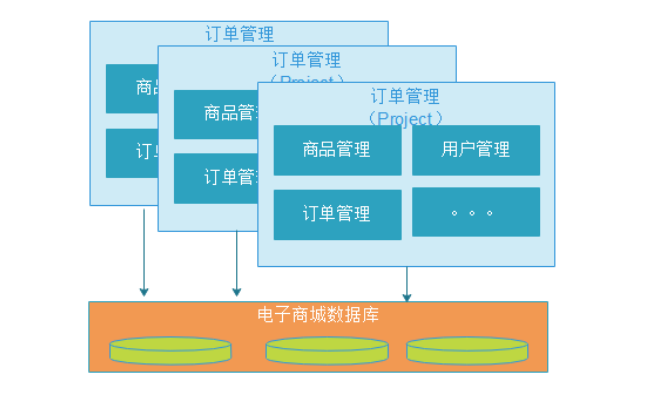
# 软件架构的演进

软件架构的发展经历了从单体架构、垂直架构、SOA架构到微服务架构的过程。

## 1.1.单体架构

单体架构Web应用程序发展的早期，大部web工程师将所有的功能模块打包到一起并放在一个web容器中运行，所有功能模块使用同一个数据库。 下图是一个单体架构的电商系统：



### 1.1.1.特点：

1、所有的功能集成在一个项目工程中。

2、所有的功能打在一个war包部署到服务器。

3、通过部署应用集群和数据库集群来提高系统的性能。

### 1.1.2.优点：

1、项目架构简单，前期开发成本低，周期短，小型项目的首选。

2、开发效率高，模块之间交互采用本地方法调用。

3、容易部署，运维成本小，直接打包为一个完整的包，拷贝到web容器的某个目录下即可运行。

4、容易测试：**IDE（集成开发环境）**都是为开发单个应用设计的、容易测试——在本地就可以启动完整的系统。

### 1.1.3.缺点：

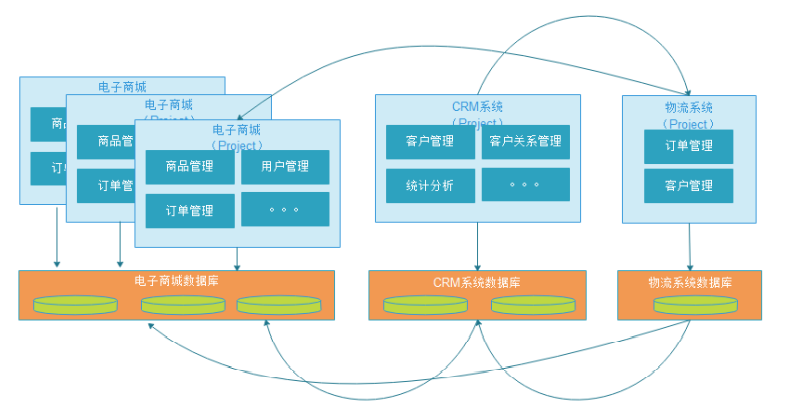
1、全部功能集成在一个工程中，对于大型项目不易开发、扩展及维护。

2、版本迭代速度逐渐变慢，修改一个地方就要将整个应用全部编译、部署、启动，开发及测试周期过长。

3、无法按需伸缩，通过集群的方式来实现水平扩展，无法针对某业务按需伸缩。

## 分布式架构

针对单体架构的不足，为了适应大型项目的开发需求，许多公司将一个单体系统按业务垂直拆分为若干系统，系统之间通过**网络交互**来完成用户的业务处理，每个系统可分布式部署，这种架构称为分布式架构。



