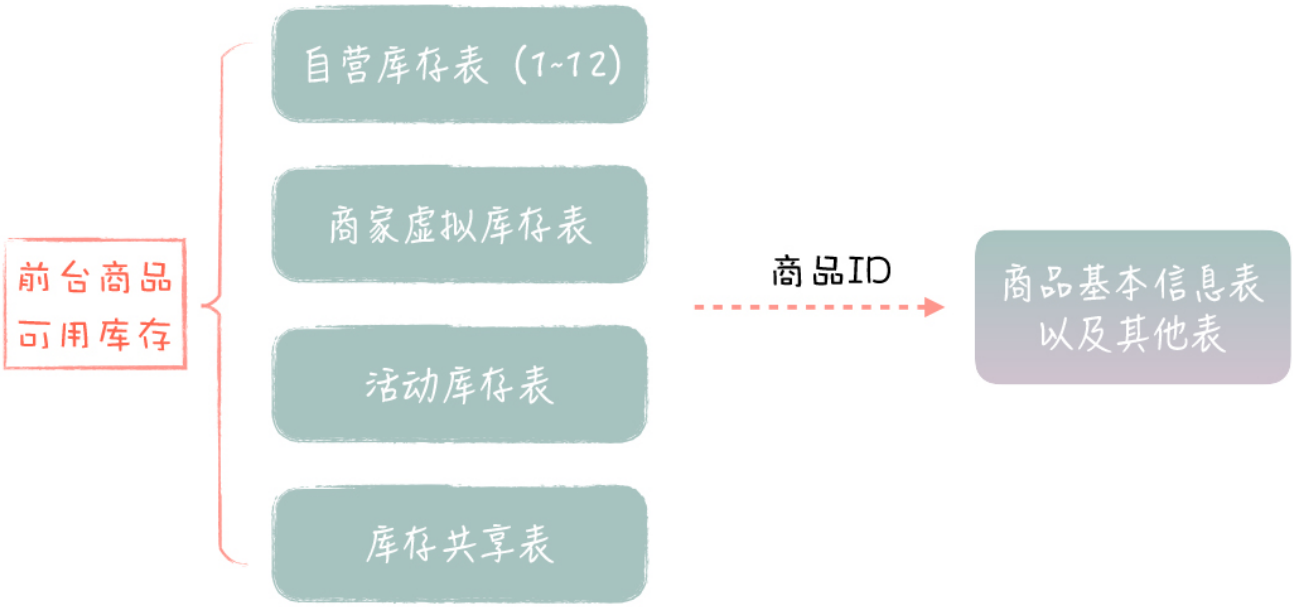
准备阶段的第一步，就是圈表。产品数据库有 100 多张表，圈表就是用来确定库存微服务具体包含哪些表，也就是确定服务的数据模型。在确定了表以后，库存微服务就负责这些表的访问，当然，库存微服务也不会访问其它的表，而业务系统后续将通过库存微服务的接口，实现对这些表的访问。

圈表是微服务改造中比较有挑战性的地方，它实际上对应了服务的边界划分。只是针对老系统做服务化改造的时候，我们更多的是从数据库表的角度来考虑划分，这样更好落地。

针对库存微服务来说，我们要求圈定的表，一方面要满足所有的库存访问需求，这些表之间关系紧密，和其它的表关联不大；另一方面，这些表的数量不能太多，一般不超过十几张。这样，我们既容易拆分数据库，又能控制服务的粒度，保证功能聚焦。

在这个例子中，由于库存的概念比较独立，圈表相对比较容易，一共有 15 张表和库存直接相关，包括自营库存表 (这里有分表，实际是 12 张)、商家虚拟库存表、活动库存表和库存共享表，这些库存表之间是紧密相关的，它们一起决定了前台用户能看到的可用库存数量。

这些库存相关的表都有商品 ID 字段，和商品基本信息表关联，我们知道，库存数量的计算不依赖于商品的具体信息。所以，这些库存表和其它表的关系比较弱，这样我们就可以比较清晰地实现库存表和其它表的切分，简化了库存服务的落地。



在微服务改造中，确定哪些表属于这个服务，会直接影响后续的所有改造工作，这需要有经验的业务架构师和数据架构师参与进来，通过深入地分析现有的业务场景和表的关系，才能对库表进行合理的划分。

所以，你可以发现，**对现有系统的改造，服务的边界划分主要是从圈表入手的，而不是从一个服务应该有哪些功能入手的，这一点和新服务设计是有所不同的。**这有两方面原因：

