乐字节教育高级架构课程

正所谓“授人以鱼不如授人以渔”，你们想要的 Java 学习资料来啦！

不管你是学生，还是已经步入职场的同行，希望你们都要珍惜眼前的学习机会，奋斗没有终点，知识永不过时。

扫描下方二维码即可领取

乐字节晓啡

乐字节官方交流群

shop商城介绍

电商行业的模式：

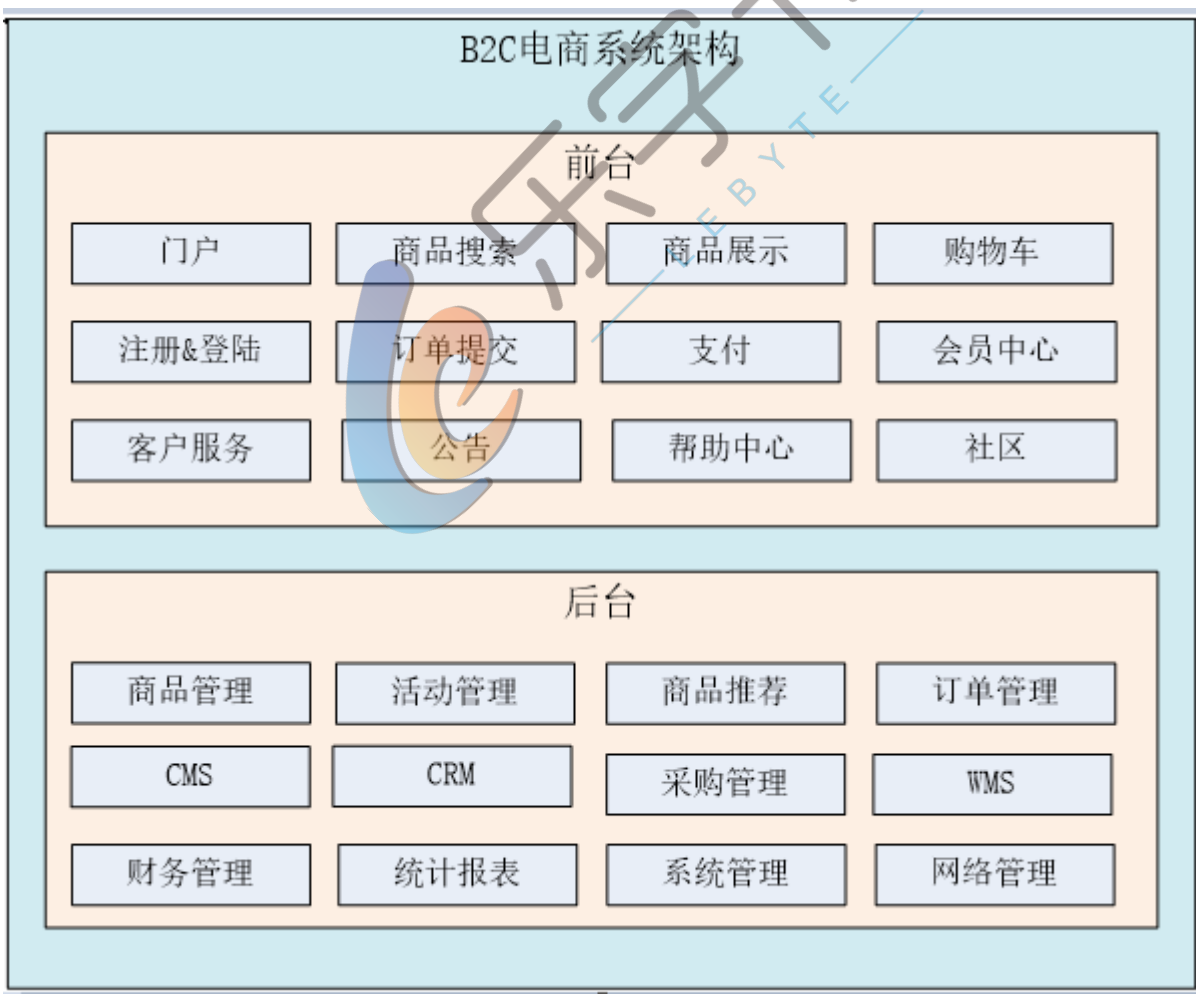
- B2B：企业到企业，商家到商家。代表：阿里巴巴、慧聪网
- B2C：商家到客户。代表：京东、淘宝商城（B2B2C）
- C2C：客户到客户。淘宝集市
- O2O：线上到线下

