架构设计-核心思维案例

四个架构设计案例分析及其背后的架构师思维

写在前面

架构的本质是管理复杂性：

抽象、分层、分治和演化思维

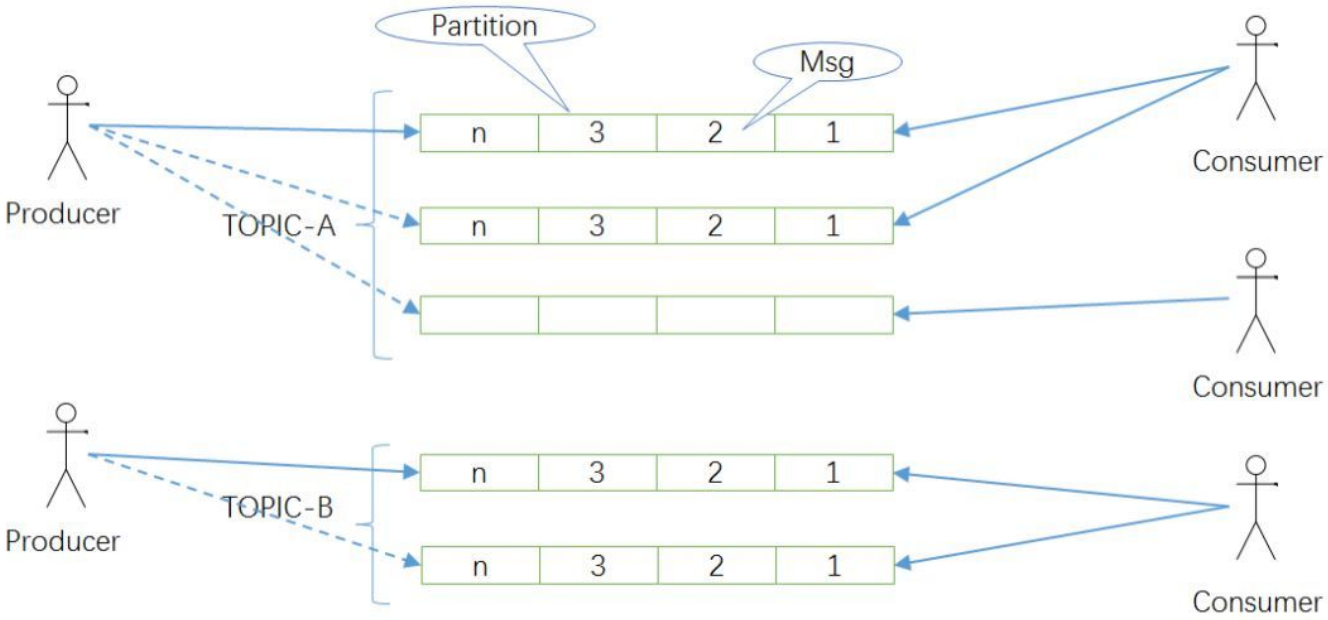
是我们工程师 / 架构师应对和管理复杂性的四种最基本武器。

在我之前写的文章 《优秀架构师必须掌握的架构思维》（点击标题查看原文） 中，我先介绍了抽象、分层、分治和演化这四种应对复杂性的基本武器。在本篇文章中，我会通过四个案例，讲解如何综合运用这些武器，分别对小型系统、中型系统、基础架构以及组织技术体系进行架构和设计。

