天堂向左，中台往右

前言

经历了半年多的996 + 9107，快到12月份的时候终于将项目完成了。公司三年前就要向新零售领域发展，经过不断的

努力和摸爬滚打终于有了一个可喜的成果，其中的心酸苦楚着实不易！承蒙领导的关爱和信任，让我负责电商中台的建设工

作，感激之情无以言表，唯有做好工作予以报答！我是一个喜欢总结的人，就以这篇文章记录和分享一下我在今年的电商中

台的建设中的一些感悟和体会吧。才疏学浅，水平不足，文章偏颇之处还请各位前辈、大佬多多指点。

本文将会以中台化建设中遇到的种种情况和问题进行阐述和分析。

1、为什么要做中台

1.1 我对中台的理解

中台这个概念起源于美军的作战体系，强调“支撑”、“高效”和“灵活”，最主要的目标就是提供强有力的后勤保障。兵

马未动粮草先行，商场即战场，在对等的外界条件下一场战争能否打赢往往看你的后勤保障能否到位。所以中台的建设更多的

是为企业提供后勤保障。具体到互联网的技术开发中来看是这样的：将已经庞大、复杂的业务细化和拆分，避免重复造轮子，

解决开发效率问题，降低创新成本，缩短工期；从而让公司的业务方快速进行市场验证和试错，让公司在和自己的友商竞争中

处于绝对优势。

比如在市场中发现了一个新的商业方向，人力资源相同的情况下需要100天才能开发完成。友商的优势是先于我们发现了

商机，并且提前20天进入了开发，缺点是他们没有中台建设，一切都要从头开始。 而我们自己的公司缺点是后发现商机，但

优势是我们有中台提供支撑，其中有80%的功能是可以完全复用的，剩下的功能开发只需要20天就可以完成。 也就是说公司

只需要20天就可以将产品上线，对比友商他们还要60天才能完成开发。这些富余出来的时间会收集到足够的市场反馈，而友

商的系统上市就面临过时的风险。这种强大的驱动力能为我们的公司在市场竞争中提供强大的优势，时间就是一切！

1.2 中台带来的好处

综上所述，中台带来了如下好处：

1. 快速推进新业务上线。
2. 快速试错、验证商业方向准确性、快速响应市场变化。
3. 降低人员成本，研究性开发与业务性开发分离。
4. 提高代码的可复用性。

对于第四点“提高代码的可复用性”在上面的例子中是一种隐性的体现，对于非技术出身的读者可能体现的有些不明显，下文

还会有更为详细的阐述。

1.3 中台为业务赋能

1.3.1 理想状态下的技术部与业务方

中台是一个为公司提供后勤保障的角色，那么谁是我们的部队和士兵呢？我的答案的是公司中的业务方。业务方去销售我

们公司的产品，他们就要和“战场”中的友商拼命厮杀，买了我的产品客户就不会用你的产品，我多吃一口你就少吃一口。市

场就这么大，客户就这么点儿，有你没我像极了一场战争。业务方在面对瞬息万变的市场环境时肯定需要根据市场反馈及时做

出业务上的调整。通常来讲中台提供的能力是可复用的，特定业务上的调整并不会造成多少影响，我们只需要调整业务代码即

可。这种情况下工程师们加的班都是有意义的，技术部门与业务方相辅相成共同完善产品，和谐了~

这种理想状态很少见，需要公司的决策者有大格局。往往这样的公司都是很成功的，因为决策者了解互联网科技公司生存

的根本是：技术。如果公司的技术不过硬，靠什么在竞争中生存？稳扎稳打，一步一个脚印，不走弯路才是捷径。急功近利还

是要付出一些代价的，要么买成熟的中台技术，要么后期下血本去对技术做中台化升级改造。

1.3.2 现实状态下的技术部与业务方(绝大部分企业存在的情况)