一方面，如果确定了服务包含哪些表，也就大致确定了服务有哪些功能，而表是现成的，它比业务功能要直观很多，所以从表入手比较高效；

另一方面，如果从表入手，构造的服务和表是对应的，服务包含的是完整的表，不会产生一个表的一部分字段属于库存服务，而另一部分字段属于别的服务的情况，避免表字段的拆分带来额外的复杂性。

值得注意的是，因为这是对现有系统的改造，为了避免一下子引入太多变化，我们先不对库存的表结构进行调整，表结构的优化可以放在服务的升级版里做，这样对业务系统的影响也最小。

第二步是收集 SQL。在确定了哪些表属于库存服务后，我们会收集所有业务系统访问这些表的 SQL 语句，包括它的业务场景说明、访问频率等等。库存微服务后续就针对这些 SQL 进行封装，提供相应的接口给业务系统使用。

这里，服务开发团队负责提供 SQL 收集的 Excel 模板，各业务系统开发团队负责收集具体的 SQL。

第三步是拆分 SQL。对于收集过来的 SQL 语句，有些 SQL 不仅仅访问圈定的这几张库存表，还会和产品库中的其他表进行关联。

比如说，商品详情页需要展示商品详情，它会发起 SQL 查询商品基本信息表和库存表，一次性获取商品的基本信息和库存数量。针对这种情况，我们就需要把查询语句拆分为两条 SQL，先查询商品表获取商品基本信息，再查询库存表获取库存数量。

对于这样的 SQL 语句，我们就要求各个业务团队先进行拆分，保证最后提供给服务开发团队的 SQL，只包含访问库存的相关表。通过 SQL 拆分，我们切断了库存表和其他表的直接联系，等后面微服务落地后，业务系统就可以通过接入微服务，完成现有 SQL 的替换。

SQL 拆分，会涉及一定的业务系统改造，这部分工作主要由各个研发团队负责，一般情况下，性能可能会受些影响，但问题不是很大。

实施阶段

完成了圈表、SQL 收集和拆分以后，接下来，我们就进入了服务实际落地的阶段。

第四步是构建库存微服务。这里面包括了接口设计、代码开发、功能测试等步骤，服务开发团队会对业务方提供的 SQL 进行梳理，然后对接口做一定的通用化设计，避免为每个 SQL 定制一个单独的接口，以此保证服务的复用能力。

这部分工作由微服务开发团队负责，第一版的服务主要是做好接口设计，聚焦业务功能，以保证服务能够落地，业务系统能够顺利对接为目标。将来，服务可以持续迭代，内部做各种技术性优化，只要服务的接口保持不变，就不会影响业务系统。