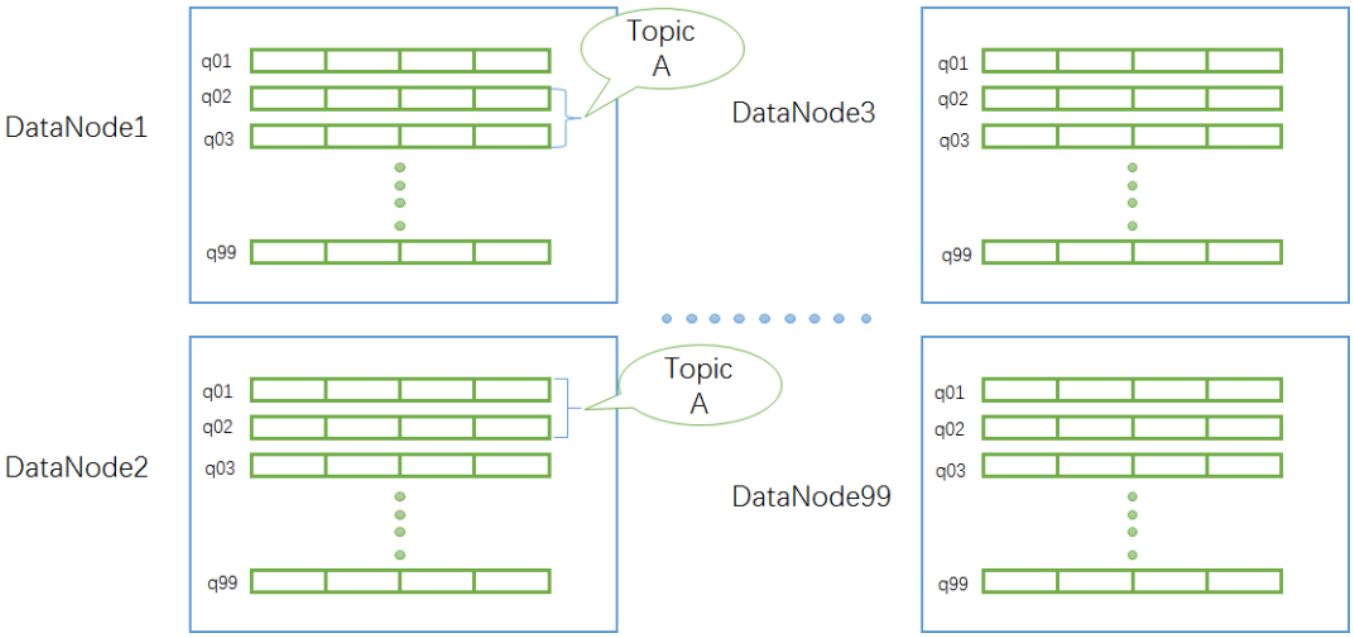
#小型系统案例：分布式消息系统

这个是一个真实生产化的消息系统案例，由 1 个架构师 +2 个高级工程师设计开发，第一期研发测试到上生产约 3 个月，目前该系统日处理消息量过亿。

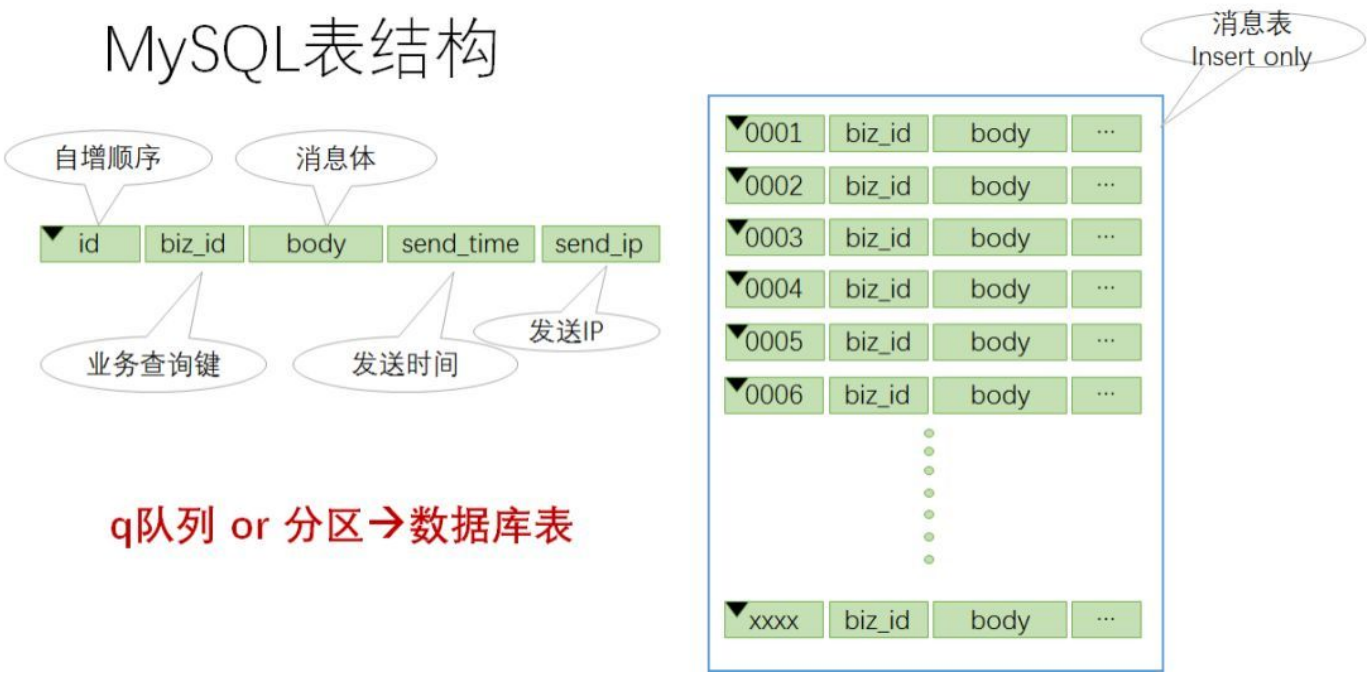
假定公司因为业务需要，要构建一套分布式消息系统 MQ，类似 Kafka 这样的，这个问题看起来很大很复杂，但是如果你抽丝剥茧，透过现象看本质，Kafka 这样的消息系统本质上是下图这样的抽象概念：