生存是企业基本底线，盈利是企业的最终目标。我相信这个观点大家都会同意，依据这条公理老板都会给自己的业务部施

压：销售要出业绩，每个季度要做考核、定目标、评绩效等等。业务方在此情景下为了完成销售目标他们会绞尽脑汁来拉取客

户、迎合市场；新奇的想法和需求会一个个飘来，不论合理与否。

**情况1：中台化已经完成**。

在这种情况下中台为业务赋能时，只要不被业务方的需求侵入中台能力，一切都还好，只更改业务中台的代码即可。如果

要是业务方的需求已经需要中台去修改自己的能力模块了，那么事情可能会慢慢朝着不和谐的方向发展(中台化不健壮)。

**情况2：中台化进行中**。

一件可怕的事。业务方与技术部门会爆发激烈的冲突。通常一个公司中，开发资源是很有限的，他们去参与中台化的项目

就肯定没有时间去维护、迭代业务方的需求。业务方身上背的业绩往往无法完成，就会反复和技术部门冲突，矛盾也会越来越

大。此时公司往往会很动荡，这也是常说的不做中台等死，做中台找死的原因。

这种情况下，公司的决策者需要作出取舍：你想用空间来换时间还是以时间来换空间。流的血都是欠的债，只要公司的中

台化能够完成，那么公司将焕发一次新的生机，整体技术实力也会成倍提升。

1.3.3 如何赋能

从前两个小节可以看出，中台从建设到赋能不是一件容易的事。中台赋能需要看重2点：

1. **关于人**

业务方与技术方需要妥协。在这个项目中我们遇到了一些很棘手的问题，就是出在人身上的。有些产品需要是需要业务

来推进技术去实现的，这是正常的产品需求；但另一些产品需求是和技术有强关联性的，应该由技术人员来推动产品去

设计。在项目中如果业务方或产品人员过于强势，轻则破坏中台架构的合理性和健壮性，重则导致项目严重延期。

1. **关于技术**

务必统一技术框架，达到各个部门通用，减少各个部门间的学习成本。

如果能满足上面的两个条件，那么中台赋能就变的容易的多了。一个新的业务开发流程和时间会大幅减少，在服务器端，新

业务只需要简单扩展一下特定的业务代码即可，其主要的工作会集中在个性化定制的前端上，这也是前后端分离形成未来趋势的

一个主要原因。

1.4 你的企业是否适合中台化改造

期望进行中台化改造的企业一般来讲会有这样的情况：公司目前或在不远的将来会出现大型、复杂的生态系统，如果不中台

化改造会出现重复建设的问题，随着业务的不断扩大项目变的会约来越多，越来越臃肿，需求迭代越来越慢，项目管理变的越来

越困难。中台化从另一个角度看是一个企业技术体系瘦身的过程，毕竟“胖”不是一件好事儿。如果存在这些需求，那么公司需

要中台化改造。然后我们再从公司的性质进行分析：

**A 伪互联网类型公司**

这类公司你说他是互联公司吧，有些牵强，说他不是吧他又做了一些互联网公司要做的事，这类公司占比很多。

售卖IT服务，对技术的要求不高，急于扩展业务，偏销售倾向，服务于传统行业人群，又和互联网打一些交道但不是很多。

这类公司往往对互联网技术的使用浮于表层，一些互联网技术他们也用，但不会考虑互联网带来的高并发和服务器性能问题，缓

存的使用占比通常在20%以下，消息队列也是可有可无，对于核心的服务治理框架通常缺少自己的积累和沉淀。如果公司还健在

，现在往往也在进行中台化改造。

特别巧，我所在的公司正是这种类型。公司高层认识到了未来的发展趋势，下了很大的决心进行中台化改造，今年终于有了一

点成效。但就客观的说，中台化的程度并不彻底。CTO规划的电商中台、营销中台、订单中台等等核心技术部门方向是很正确的

但是每个中台的技术框架与核心底层代码并不通用，这无疑会加大技术人员的学习成本。中台化即是后勤保障更是一个“制式武器”