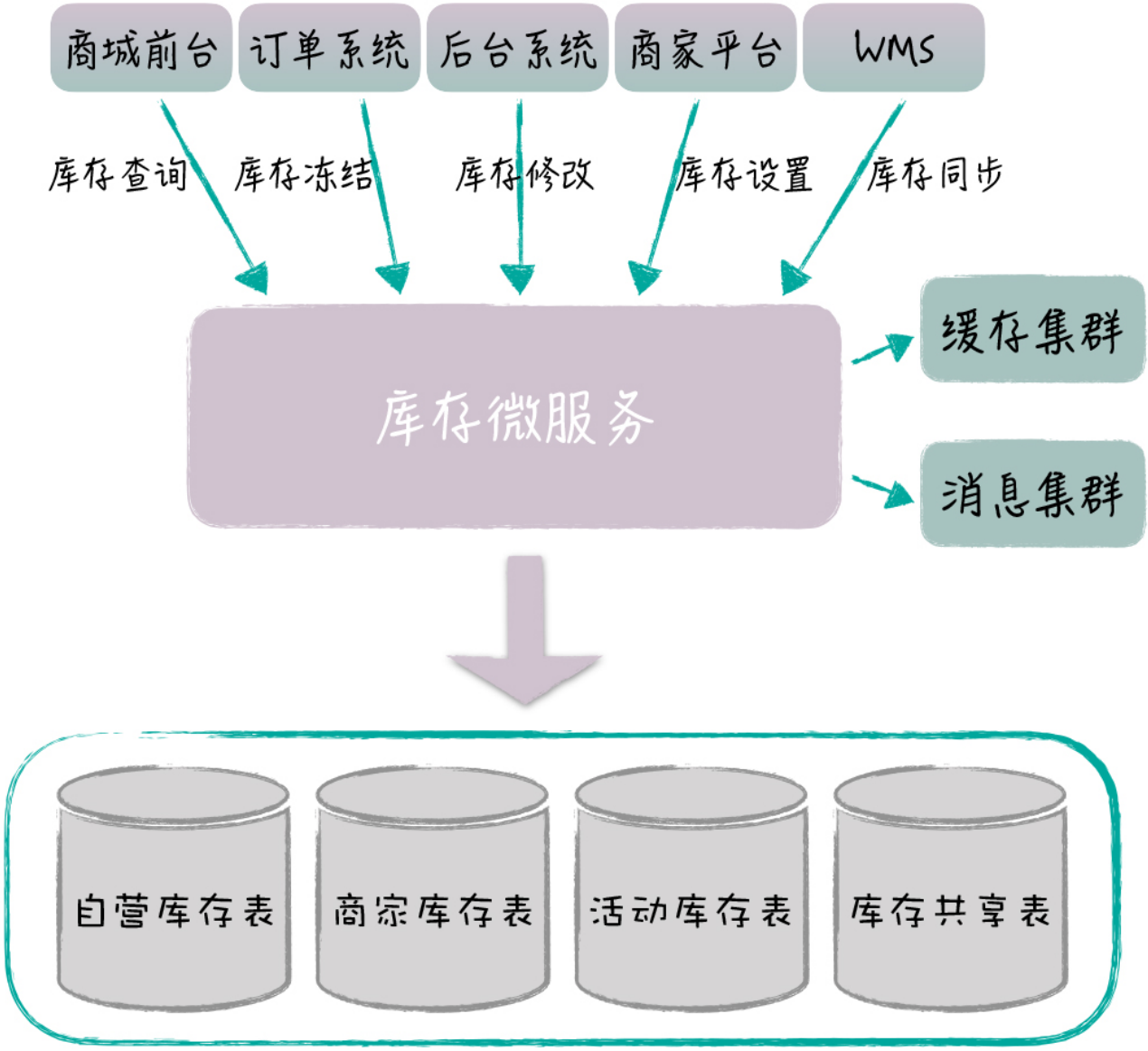
第五步是接入库存微服务。库存服务经过功能和性能验证以后，会由各个业务开发团队逐步接入，替换原来的 SQL 语句。这部分工作主要由业务研发团队负责，难度不大，但需要耗费比较多的时间。

最后一步是数据库独立。当服务接入完成，所有的 SQL 语句都被替换后，业务系统已经不会直接访问这些库存的表。这时，我们就可以把库存相关的表，从原来的产品库中迁移出来，部署成为一个物理上独立的数据库。业务系统是通过服务来访问数据库的，因此，这个数据迁移对于业务系统来说是透明的，业务团队甚至都不用关心这些表的新位置。

通过库存表独立成库，我们可以从物理层面，切断业务团队对这些表的依赖，同时，也可以大幅度降低产品库的压力，特别是大促的时候，库存读写压力是非常大的，数据库独立也为库存服务后续的技术优化打下了基础。

这部分工作主要由微服务开发团队和 DBA 一起配合完成，主要是要避免业务系统还有遗漏的 SQL 语句，避免它们还在直接访问库存的表。我们可以在迁库前，通过代码扫描做好相应的检查工作。

改造完成后的库存微服务架构如下图所示，库存微服务一共包含了 15 张表，对外有 30 多个接口，几十个业务系统接入库存服务。平时，库存服务会部署 50 个实例，大促时会部署更多，我们很容易通过加机器的方式，实现库存服务的水平扩展。