shop商城的模式

shop网上商城是一个综合性的B2C平台，类似京东商城、天猫商城。会员可以在商城浏览商品、下订单，以及参加各种活动。

管理员、运营可以在平台后台管理系统中管理商品、订单、会员等。客服可以在后台管理系统中处理用户的询问以及投诉。

功能模块



功能描述

后台管理系统：管理商品、订单、类目、商品规格属性、用户管理以及内容发布等功能。

前台系统：用户可以在前台系统中进行注册、登录、浏览商品、首页、下单等操作。

会员系统：用户可以在该系统中查询已下的订单、收藏的商品、我的优惠券、团购等信息。

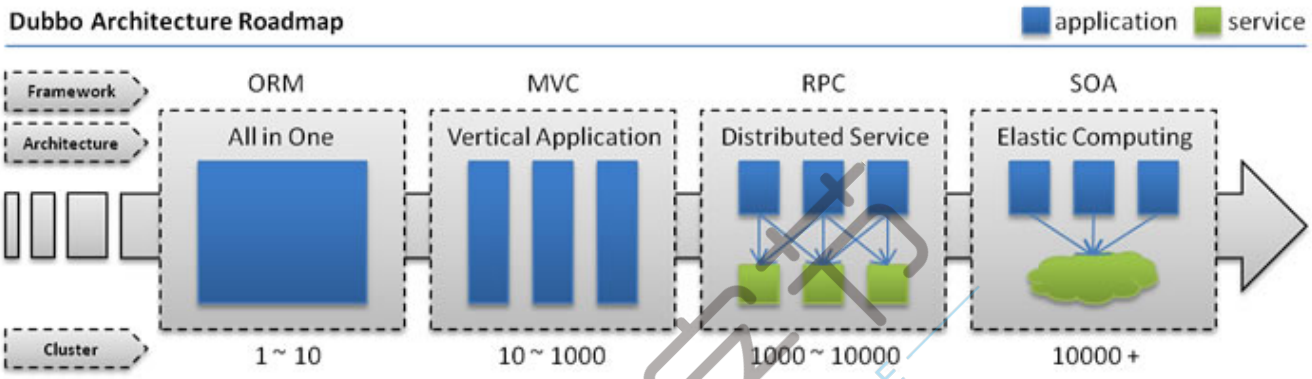
订单系统：提供下单、查询订单、修改订单状态、定时处理订单。

搜索系统：提供商品的搜索功能。

单点登录系统：为多个系统之间提供用户登录凭证以及查询登录用户的信息。

技术架构

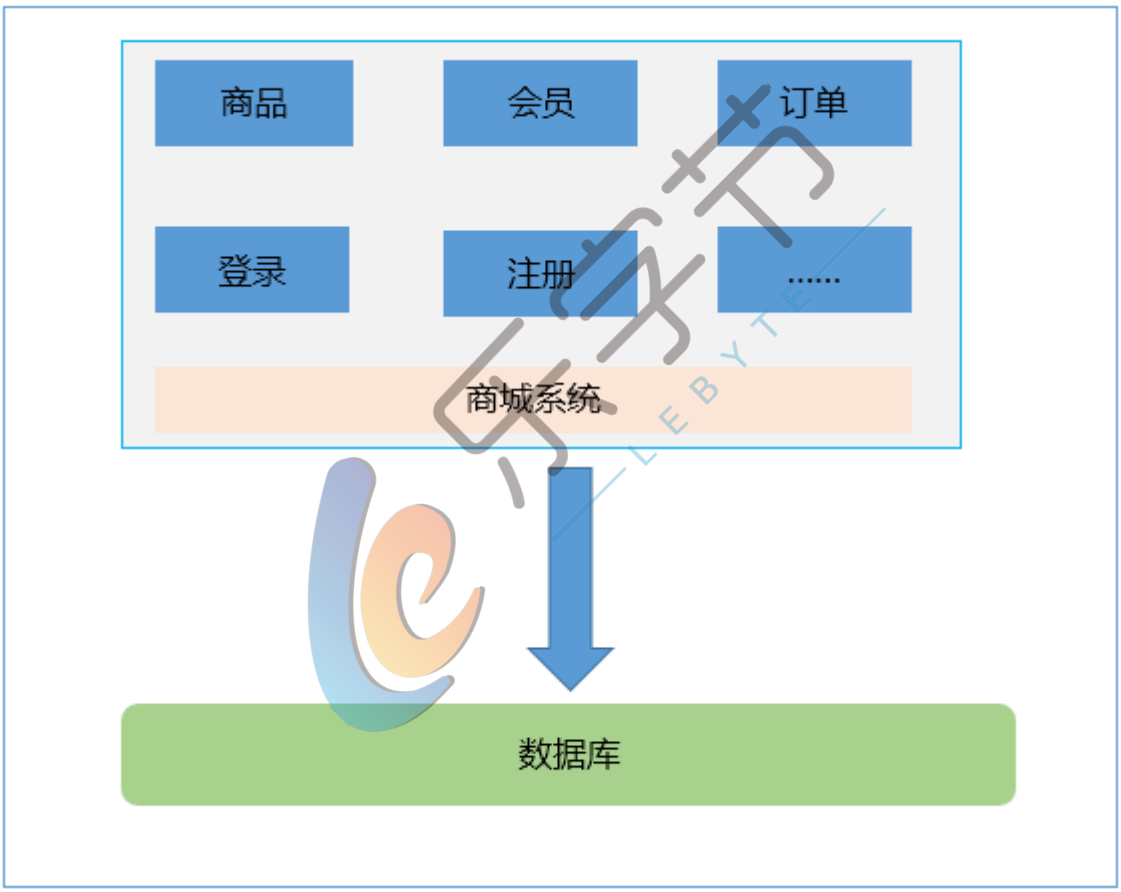
随着互联网的发展，网站应用的规模不断扩大，常规的垂直应用架构已无法应对，分布式服务架构以及流动计算架构势在必行，亟需一个治理系统确保架构有条不紊的演进。



- **单一应用架构**
  - 当网站流量很小时，只需一个应用，将所有功能都部署在一起，以减少部署节点和成本。
  - 此时，用于简化增删改查工作量的 **数据访问框架(ORM)** 是关键。
  - 缺点：随着应用功能的增多，代码量越来越大，越来越难维护，那怎么解决代码一体化的问题？
- **垂直应用架构**
  - 当访问量逐渐增大，单一应用增加机器带来的加速度越来越小，将应用拆成互不相干的几个应用，以提升效率。
  - 此时，用于加速前端页面开发的 **Web框架(MVC)** 是关键。
  - 缺点：垂直架构中相同逻辑代码需要不断的复制，不能复用。每个垂直模块都相当于一个独立的系统。