，在战场上武器如果是万国造，打起仗来肯定会掣肘。

出现这样的问题，我得出的结论还是在人的身上。每个中台的负责人都有自己的个性，他们会按照规划去做，但是如何做得

“我”自己说了算，也就是说并没有完全服从指挥。这种情况事实上很常见。

所以结论是：适合。这类公司一旦中台化完成也就完全变成了纯互联网公司，对比自己的友商将有更为强大的竞争力；竞争

力的强弱会根据其中台化的彻底程度来决定。

**B 纯传统行业公司**

使用的技术以SpringMvc、SSH等单体应用框架为主，提升性能以扩展机器为主，很少用互联网技术，基本不涉及服务治

理，会简单的使用缓存。这类企业不在少数，其中还不乏一些知名的电商公司。 这类公司的系统中鱼龙混杂，有一些子系统还

是外包公司做的，或者直接购买第三方现成的项目，可维护性很差。

如果单纯的只做政府项目、内部系统，我觉得没有中台化的必要。

**C 以单体应用为主体技术的电商/互联网公司**

使用的技术与B类公司一样，都是以SpringMvc、SSH等单体应用框架为主，以几十台服务器以内的小集群模式存在，流

量峰值在1万以下的中小型电商。他们起步早，同时已经在市场立足；比较有代表性的是惠家有、乐友孕婴童等(2018年前了解

到的)。核心项目中的代码都是自己研发人员开发的，周边的进销存系统、BI系统基本是买的或外包人员开发的。

这种类型，中台化改造的代价是最大的。业务运行的很好，也没啥毛病，在流量高的时候多买几台阿里云的服务器就可以解

决。如果进行中台化改造，短时间内恐怕无法完成(我们公司中台化改造至今已经3年，无经验可循只能公司内部反复摸索)，无

论是时间成本还是风险成本都很高。但如果不改造又会制约企业的战略和未来的发展，相当纠结。

我对这种情况的看法是这样：不要再内部摸索了，尽量去找成功的案例，能花钱解决的问题都不是问题。中台化就已经很难

了，同时还要面对为老项目进行微服务化重构的任务，而且老项目还不止一个！个人感觉成功的希望比较渺茫。假如公司获取了

现有的中台化改造方案，将工作重点放在老项目的微服务化重构上，人力资源成本也会很高。按照现在的市场行情来看，一个工

作经验在4年以上，同时微服务开发经验在2年以上的人，通常薪资水平会在2万上下，公司承担的人力成本会在2万6左右，

那么一个人一年的成本约为35万。根据公司项目的大小不同，这样一个团队的规模会维持在30到40人，一年的成本约为1000

万到1500万之间。

这里还未考虑HR组件团队的时间成本以及公司内部的老员工是否全力配合。中台化如果一年之内能够彻底完成，简直是奇

迹。所以各位大佬还是要谨慎。

**D 创业型公司**

主要看创业者的心态。如果想搂一笔钱就把公司卖给天使们，那无所谓什么中台化，依托于业务怎么快怎么来；如果是将它

当成自己的事业和梦想，我觉得有必要进行中台化，让他为你的梦想保驾护航。公司在创始之初就以中台化的方式来支撑和设计

，那么系统整体的健壮性、可扩展性和稳定性一定会很强。你的任何业务都需要以技术来进行支撑，技术才是公司的根本所在。

在15年的互联网大潮中，很多创业公司如雨后春笋一般出现，但又很快消失掉了。根本原因就在于其技术基础薄弱，光靠PPT

是赢不了市场的。那时候很多公司的商业方向都很好，不少都拿到了A轮投资；但在进入B轮之前基本都死掉了。因为投资人

会不断围绕着你的商业方向提出新的市场需求，市场在不断变化而你的技术所支撑的系统无法快速响应这种变化，投资人会怎么

想？在这个时期公司扩招了员工，资金链一旦断裂公司会非常动荡；人心不安，离关门也就不远了。

截止到此处为什么要做中台就论述完了，4千字有点长。

2、中台的团队形态与定义

2.1 中台团队常见形态

中台的建设需要一个统一的组织来领导，从上而下去推动，否则中台化很难落地。这就涉及到组织架构上的调整，需要建立一

个中台团队。中台团队的常见形态有2种：技术委员会和平台架构部。

1. 技术委员会

业务流派的中台团队常见形态，由每一个项目组的负责人或技术Leader来共同组成，承担组织和协调的任务，执行人员分

散在原有的各个部门里。实际项目中沟通和协作成本很高，中国人的特点是喜欢看热闹，所以实际产出并不高，复用的价值

体现的并不多。当然了，每个公司的企业文化是不同的，这种情况并不一定存在。

1. 平台架构部

技术流派中台团队的常见形态，公司需要投入的成本比较高，通常由运维架构师和系统架构师共同组成，主要负责核心底层

框架的封装工作和探究性质的技术开发工作。平台架构部通常是高成本部门，没有商业价值产出，从企业角度看这是个挺严

重的问题；如果技术中台趋于稳定以后，能够将这些技术精英投入到各个业务部门中将会是一个很好解决方案。让他们去基

层培训公司的初/中级工程师，将技术中台的思想传播给他们，从而提升人才梯队的技术能力和思想水平。

2.2 中台的定义

对中台的定义主要集中在2个流派上：业务流派和技术流派。业务流派认为中台其实是一套思想，只要能够实现能力的复用

，降本企业成本，提升交付效率；所采取的一切措施都是中台的范畴。所以从这种观点出发中台更偏重为一种组织方式，企业内

部复用的是彼此之间的能力。由此衍生出来的中台框架以多语言与多架构共存型中台为主。技术流派认为中台一定要有核心框架

来支撑，需要定义一套技术规范和通用的核心底层框架，事业部之间使用同一套技术框架，而核心底层框架由架构师组成的架构

部来维护和迭代；通用底层约束着业务组件的行为，性能、稳定性和扩展性对业务开发者隐藏，业务部门只关心自己的业务即可

，不用参与到费神耗时的“研究性”开发中去。由此衍生出来的中台框架以多统一架构型中台为主。

2.2.1 多语言与多架构共存型中台

业务流派所认同的中台形式。这种情况多存在于系统比较多的公司，系统之间架构也不同，语言也各异：SSH、SpringMVC

Spring-boot、Dubbo、PHP和.Net等等。公司通常在一段时间内经历了快速成长，对于技术架构和底层代码没有夯实的积累，

技术部门急于完成开发任务，从而催生的一种现象。

好处和优点是企业可以快速完成中台的建设任务；但缺点更多：

1. 内部系统的调用方式以Http接口的形式为主，效率低。
2. 框架之间不通用，没有形成统一的技术体系，缺少统一的约定与规范，导致开发人员学习成本变高，中台的可维护性低。
3. 各个子系统之间的项目性能无法保障，导致中台整体性能受限。
4. 错误排查困难，沟通链过长，项目开发周期会更长；同时会消耗开发人员更多的精力，开发团队稳定性也会下降。
5. 技术栈过多、不统一，导致运维层面更加复杂；互联网公司到最后基本拼的都是运维能力强弱。

