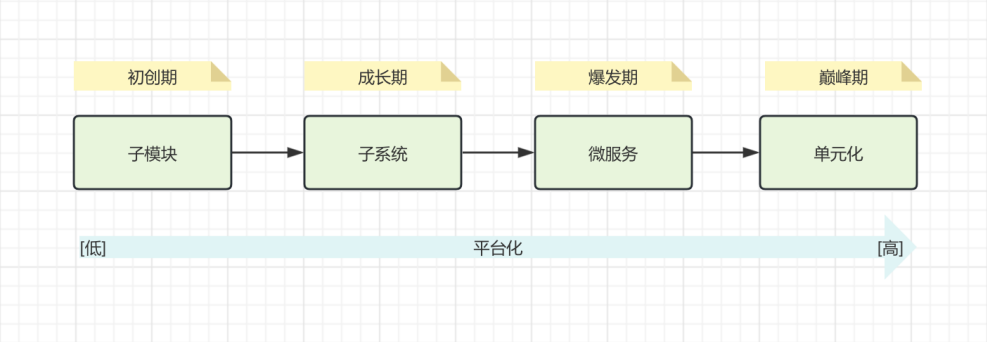
【订单领域】你的订单系统如何从单体项目，演进微服务架构？主要解决什么问题?

来自： 芋道快速开发平台 Boot + Cloud

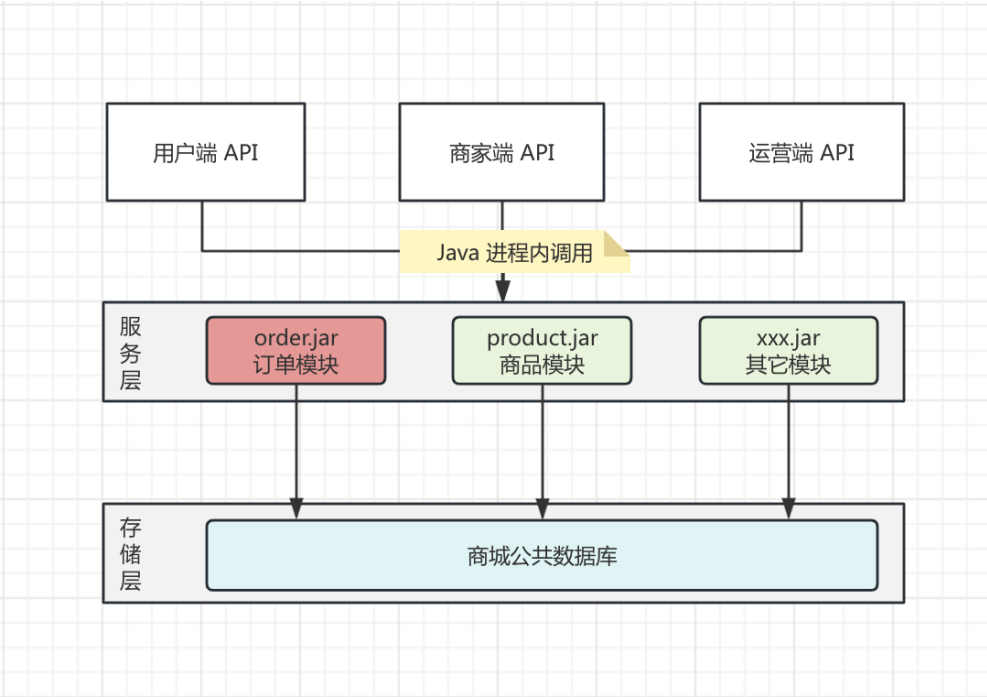
芋道源码 2023年08月02日 07:23

订单系统的演进路线是：子模块 ⇒ 子系统 ⇒ 微服务 ⇒ 单元化。
在这个过程中，平台化从低到高也是个重要发展方向。



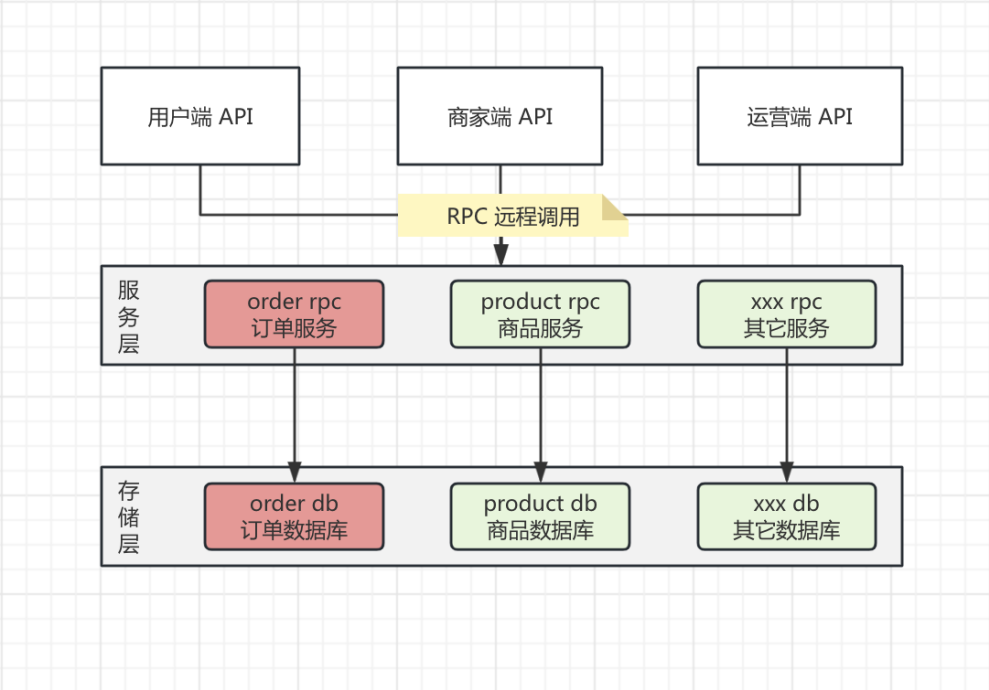
单元化、平台化比较复杂，值得单独开个帖子，所以本帖只聊从单体项目到微服务架构的演进路线。