- 1. 队列其实就是类似数组一样的结构（用数组建模有个好处，有索引可以重复消费），里头存放消息 (Msg)，数组一头进消息，一头出消息；
- 2. 左边是若干生产者 (Producer)，往队列里头发消息；
- 3. 右边是若干消费者 (Consumer)，从队列里头消费消息；
- 4. 对于生产者和消费者来说，他们不关心队列实现细节，所以给队列一个更抽象的名字，叫主题 (Topic)；
- 5. 考虑到系统的扩容和分布式能力，一般一个主题由若干个队列组成，这些队列也叫分区 (Partition)，而且这些队列可能还是分布在不同机器上的，例如下图中 Topic A 的两个队列分布在 DataNode1 节点上，另外两个队列分布在 DataNode2 节点上，这样以后 Topic 可以按需扩容，DataNode 也可以按需增加。当然这些细节由 MQ 系统屏蔽，用户只关心主题，不关心底层实现。

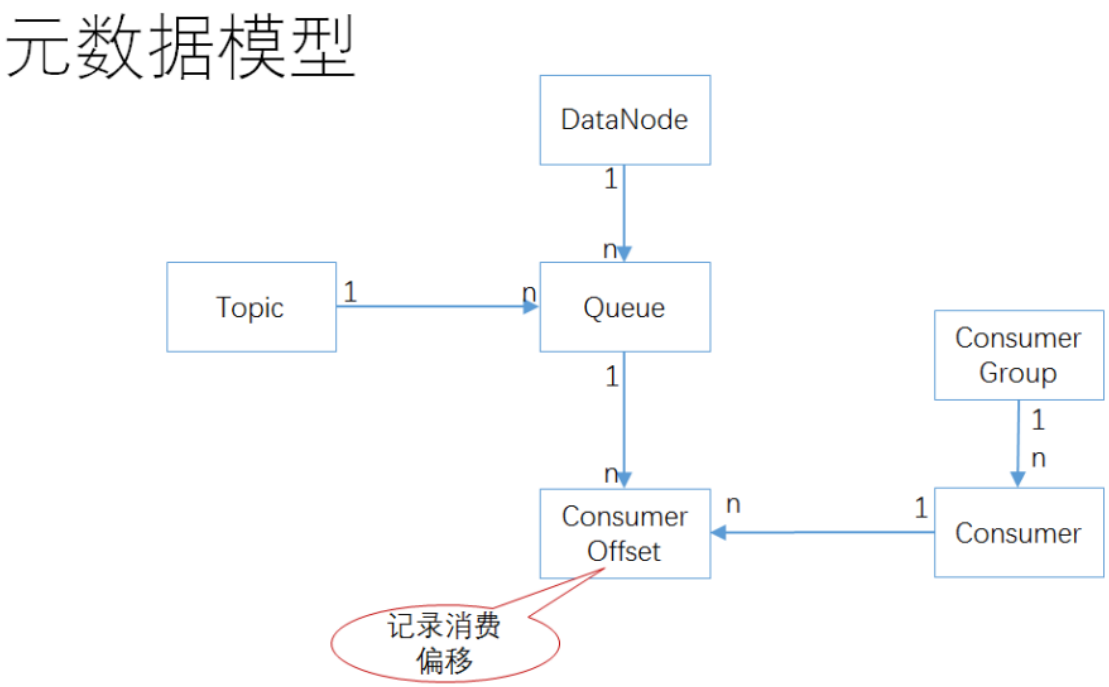


单个数组队列的建模是整个 MQ 系统的关键，我们知道 Kafka 使用 append only file 建模队列，存取速度快。假设我们要存业务数据需要更高可靠性，也可以用数据库表来建模数组队列，如下图所示：



一个队列 (或者一个分区) 对应一张数据库表，表中的一个记录就是一条消息，表采用自增 id，相当于数组索引。这张表是 insert only 的，且 MySQL 会自动对自增 id 建优化索引，没有其它索引，所以插入和按 id 查找速度都非常快。