这方式来推进企业中台化速度是最快的，但治标不治本，没有从根本上解决企业存在的问题。

2.2.2 统一架构型中台

技术流派所认同的中台形式。这种情况多存在于技术积累夯实、研发预算充足的中大型公司。比如阿里，中台化历时三年之

久，其内部开发标准、约定、技术都是统一的；阿里的中台化战略十分宏伟，但一般公司根本模仿不来，即便像腾讯这样的公司，

中台化也没有这么彻底。我很佩服阿里，非常厉害，又一次为行业设立的标准。

由于统一架构型中台的特点是成体系、成建制的，那么在【多语言多架构共存型中台】中所暴露出来的问题基本都不存在；

可是他也有自己的弊端：技术成本太高，人力资源成本也太高。随着项目的展开，如果没有充分的准备，项目风险会越来越大。

阿里虽然树立了行业标杆，但盲目模仿阿里的中台策略是不可行的；要知道阿里每年在技术上的投入都是以“亿”来衡量，很多

公司每年的利润都没有达到千万级。那我们应该如何去做？

2.2.3 技术中台与业务中台分离

这是我在项目中的一点体会，核心想法也是围绕着【统一架构型中台】来实现的；在阿里的中台思想上退一步，因为他们的

做法对于99.99%公司来讲偏“激进”，中小型企业无法承受，即便大一些的二线互联网公司也很难招架得住。我的思路是这样的：

服务治理使用Dubbo来做，消息队列使用RocketMq，一级缓存用Ehcache，二级缓存使用Redis集群，数据库使用Mycat+My

SQL，Api(https网关)出口使用SpringMvc。Web项目作为服务的组合层来聚合用到的Dubbo Rpc服务，然后统一对外提供API

接口。在项目中无论是Dubbo项目还是Web项目，他们都共同依托于同一个核心底层项目，而且项目中的Spring配置文件都会

被剥离到核心底层项目中统一维护和管理。让【核心底层项目】形成一个有通用性和无关性的【技术中台】；所有业务中台的项目

均依托于技术中台，在底层架构之上集成开发能力。

打个比方：技术中台就像土壤，业务中台就像农作物；技术中台为土壤提供什么样的肥料、什么时候浇水自己来决定，只保证

农作物能够健康、快速的成长即可。业务中台只关心种哪种植物，种玉米挣钱多你就选玉米，种人参挣钱多你就选人参。如果今年

种玉米赔钱了，那么OK马上把所有的玉米都砍掉，我们种人参；但是土壤还是以前的土壤，还能继续复用。映射到实际的项目开

发中，我们只是去掉了不赚钱的业务模块，而技术含量更高的核心引擎没有变化。

经过【技术中台】的高度抽象和封装以后，所有依托于他的业务项目就变成了最简单的增删改查，最多再加个缓存设置和消息

队列，这些活儿刚毕业的大学生都能做。让业务开发人员更加专注于业务代码，减少难度较高的研究性开发，这么做在实际项目开

发中能够大幅度的提升开发效率，减少交付时间，同时系统的整体性能和稳定性也更有保障；公司中的2个项目是这么做的，亲测

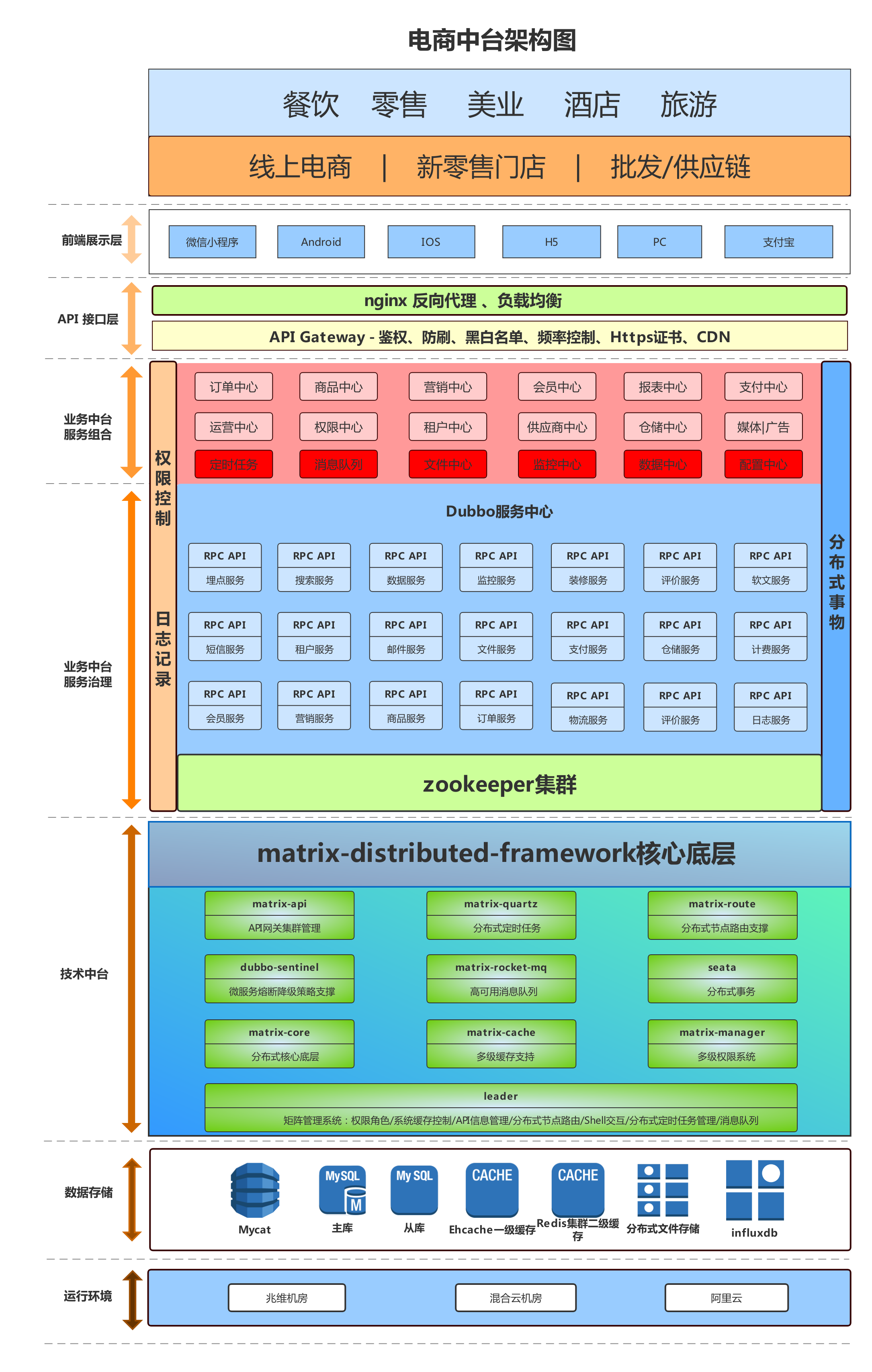
有效。随着业务中台的分离，项目整体的可控性也会提高很多；技术中台的剥离意味着对项目进行了一次有效的减负，优化了资源

配置，同时也能有效降低人员成本。

3 中台架构如何落地