### 特点：

1、按业务垂直拆分成一个一个的单体系统，此架构也称为垂直架构。

2、系统与系统之间的**存在数据冗余，耦合性较大，**如上图中三个项目都存在客户信息。 3、系统之间的接口多为实现数据同步，如上图中三个项目要同步客户信息。

### 1.2.2.优点：

1、通过垂直拆分，每个子系统变成小型系统，功能简单，前期开发成本低，周期短。

2、每个子系统可按需伸缩。

3、每个子系统可采用不同的技术，有利于技术团队的发展。

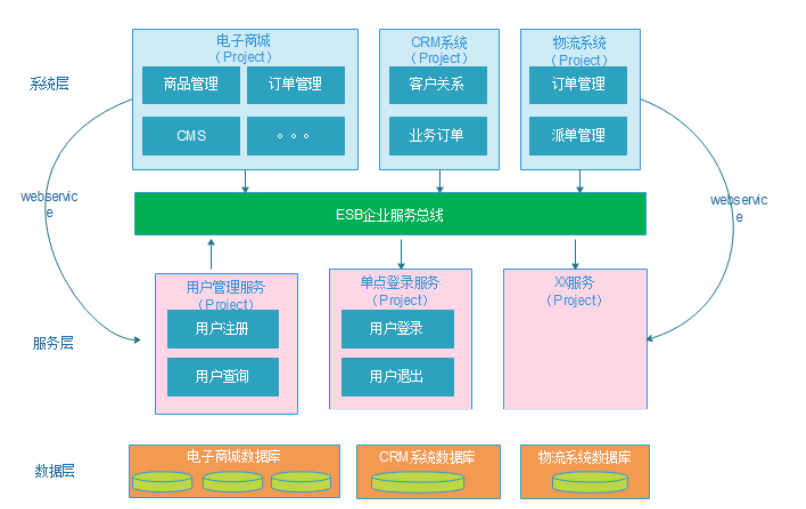
### 1.2.3.缺点：

1、**子系统之间存在数据冗余、功能冗余，耦合性高。**

2、**按需伸缩粒度不够**，对同一个子系统中的不同的业务无法实现，比如订单管理和用户管理。

## 1.3.SOA架构

SOA是一种面向服务的架构，基于分布式架构，它将不同业务功能按服务进行拆分，并通过这些服务之间定义良好的接口和协议联系起来。



### 1.3.1.特点：

1、基于SOA的架构思想，将重复公用的功能抽取为组件，以服务的方式向各各系统提供服务。

2、各个系统与服务之间采用webservice、**RPC（远程过程调用）**等方式进行通信。

3、ESB企业服务总线作为系统与服务之间通信的桥梁。

### 1.3.2.优点：

1、将重复的功能抽取为服务，提高开发效率，提高系统的**可重用性、可维护性**。

2、可以针对不同服务的特点按需伸缩。

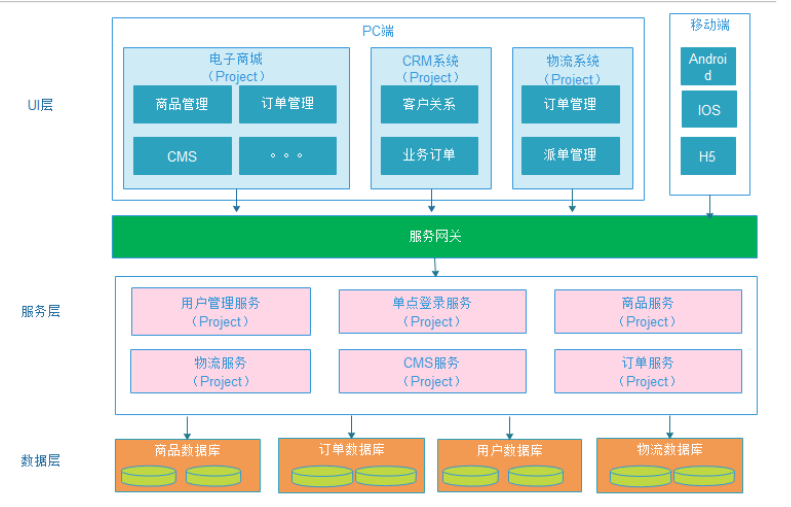
3、采用ESB减少系统中的接口耦合。

### 1.3.3.缺点：

1、系统与服务的界限模糊，会导致抽取的服务的粒度过大，系统与服务之间耦合性高。 2、虽然使用了ESB，但是服务的接口协议不固定，种类繁多，不利于系统维护。

## 1.4.微服务架构

基于SOA架构的思想，为了满足移动互联网对大型项目及多客户端的需求，对服务层进行细粒度的拆分，所拆分的每个服务只完成某个特定的业务功能，比如订单服务只实现订单相关的业务，用户服务实现用户管理相关的业务等等，**服务的粒度很小，所以称为微服务架构**。



### 1.4.1.特点：

1、服务层按业务拆分为一个一个的微服务。

2、微服务的职责单一。

3、微服务之间采用RESTful**（REST 指的是一组架构**[**约束条件**](https://baike.baidu.com/item/%E7%BA%A6%E6%9D%9F%E6%9D%A1%E4%BB%B6?fromModule=lemma_inlink)**和原则。满足这些约束条件和原则的应用程序或设计就是 RESTful。）**、RPC等轻量级协议传输。