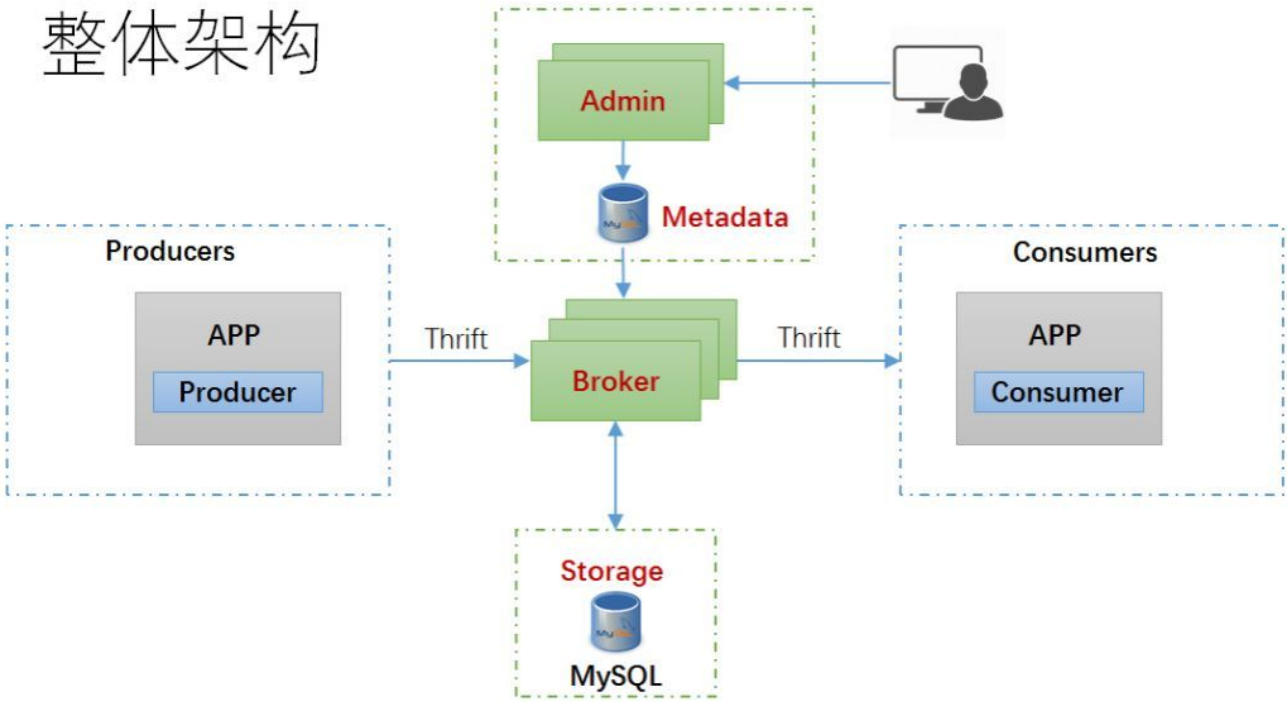
#下面是总体元数据模型：



- 1. 一个主题 Topic 对应若干个队列 Queue
- 2. 一个数据节点 DataNode 上可以住若干个队列 Queue
- 3. 消费者 Consumer 和队列 Queue 之间是多对多关系，通过消费者偏移 Consumer Offset 进行关联
- 4. 一个消费者组 Consumer Group 里头有若干个消费者 Consumer，它们共同消费同一个主题 Topic

#MQ抽象模型

至此，我们对 MQ 的抽象建模工作完成，下面的工作是将这个模型映射到具体实现，经过分解，整个系统由若干个子模块组成，每个子模块实现后拼装起来的 MQ 总体架构如下图所示：



- 1. Admin 模块管理数据库节点，生产者，消费者 (组)，主题，队列，消费偏移等元数据信息。
- 2. Broker 模块定期从 Admin 数据库同步元数据，接受生产者消息，按路由规则将消息存入对应的数据库表 (队列) 中；同时接受消费者请求，根据元数据从对应数据库表读取消息并发回消费者端。Broker 模块也接受消费者定期提交消费偏移。
- 3. Producer 接受应用发送消息请求，将消息发送到 Broker；
- 4. Consumer 从 Broker 拉取消息，供上层应用进一步消费；
- 5. 客户端和 Broker 之间走 Thrift over HTTP 协议，中间通过域名走 Nginx 代理转发；
- 6. 这个设计 Broker 是无状态，易于扩展。

#架构思维总结：

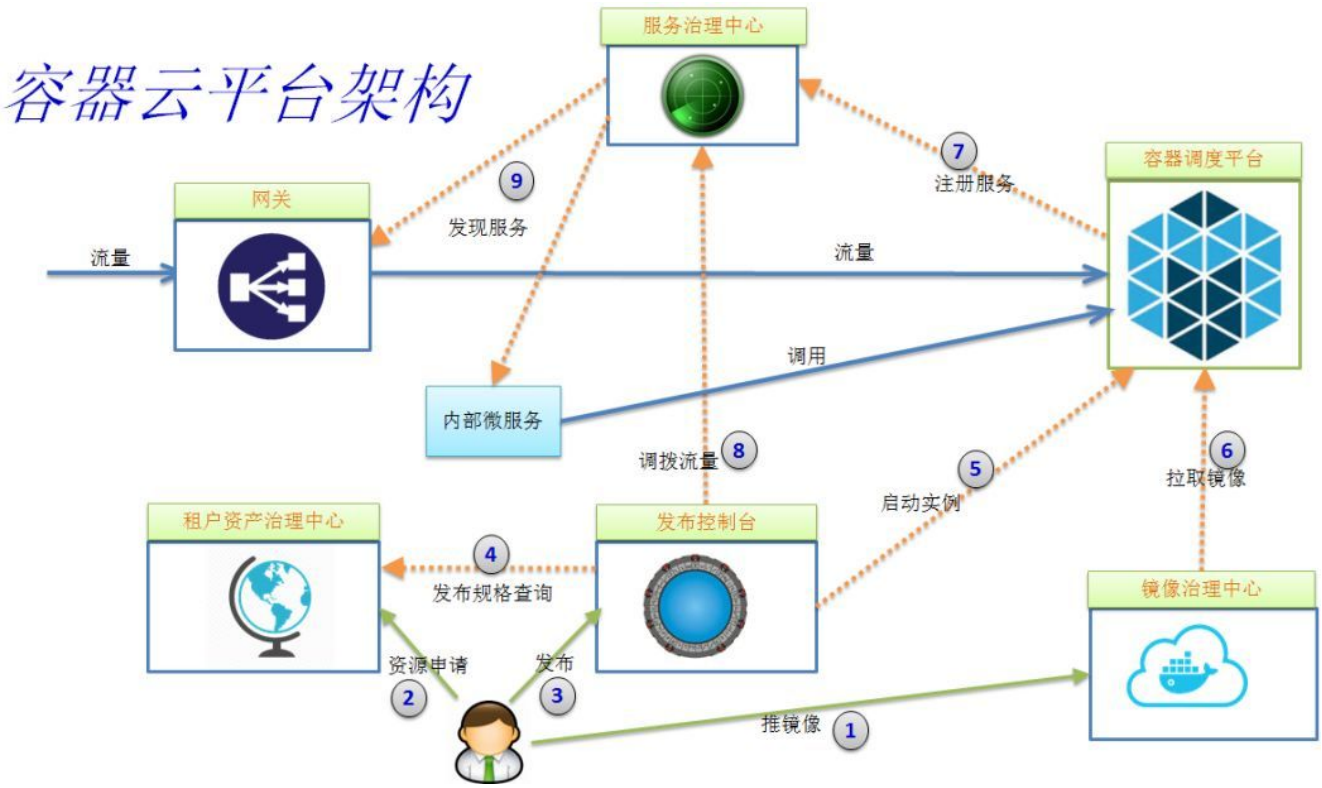
整个架构设计的思路体现了先总体抽象，再分解按模块抽象并实现，最后组合成完整的 MQ 系统，也就是 **抽象 + 分治**。这个 MQ 的实现工作量并不大，属于小型系统范畴，初期设计和开发由 1 个架构师 +2 个中高级工程师可以搞定。