- **分布式服务架构**
  - 当垂直应用越来越多，应用之间交互不可避免，将核心业务抽取出来，作为独立的服务，逐渐形成稳定的服务中心，使前端应用能更快速的响应多变的市场需求。
  - 此时，用于提高业务复用及整合的 **分布式服务框架(RPC)** 是关键。
  - 缺点：服务越来越多，需要管理每个服务的地址，调用关系错综复杂，难以理清依赖关系，服务状态难以管理，无法根据服务情况动态管理。
- **流动计算架构**
  - 当服务越来越多，容量的评估，小服务资源的浪费等问题逐渐显现，此时需增加一个调度中心基于访问压力实时管理集群容量，提高集群利用率。
  - 此时，用于提高机器利用率的 **资源调度和治理中心(SOA)** 是关键。
  - 缺点：服务间会有依赖关系，一旦某个环节出错会影响较大，服务关系复杂，运维、测试部署困难，不符合DevOps思想。
- **微服务架构**：
  - 单一职责：微服务中每一个服务都对应唯一的业务能力，做到单一职责

- 微：微服务的服务拆分粒度很小，例如一个用户管理就可以作为一个服务。每个服务虽小，但“五脏俱全”。
- 面向服务：面向服务是说每个服务都要对外暴露服务接口API。并不关心服务的技术实现，做到与平台和语言无关，也不限定用什么技术实现，只要提供Rest的接口即可。
- 自治：自治是说服务间互相独立，互不干扰
-
  - 团队独立：每个服务都是一个独立的开发团队，人数不能过多。
  - 技术独立：因为是面向服务，提供Rest接口，使用什么技术没有别人干涉
  - 前后端分离：采用前后端分离开发，提供统一Rest接口，后端不用再为PC、移动端开发不同接口
  - 数据库分离：每个服务都使用自己的数据源
  - 部署独立，服务间虽然有调用，但要做到服务重启不影响其它服务。有利于持续集成和持续交付。每个服务都是独立的组件，可复用，可替换，降低耦合，易维护 Docker部署服务