4、有利于采用前后端分离架构。

### 1.4.2.优点：

1、服务拆分粒度更细，有利于资源重复利用，提高开发效率。

2、可以更加精准的制定每个服务的优化方案，按需伸缩。

3、适用于互联网时代，**产品迭代周期更短**。

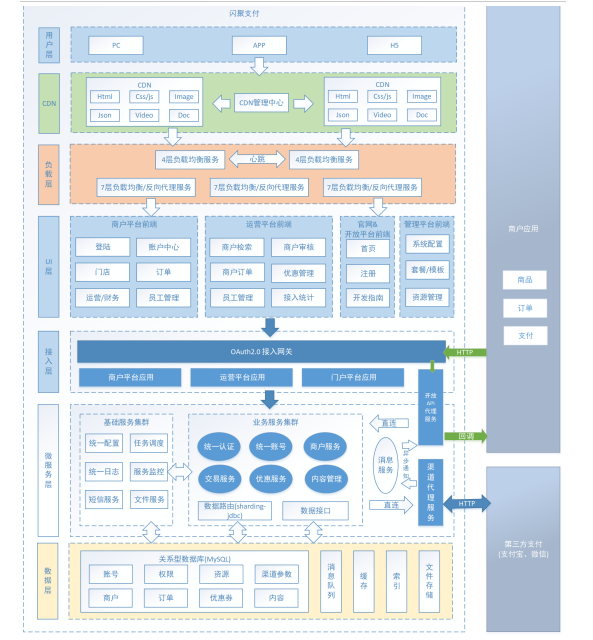
### 1.4.3.缺点：

1、开发的复杂性增加，因为一个业务流程需要多个微服务通过网络交互来完成。

2、微服务过多，服务治理成本高，不利于系统维护。

# 2.宜亩支付项目技术架构

宜亩支付采用当前流行的前后端分离架构开发，由用户层、UI层、微服务层、数据层等部分组成，为PC、H5等客 户端用户提供服务。下图是系统的技术架构图：



业务流程举例：

1、用户可以通过PC、手机等客户端访问宜亩支付。

2、系统应用**CDN（内容分发网络）**技术，对一些图片、CSS、视频等资源从CDN调度访问。

3、所有的请求全部经过负载均衡器。

4、首先请求UI层，渲染用户界面。

5、商户通过平台进行注册和企业认证，UI层通过网关请求服务层，服务层完成业务处理后将数据持久化到数据层。

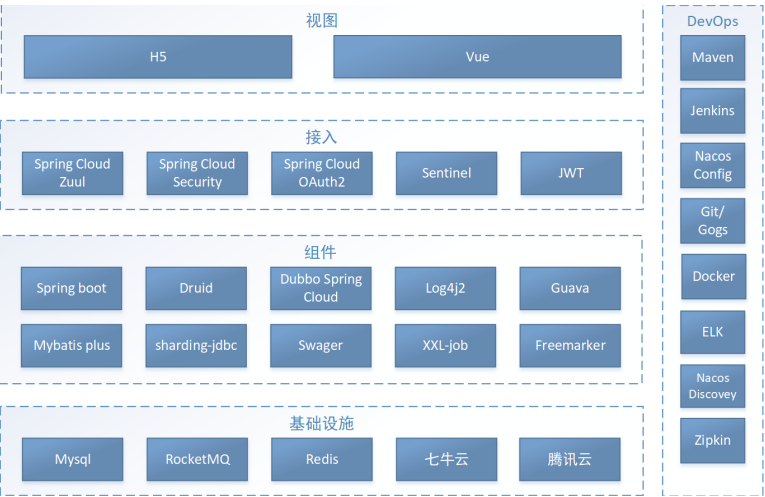
6、平台运营人员对商户信息进行审核，其系统执行流程和商户注册过程一致，UI层请求服务层业务处理，服务层查询数据层并将数据持久化到数据层。

各模块说明如下：



# 3.技术栈

下图是项目技术架构的简图，通过简图了解项目所使用的技术栈。



重点了解微服务技术栈：

宜亩支付服务端基于Spring Boot构建，采用**Spring Cloud Alibaba**微服务架构。

1）基础设施 业务数据持久化采用MySQL，数据缓存采用Redis，采用RocketMQ的事务消息机制完成部分场景下的分布式事务控制，采用第三方云平台完成文件上传与分布式存储。

2）组件 系统微服务基于SpringBoot开发，[**数据库连接池**](https://baike.baidu.com/item/%E6%95%B0%E6%8D%AE%E5%BA%93%E8%BF%9E%E6%8E%A5%E6%B1%A0?fromModule=lemma_search-box)采用Druid，POJO构建采用Lombok，日志系统采用Log4j2， Guava工具类库，Mybatis Plus持久层接口实现，Sharding-jdbc分库分表组件，Swagger接口规范组件，XXL-job 分布式任务调度组件，Sentinel限流组件等。

3）接入 Zuul网关完成客户端认证、路由转发等功能，Dubbo RPC完成微服务远程调用，JWT提供前后端令牌管理方案。

4）视图 平台支持H5、PC等各种前端.

# 4.参照

<https://mp.weixin.qq.com/s?__biz=MzU0OTE4MzYzMw==&mid=2247487084&idx=7&sn=3f8b95a2832393cb4d75746c1a57751c&chksm=fbb28792ccc50e84ed2d3eab8537f563ed347aa9201a0d1987bf6569f3be19953b90aece8821&scene=27>

<https://baike.baidu.com/item/%E5%86%85%E5%AE%B9%E5%88%86%E5%8F%91%E7%BD%91%E7%BB%9C/4034265?fromtitle=CDN&fromid=420951&fr=aladdin>

https://baike.baidu.com/item/%E6%95%B0%E6%8D%AE%E5%BA%93%E8%BF%9E%E6%8E%A5%E6%B1%A0?fromModule=lemma\_search-box