在初期研发和上生产之后，根据用户的不断反馈，系统设计经过多次优化和调整，符合三分架构、七分演化的 **演化式架构**理念。目前该系统已经进入 V2 版本的架构和研发，其架构仍在持续演化当中，用户需求的多样性和对系统灵活性的更高要求，是系统架构演化的主要推动力。

#中型系统案例：容器云平台架构设计

这个也是一个实际研发中的案例。

目前不少技术组织在往 DevOps（研发运维一体化）研发模式转型，目标是支持业务持续创新和规模化发展。支持 DevOps 的关键是需要一套 DevOps 基础平台，这个平台可以基于容器云构建，我们把它称为容器云平台。这个问题很大很复杂，我基于近年在一线互联网的实战经验积累 + 广泛调研，设计了如下容器云平台的总体抽象架构：



#核心模块：

- 1. 集群资源调度平台：屏蔽容器细节，将整个集群抽象成容器资源池，支持按需申请和释放容器资源，物理机发生故障时能够实现自动故障转移 (fail over)。目前基于 Mesos 实现，将来可考虑替换为 K8S。
- 2. 镜像治理中心：基于 Docker Registry，封装一些轻量的治理功能，例如权限控制，审计，镜像升级流程（从测试到 UAT 到生产）治理和监控等。

- 3. 租户资源治理中心：类似 CMDB 概念，在容器云环境中，企业仍然需要对应用 app，组织 org，容器配额 quota 等相关信息进行轻量级的治理。
- 4. 发布控制台：面向用户的发布管理平台，支持发布流程编排。它和其它子系统对接交互，实现基本的应用发布能力，也实现如蓝绿，金丝雀和灰度等高级发布机制。
- 5. 服务注册中心：类似 Netflix Eureka，支持服务的注册和发现，流量的拉入拉出操作。
- 6. 网关：类似 Netflix Zuul 网关，接入外部流量并路由转发到内部的微服务，同时实现安全，限流熔断，监控等跨横切面功能。

#核心流程：

- 1. 用户或者 CI 系统对应用进行集成后生成镜像，将镜像推到镜像治理中心；
- 2. 用户在资产治理中心申请发布，填报应用、发布和配额等相关信息，然后等待审批通过；
- 3. 发布审批通过，开发人员通过发布控制台发布应用；
- 4. 发布控制台通过查询资产治理中心获取发布规格信息；
- 5. 发布控制台向容器资源调度平台发出启动容器实例指令；
- 6. 容器资源调度平台从镜像治理中心拉取镜像并启动容器；
- 7. 容器内服务启动后自注册到服务注册中心，并保持定期心跳；
- 8. 用户通过发布控制台调用服务注册中心接口进行流量调拨，实现蓝绿，金丝雀或灰度发布等机制；
- 9. 网关和内部微服务客户端定期同步服务注册中心上的路由表，将流量按负载均衡策略分发到服务实例上。

#架构思维总结：

经过抽象梳理，我们已经得到最终容器云平台的 6 大关键抽象模块和模块间交互流程，下一步就是围绕这 6 大核心模块组织 6 个小的研发团队，每个团队负责一个模块的设计和实现，待每个团队完成各自的模块，再将所有模块组合拼装起来，就能最终产出我们需要的容器云平台产品。整体架构设计思路还是 **抽象 + 分治**，只不过每个模块的抽象粒度更大，整个平台的规模也更大，需要投入的研发团队资源也更多，对架构师的抽象能力要求也更高。每个模块的技术负责人在研发各自的模块时，同样遵循 **抽象 + 分治**的思维方式，先做抽象架构，划分子模块，安排组员实现子模块，最后拼装组合成完整模块。