传统架构



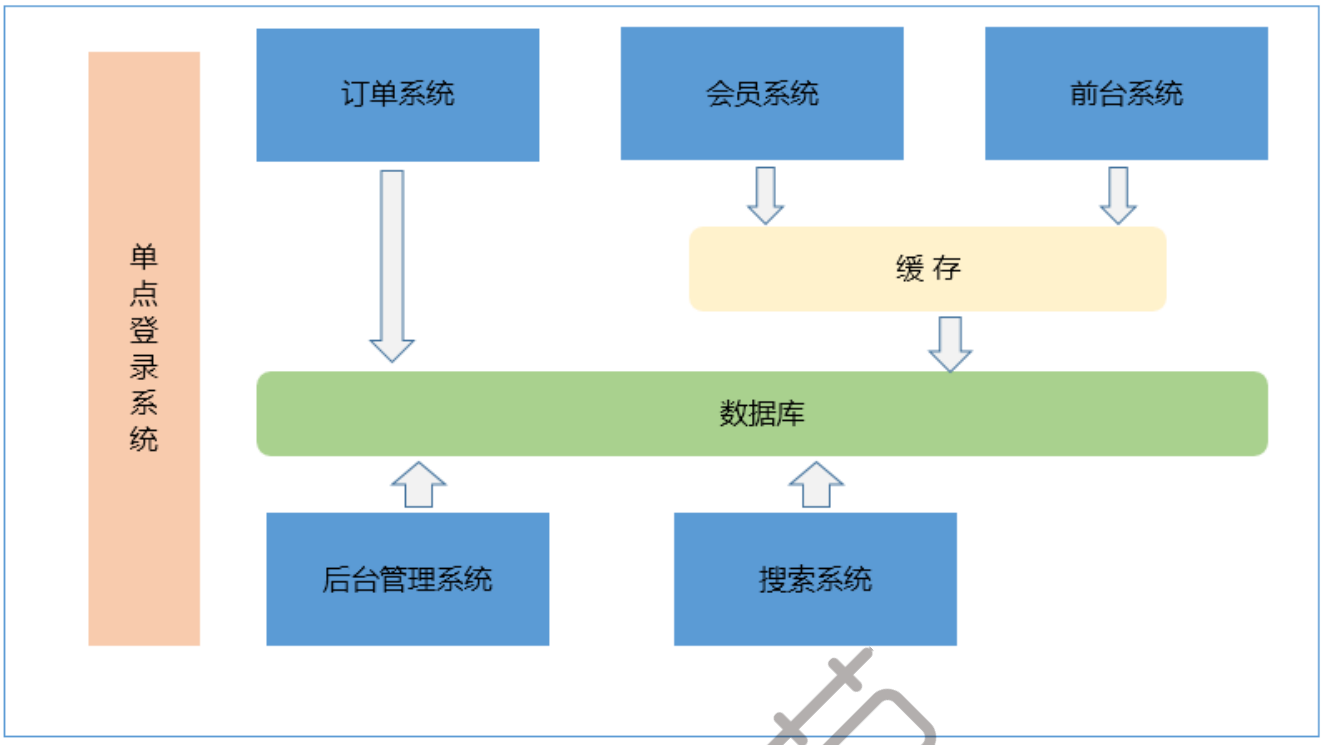
思考：有什么问题?

1. 模块之间耦合度太高，其中一个升级其他都得升级
2. 开发困难，各个团队开发最后都要整合一起
3. 系统的扩展性差
4. 不能灵活的进行分布式部署。

解决方法：系统拆分

优点：把模块拆分成独立的工程，单点运行。如果某一个点压力大可以对这一个点单独增加配置。其他的点不受影响。

分布式架构



分布式架构：把系统按照模块拆分成多个子系统。

优点：

1. 把模块拆分，使用接口通信，降低模块之间的耦合度。
2. 把项目拆分成若干个子项目，不同的团队负责不同的子项目。
3. 增加功能时只需要再增加一个子项目，调用其他系统的接口就可以。
4. 可以灵活的进行分布式部署。

缺点：

1. 系统之间交互需要使用远程通信，接口开发增加工作量。

最终架构图