3.1 系统架构设计

很多公司都在内部推中台的事情，我们的落地方式仅供参考：



依托技术中台，提供了API的统一入口项目、基于Quartz二次封装的分布式定时任务、服务节点路由、服务降级、分布式

消息队列、分布式事物(seata目前尚不稳定，还需要继续跟进)、多级缓存、统一的权限账号体系、核心底层matrix-core和开发

控制台：矩阵分布式管理系统(leader)。matrix-distributed-framework是我们的技术中台，译为矩阵分布式框架；这个名字是

根据哥德尔命题来的：在任何矩阵系统中，总有真理游离于逻辑之外。之所以这样说是因为一个系统也好、平台也好无论设计的

多么完美(至少在你看来)，他都会有自己的缺陷和漏洞。

Matrix提供的各个模块都是一个个以maven来管理的子项目，使用的时候他们会被编译成文Jar包发布到私服上，供各个

业务项目来使用，Dubbo服务或Web项目再通过Jenkins来进行构建和发布项目。这些高度抽象的模块，强有力的封装了他们

本身的功能域，项目中所遇到的绝大多数情况都会被考虑到。比如一个基于Tomcat容器的Web项目如果想对外提供统一的API

接口，那么直接引入matrix-api项目的maven坐标到pom中即可实现，你不用关心鉴权、验签；再比如你想让这个项目提供定

时任务的能力，那么你只需要引入matrix-quartz即可，至于分布式定时任务如何实现、分布式锁什么时候加都不需要业务开发者

关心，你只需要实现一个抽象类：RootJob.java，然后在固定的方法中写增删改查，完成后在矩阵控制台中加入你的定时任务，配

置好触发规则。

matrix-core是一个最核心的模块，他剥离了每个项目中的Spring配置文件；也就是说在Matrix的框架体系内，Dubbo项目

和Web项目都是没有applicationContext.xml配置文件的。这样的设计为项目提供了极高的灵活性，如果私有化则所有的Dubbo

服务编译成jar包放入到Web项目中形成单体应用；如果微服务化，则正常部署即可。

在技术中台之上，是基于Dubbo的服务中心。服务依据业务和功能来进行划分，他们都有各自明确的职能边界；他和服务组合

层共同组成了业务中台。Dubbo的服务组合会在Web层来进行，尽量避免Dubbo服务相互间的调用。技术中台会为业务中台提供

统一的权限控制、日志管理和分布式事物控制功能。

3.2 架构细节分解

3.2.1 微服务选择

在服务治理框架上，我们选择了Dubbo。原因有如下几个：

1. 久经考验的阿里战士，为电商而生，无论从性能还是稳定性上都是有目共睹的，能够抗住淘宝的超高流量。
2. 版本稳定。Spring-Cloud虽然是新贵，但是版本迭代过快，并不稳定；在项目开发中是大忌。
3. 中文文档齐全，研究的人也多，社区活跃。
4. 与Spring无缝融合，只要写过SSH、SpringMvc的人都可以通过简单培训快速入手项目。Spring-Cloud学习成本高。
5. 基于Rpc协议的调用比Spring-Cloud提供的Http协议调用少了3次握手，消息传输效率更高。