由于这个平台规模较大较复杂，目前已经投入了近两个季度的时间，做第一期架构设计和研发，目前还没有完全生产化。在第一期过程中，随着对问题域的理解不断深入，架构设计经过多次调整，目前架构趋于稳定，已经进入预上线期。在后续生产落地过程中，仍然需要根据用户的反馈，借助进化的力量不断地调整和优化架构。这个符合 **演化式架构**的思路。

#大型系统案例：微服务基础架构

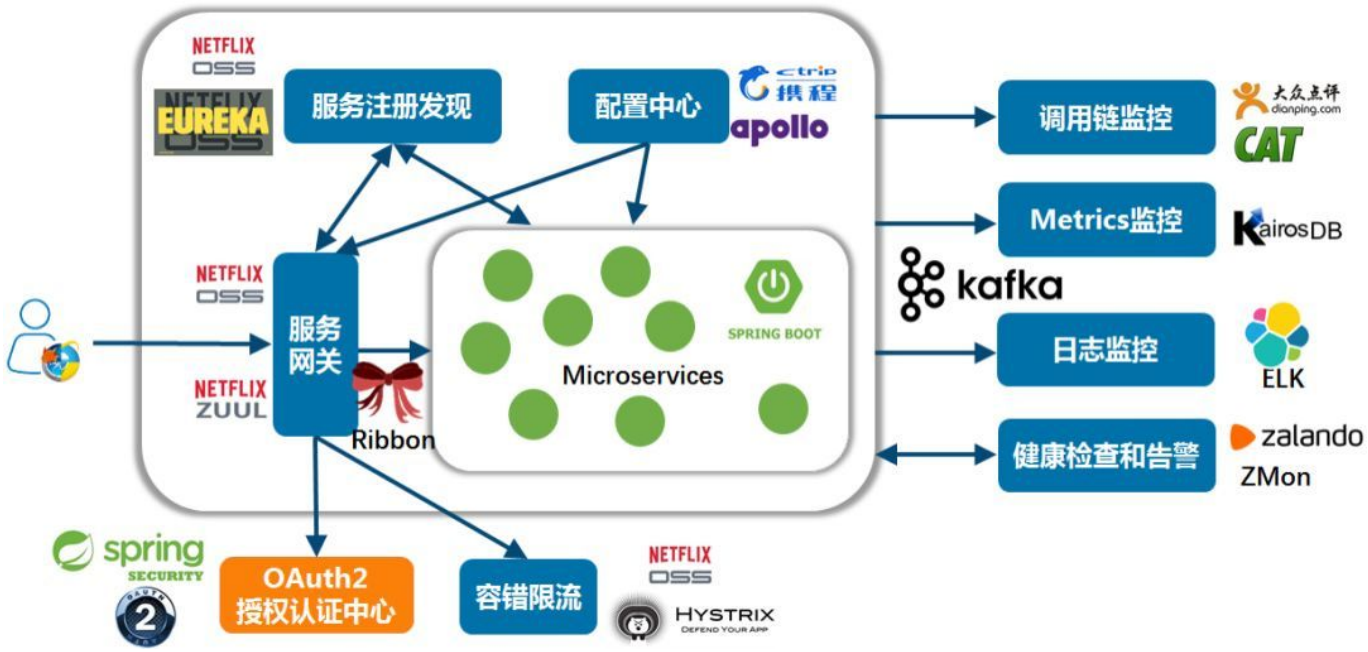
微服务架构是近年很多企业技术架构转型的趋势，实际上，微服务架构可以抽象分解为一个 **两层架构**：上层是微服务业务架构，下层是微服务基础架构。上层业务架构由于每个企业的业务场景各不相同，所以一般很难通用化，大多企业都是定制自研。而下层基础架构由于近年业界实践的不断沉淀，已经比较通用化和模块化，其中的核心模块一般不需要自己重造轮子，重用那些在一线互联网公司已经落地并开源出来的产品就可以了。

Netflix 是一家伟大的科技公司，它内部的基础架构团队很牛，或者说抽象能力非常强，把一些核心微服务基础组件都以模块化方式开源出来了，使得其它公司只需组合拼装这些组件就可以快速搭建微服务架构，可以说 Netflix 将整个行业的技术水平提升了一个层次。