V1.0：子模块



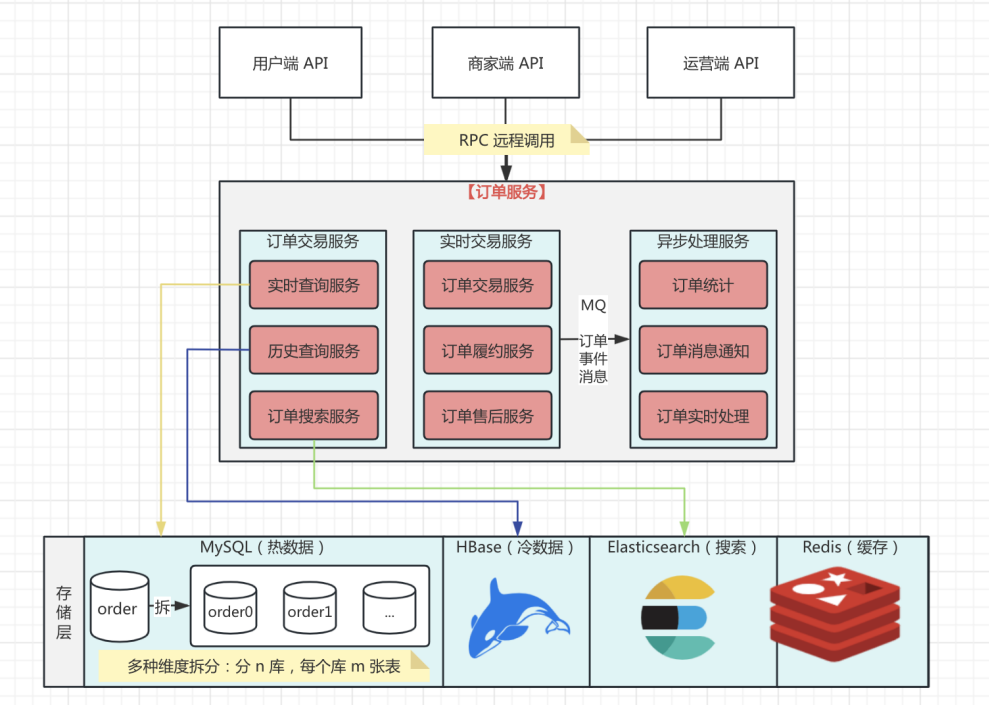
① **业务背景**：初创期【天使轮】订单量比较小，日均 100+ 单
② **核心问题**：要求快速迭代、快速试错，频繁发版是常态
③ **技术架构**：【类似 ruoyi-vue-pro 项目的 trade 模块】
1、单体架构：部署简单，维护成本低
2、共存进程：订单作为一个 jar 包，内嵌在后端系统
3、共享存储：订单共享使用商城公用数据库

V2.0：子系统



① **业务阶段**：成长期【A 轮融资】订单量开始增长，日均 2w+ 单
② **核心问题**：业务复杂度逐渐增大，逻辑耦合导致订单不稳定
③ **技术架构**：【类似 yudao-cloud 项目的 trade 服务】
1、独立服务：订单拆分独立 order 服务，支持独立开发、部署上线
2、独立存储：订单拆分独立数据库，资源隔离，避免和其它业务相互影响

V3.0：微服务



① **业务阶段**：爆发期【B/C 轮融资】，订单量开始腾飞，日均 100w+ 单
② **核心问题**：系统面临瓶颈，服务和存储的读写压力过大，线上事故不断
③ **技术架构**：
1、微服务化：
- 基于职责：拆分订单交易、履约、售后等服务

- 基于读写：拆分“查询类”服务（订单查询服务等等）、”写入类”服务（实时交易服务等等）

2、存储分片：

- 分库分表：支持更大量数据的存储、更高 QPS 的读写
- 冷热分离：热数据（近 90 天）存储 MySQL，冷数据（超 90 天）归档 HBase

3、逻辑异步化：核心逻辑同步处理，非核心逻辑异步解耦

ps：关于海量订单的存储，也会单独再开个帖子，单独讲解 MySQL、Redis、HBase、Elasticsearch 存储方案。

题外话：分享这个话题的目的，大家在日常做架构设计，或者被面试官问到架构演进的问题时，要去思考可以分成几个阶段，每个阶段的数据规模是多大，面临的核心问题是什么，需要做什么样的技术改造，每个改造的关键点是什么。

另外，大家从 V2.0 和 V3.0 的演进中，思考微服务应该基于什么原则拆分？应该拆分的多细？SOA 和微服务有什么区别？

仅供参考，欢迎评论区留言，一起讨论。

知识星球