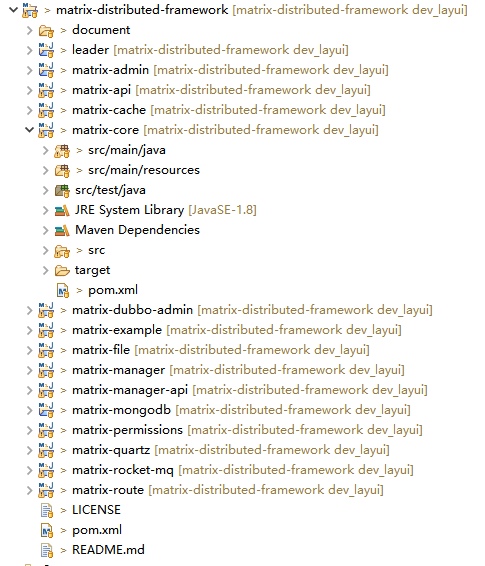
3.2.2 Matrix职能介绍

Matrix的最初目标是积累和沉淀，致力于打造一个企业自己的核心底层框架。

从架构图中可以看出每一个Dubbo服务项目都依赖于Matrix所提供的基础能力，向上支持业务开发，向下则打通每一个孤

岛连接。矩阵分布式管理系统(leader)负责提供一个Web视图，用于管理核心配置项，形成最基础的技术中台。下面将介绍每一

个子项目的职能。



上图是Matrix整体项目结构，下面会将最重要的子项目逐一介绍。

项目地址：https://github.com/PowerYangcl/matrix-distributed-framework.git

**1 Leader**

针对matrix-admin项目的升级版。Leader使用付费的Layui作为新的页面框架，相比之前简陋的Jsp页面封装更加友好。

Layui版本为2.5.3，在其基础上对table.js进行了少量修改。其余功能与matrix-admin完全一致。

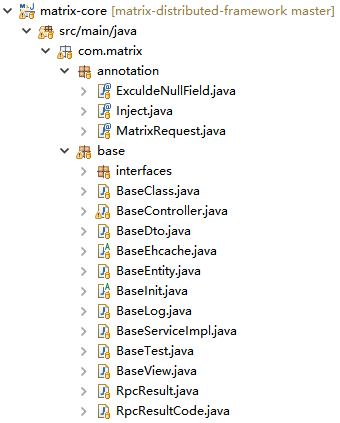
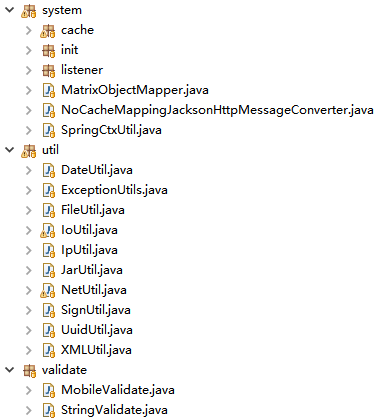
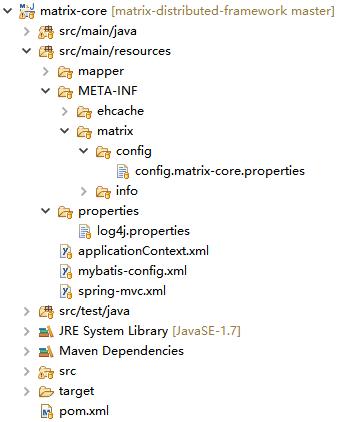
**2 matrix-core**