#微服务架构8 大模块包括

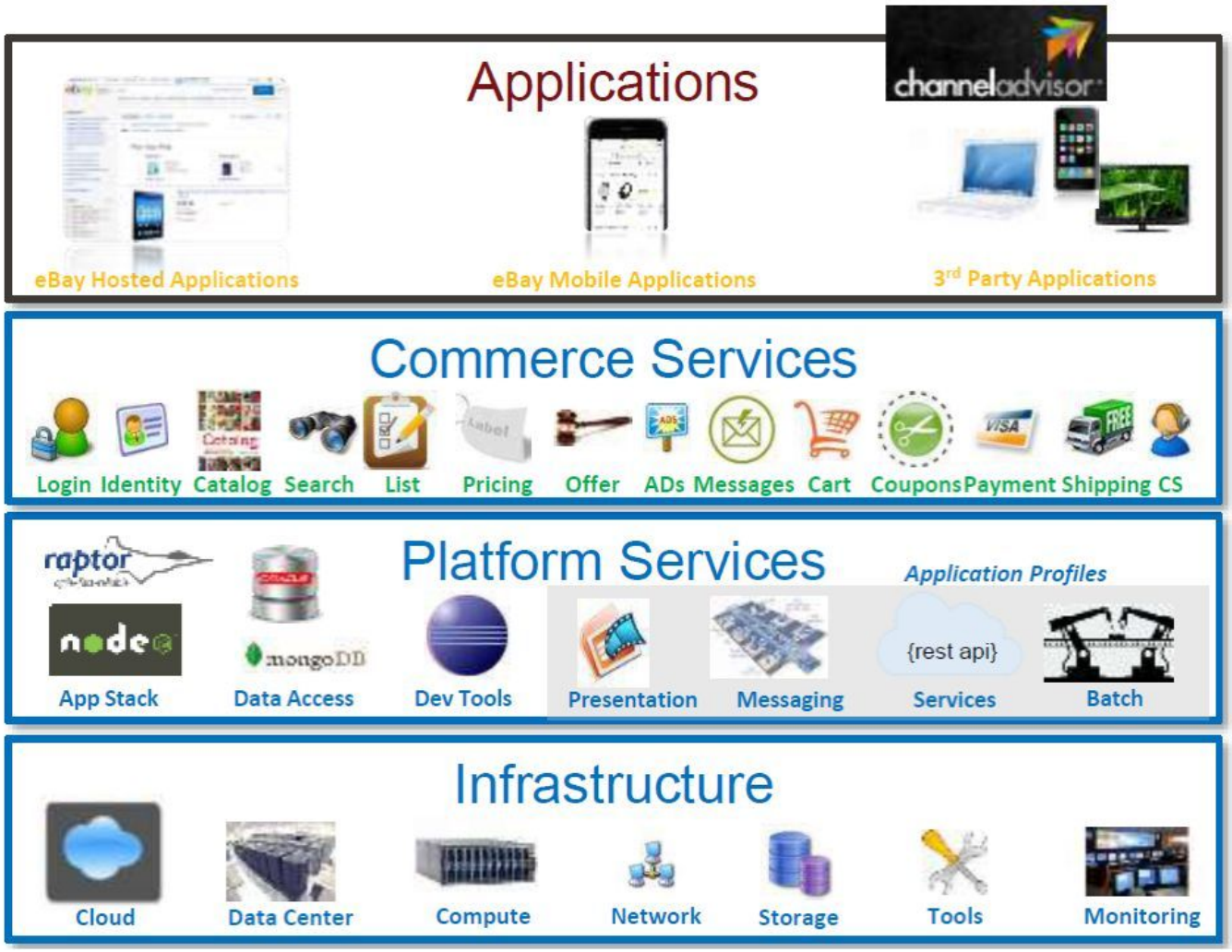
- 服务认证授权中心 Spring Security OAuth2
- 服务配置中心 Apollo
- 服务调用链监控 CAT
- 服务网关 Zuul
- 服务限流熔断 Hystrix/Turbine
- 服务注册发现和软路由 Eureka/Ribbon
- 服务时间序列监控 KairosDB
- 服务监控告警 ZMon

整体拼装起来的微服务基础架构如下图所示，这个架构是经过实践落地的，可以作为一线企业搭建微服务基础架构的参考：



#技术体系架构案例

在企业的整个技术体系架构层面，最基本的思考方式还是 **抽象 + 分治**，只不过问题域更大更复杂，还涉及到组织和业务架构，所以一般还要增加 **分层**的维度来解决，下图是 2016 年的 eBay 技术体系架构（图片来自文末参考链接）：