微服务改造小结

到这里，我们的库存微服务就改造完成了，整个改造大概持续了 3 个月，主要是对接的工作比较耗时。

从前面的步骤中，你可以看到，**除了做好库存服务本身的设计开发工作，相关团队之间的配合也是非常重要的。**

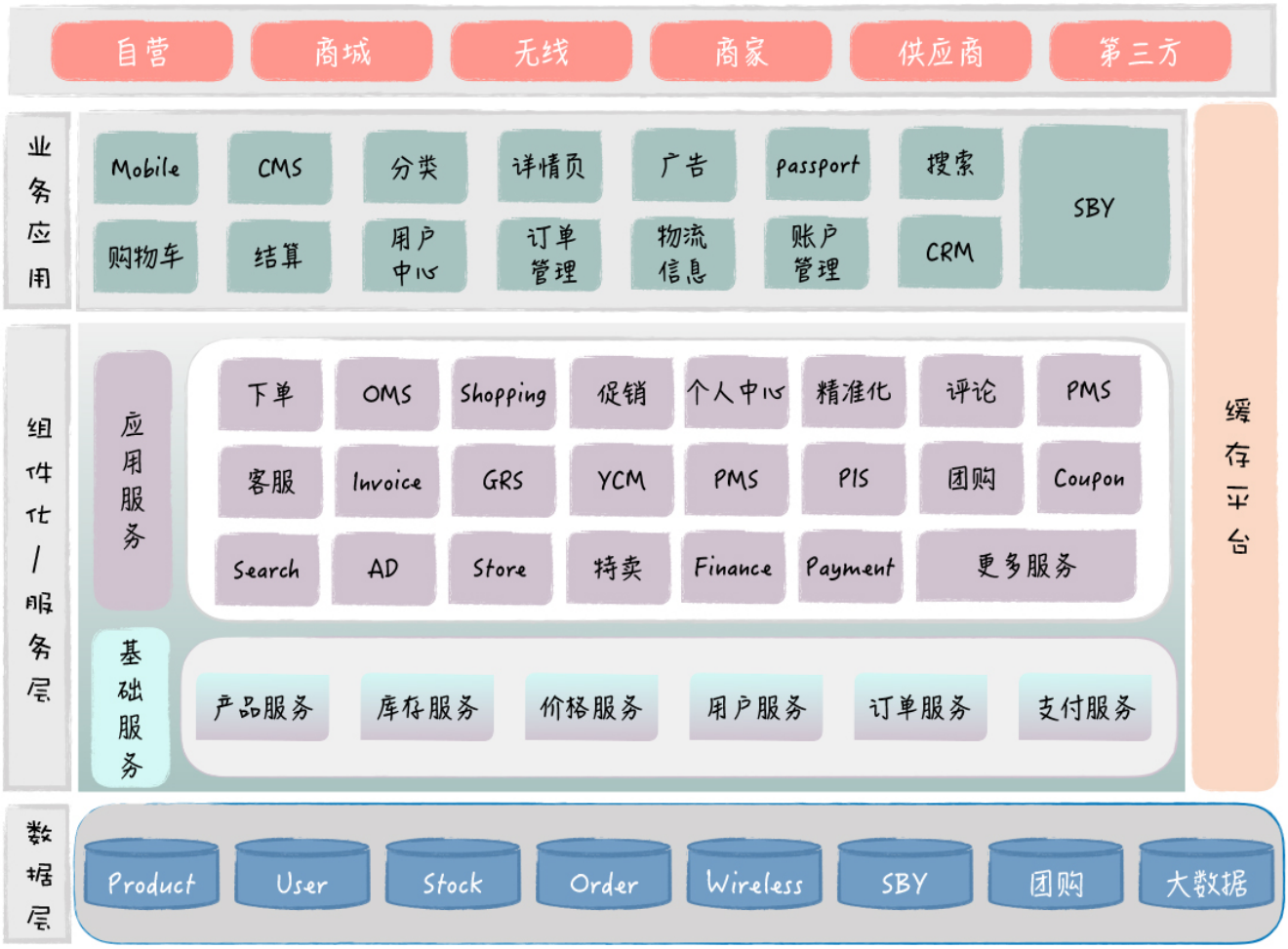
在整个改造过程中，有很多**团队之间沟通和确认**的环节。比如说，服务开发团队圈定表以后，需要和业务开发团队一起确认，保证圈表的合理性；在业务团队拆分 SQL 的过程中，服务开发团队需要介入进去，帮助解决拆分时带来的性能和一致性问题；在服务接口设计和

接入过程中，服务的接口可能需要重新调整，也可能有新的 SQL 进来，双方需要及时沟通，相互配合。

这些都是纯技术层面的问题，值得一提的是，系统改造不会产生直接的业务价值，对于业务开发团队来说，他们往往还需要承担大量新需求的开发工作。所以，**从项目推进的角度来看，这种核心服务的改造，很多时候都是技术一把手工程**。在库存微服务改造过程中，我们也是老板高度重视，大家事先定好时间计划，每周 Review 进度，协调各个团队工作的优先级，确保改造的顺利落地。

以上就是库存微服务改造的例子。1 号店的系统从 08 年就开始建设了，由于历史原因，形成了几个典型的大库，比如产品库、用户库等等，我们通过类似的微服务改造，逐步把这些大库拆分开，构建了一系列的基础服务，如订单服务、用户服务、产品服务、库存服务、价格服务等等。而且通过这些微服务化改造，我们同时提升了业务的复用性和系统的稳定性。