分布式系统的核心底层项目，提供大量基础工具与脚手架支持，所有衍生子项目、业务项目都需要在POM中集成这个Jar

包。他剥离了applicationContext.xml让每一个业务项目形成一个壳子，Dubbo项目和Web项目共同使用这个配置文件。Web

项目则需要再加入额外的配置文件：spring-mvc.xml，这个文件专门在applicationContext.xml基础上增加了Controller层的

配置

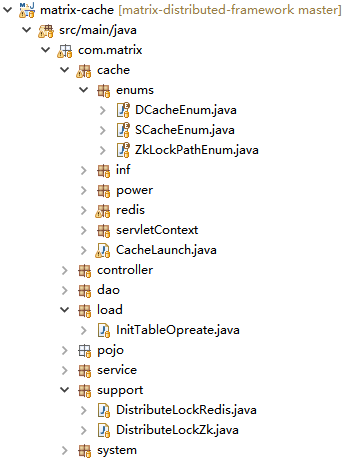
**3 matrix-cache**

系统缓存封装，一个系统性能是否强大，通常要看其在业务代码中缓存使用的占比和如何使用消息队列。

此项目提供Servlet Context、Ehcache和Redis三种缓存模式，Matrix系统使用Ehcache作为系统一级缓存，Redis作为系统

二级缓存，Mycat做数据库分库中间件。系统一级缓存的读写效率是二级缓存的上万倍，但缺点是需要项目经验丰富、技术能力强

的人才能在具体业务场景中使用。由于是分布式系统，故对Ehcache不了解的人请先详细了解他的使用方法，以免出现大的差错。



DcacheEnum.java定义了数据字典相关枚举信息，ScacheEnum.java定义了业务字典相关枚举信息。CacheLaunch.java是整

个缓存的入口类。DistributeLockRedis.java提供基于Redis的高性能分布式锁，DistributeLockZk.java提供基于Zookeeper的高可

靠分布式锁，这两种锁可以根据具体的业务场景，自由选择。

缓存底层把数据库查询、拼接数据的业务的代码统一归纳到一个具体的处理类中，这么做的好处是查库放到缓存的这个过程，只

有一个入口，业务开发人员不必担心会有多个地方存在同一个缓存的获取行为，以架构约束的方式，将与业务代码无关的缓存获取操

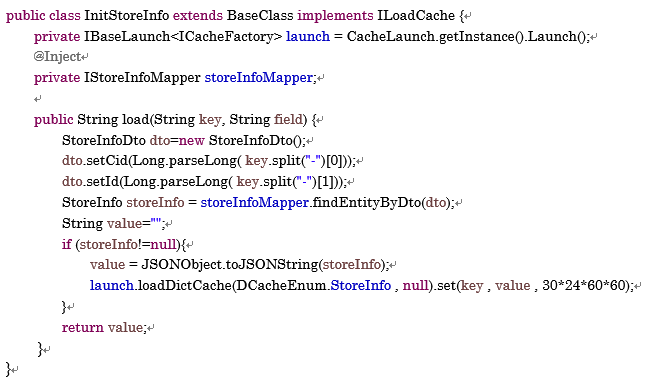
作从Spring的Service层中分离出来，单独保存，最大限度的减少因为遗忘、疏漏而产生的缓存不同的问题。同时，此方法做了防穿透