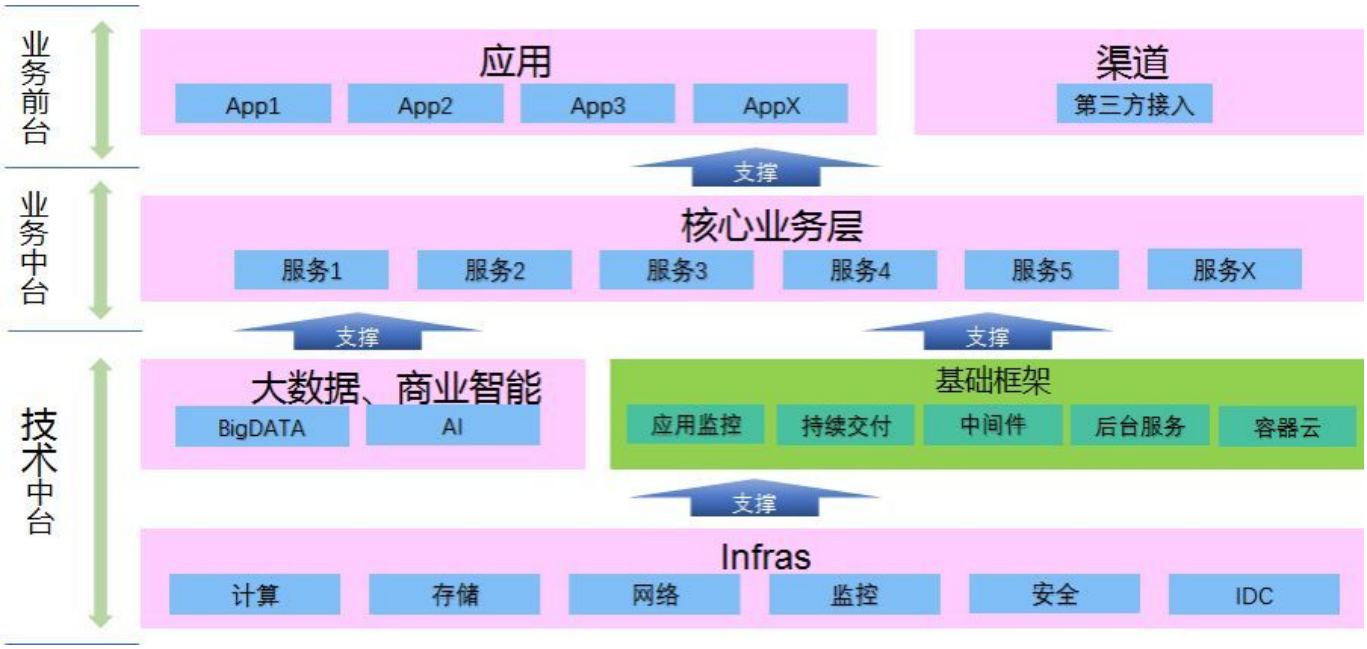
我最早看到这个架构图是在 2008 年左右的一次 all hands meeting 上（当时我还在 eBay 中国研发中心做工程师），也就是说大致在 2008 年左右，eBay 就已经有比较清晰的，以分层方式组织的技术体系架构。eBay 当时把它的系统称为电子商务操作系统，因为据说整个系统的代码量超过 Windows 7 操作系统的代码量。

#eBay 架构分为清晰的四个抽象层次：

- Infrastructure： 底层基础设施，包括云计算，数据中心，计算 / 网络 / 存储，各种工具和监控等，国内公司一般把这一层称为运维层。
- Platform Services： 平台服务层，主要是一些框架中间件服务，包括应用和服务框架，数据访问层，表示层，消息系统，任务调度和开发者工具等等，国内公司一般把这一层称为基础框架或基础架构层。
- Commerce Services： 电商服务层，eBay 作为电子商务平台多年沉淀下来的核心领域服务，相当于微服务业务层，包括登录认证，分类搜索，购物车，送货和客服等等。

- Applications： 应用层，也称用户体验 + 渠道层，包括 eBay 主站，移动端 app，第三方接入渠道等。

在吸收了 eBay 技术体系架构的基础上，也吸收了一些阿里巴巴中台战略的思想，同时融合近年的一些业界趋势（比如大数据 /AI），抽象出一个更通用的分层技术体系架构，可以作为互联网公司技术体系架构的一般性参考，如下图所示：



顺便提一下，近年阿里提出的所谓大中台，小前台战略，其实就要强化技术中台 + 业务中台，中台做大做强了，业务前台才可以更轻更灵活的响应业务需求的变化。

#架构思维案例总结

- 1. 架构的本质是管理复杂性，抽象、分层、分治和演化思维是架构师征服复杂性的四种根本性武器。
- 2. 掌握了抽象、分层、分治和演化这四种基本的武器，你可以设计小到一个类，一个模块，一个子系统，或者一个中型的系统，也可以大到一个公司的基础平台架构，微服务架构，技术体系架构，甚至是组织架构，业务架构等等。
- 3. 架构设计不是静态的，而是动态演化的。只有能够不断应对环境变化的系统，才是有生命力的系统。所以即使你掌握了抽象、分层和分治这三种基本思维，仍然需要演化式思维，在完成系统的初步架构设计之后，后续借助反馈和进化的力量推动架构的持续演进。
- 4. 架构师在关注技术，开发应用的同时，需要定期梳理自己的架构设计思维，积累时间长了，你看待世界事物的方式会发生根本性变化，你会发现我们生活其中的世界，其实也是在抽象、分层、分治和演化的基础上构建起来的。另外架构设计思维的形成，会对你的系统架构设计能力产生重大影响。可以说对抽象、分层、分治和演化掌握的深度和灵

> 活应用的水平，直接决定架构师所能解决问题域的复杂性和规模大小，是区分普通应用型架构师和平台型 / 系统型架构师的一个分水岭。

参考资料

MicroServices at eBay

https://www.slideshare.net/kasun04/microservices-at-ebay

#参考文章

- https://www.sohu.com/a/232016795_355140