最后，我在这里放了一张 1 号店的总体系统架构图，你可以深入看下，**一个历史包袱很重的系统，它是如何经过服务化改造，最终变成一个能够高度复用和扩展的平台的。**



总结

好了，下面我总结一下今天所讲的内容。

基于现有系统进行改造和全新的服务设计是有所不同的，我们不能追求理想化和一步到位，而是要考虑到系统的平滑过渡，先实现微服务的顺利落地，后续再考虑各种优化。

今天，我通过 1 号店库存微服务改造的例子，给你提供了一种可行的微服务落地套路，让你可以顺利地完成老系统的架构升级。

相信通过今天的分享，你对现有系统如何进行微服务化改造有了更深入的理解，希望你在实践中也能灵活运用。

最后，给你留一道思考题：你在做现有系统服务化改造的过程中，具体碰到了哪些挑战，你又是如何克服的呢？

我是王庆友，欢迎你在留言区与大家分享你的思考，我们一起讨论。如果觉得有收获，也欢迎你把这篇文章分享给你的朋友。感谢阅读，我们下期再见。

上一篇　08 | 可复用架构案例（一）：如何设计一个基础服务?

下一篇　10 | 可复用架构案例（三）：中台是如何炼成的?

精选留言 (8)

写留言

Alex
2020-03-11

拆分最大的挑战还是服务间的多表查询问题。有些用冗余少量数据实现了，有的引入了汇总分析的大数据库。服务拆分后根据业务需要不得不冗余部分数据来换取性能，但是业务需要允许数据会存在短时间的不一致。比如原来设计表里只存了用户id 以前通过关联获取用户名，现在业务表可能就需要冗余用户名信息

展开

3

Jxin
2020-03-11

对标ddd的分层。

1.业务应用对标接入层，是对应用（应用层）接口的调度，并提供外放的出入口。

2.应用服务对标应用层，是对基础服务（领域层）的调度操作，通过整合和操作多个领域...

展开

作者回复: 1.单个包内部还是按照单体应用进行分层，比如control-service-manager->dao
2. 应用服务也有自己的数据库，比如一些配置信息，一些不属于基础服务的普通业务数据。

2

Jeff.Smile
2020-03-11

不错的案例!

展开

1

孙同学
2020-03-11

https://www.processon.com/view/link/5e51378ce4b0c037b5f9d1e3 学习总结更新；突然感觉自己是不是学早了，思考题好多都没经验可答

展开

1

小洛
2020-03-16

具体落地库存微服务的时候，还要考虑数据迁移方案，做到平滑过度，还要考虑出故障了可以随时回滚，这应该属于技术架构去保证的了

展开

蓝天
2020-03-13

1，统一思想很重要，需要上层强有力的推动，否则很难

2，我们现在的困境，业务方不愿配合，很多老代码换了好多批人了，想改调用方式可以，直接去业务方代码里自己改，改出问题你们负责，上线不能停机，业务不能受影响，有问题要能立即回退，（我们现在都是代码里加开关，但是数据库迁移不好弄）...

展开

> 作者回复: 这样做功能调整确实比较痛苦，代码要两套，数据两份，要随时保证能够回去，系统越改越臃肿。
>
> 可以先做好老板工作，然后在公司内部架构布道下，让业务开发方也意识到架构太旧，隐患很大，公司上下形成共识和合力。

Din
2020-03-11

曾经经历过一个单体系统的改造过程，主要遇到以下的问题：
1. 系统改造短期是不会产生业务价值的，所以需要说服领导，让领导从长远利益考虑同意这件事
2. 不能停下来完全做业务改造，同时还要兼顾新功能的开发
3. 团队协作问题，需要大量的沟通和业务梳理...

展开

> 作者回复: 嗯，这些都是比较典型的问题，和本文描述的非常类似。
> 通过老系统改造，相信你也学到很多宝贵的经验。

tt

2020-03-11

对于老系统，先做数据的拆分，将业务子域的数据从全集数据中拆分出来，然后通过服务的形式把数据暴露出去，形成微服务。

在选择将哪个业务子域的数据拆分出来时选择的标准是收益成本比较高，达成的目标是...

展开

作者回复: 赞一个，很好的分享，把文章和实际做过的东西很好地结合起来，大家感触更深。
对于服务来说，应该是代码和数据一体的，不然，表访问会不断越界，这相当于，部分业务规则游离于服务外部，违背服务完整性原则。