处理，如果一个缓存Key在10分钟内尝试20次连续的数据库查询操作，那么将会在10分钟内连续返回空字符串；也就是说业务开发人

员无需在实际业务开发中再考虑缓存穿透的问题。



如下代码段展示了一个简单的缓存未在Redis中命中，从而去数据库中查询数据、拼装JSON串儿的过程：



4 matrix-permissions

对matrix-manager的升级项目，此处提供了统一的系统权限。在Matrix体系内，权限被高度抽象和剥离；开发者从Leader系统

中录入每一个系统的功能点和权限，然后在分配给对应平台的超级管理员；Leader中维护了所有的子平台权限和功能。这里体现了矩阵

框架“统一”的想法。

**5 matrix-api**

平台内置的API网关项目。所有基于Tomcat的容器型项目，在POM中加载这个Jar包后，都具有API接口提供能力。API项目

采用MD5加密的方式进行验签，矩阵系统会对请求者提供对应的公钥和私钥。一个完整的接口请求数据结构如下：



target：api标识符

accessToken：登录后的令牌，过期时间半小时

client：3代表客户端，固定值

version：版本号，vsesion-2.0.0.1，固定值

requestTime：本次请求发起时间，md5加密依据之一

channel：渠道，固定值

key：公钥，由Leader平台系统进行分配

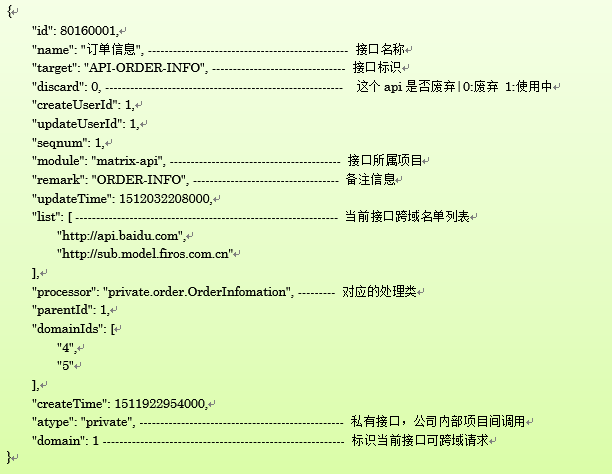
私钥：此处不体现，由系统分配给对应客户端的请求者。

value：md5加密后的结果。

data：请求所携带的参数，其中platform为固定值，每个平台对应的值不同，由数商管理平台统一分发。

在系统设计中，所有的API信息均以缓存的方式存在于Redis中，每一个接口都可以针对指定的域名进行跨域操作，接口间的跨域问题由

底层统一解决，不用开发者处理。完整的API JSON结构信息如下：



6 matrix-file

文件操作。这个项目中提供了操作文件的接口，当一个Tomcat项目在POM中引入此Jar包后，他即可成为一个简单的单点文件服务器。

7 matrix-quartz

为分布式定时任务提供支持。基于Quartz，系统对其再次进行了高度封装，使得开发人员可以轻而易举的完成定时任务的开发工作。

在矩阵后台，有非常人性化的界面来配置、修改、查看和关闭你想要操作的定时任务。

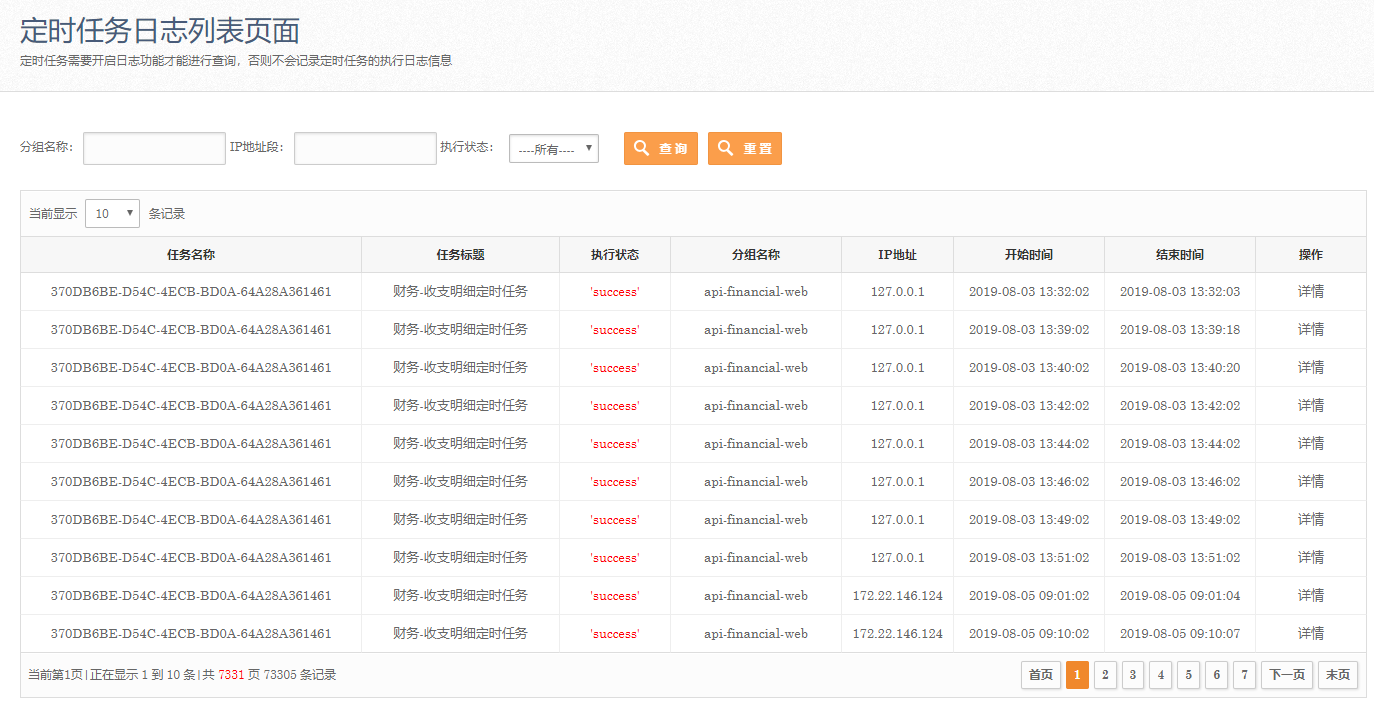


系统将定时任务的并发控制放到了底层，不需要业务开发人员干预。当一个定时任务执行完成后，通过配置还可以顺序性的触发另一

个定时任务。定时任务可以配置是否记录日志，如果记录则可以查看定时任务的执行情况。定时任务分组功能，可以让定时任务在指定的

服务器上执行，每个分组对应一个或几个服务器的IP地址。定时任务项目：matrix-quartz可以配置在基于Tomcat的web项目中，也可以

配置在Dubbo服务中，注意：无需在Spring配置文件中添加任何xml配置，后台界面全部都能帮你搞定。下面将展示相对应的功能截图：





8 matrix-route

分布式系统节点路由。系统高级功能，他能让矩阵系统拥有控制所有基于Matrix底层的服务节点的能力。一般公司在使用Dubbo的时候

9 matrix-rocket-mq

<https://www.processon.com/diagraming/5ddba84ce4b09e8b0b73d8fc>

当中台拥有了这些基础能力以后就可以开始组合与复用了，将他们映射到零售门店领域、线上电商领域、批发/供应链领域；各

中渠道：微信生态、抖音营销、支付宝、百度推广、H5站、安卓、IOS等等。