



欢迎大家加入艾编程

今天是开班典礼&第一次课，需要和大家同步以下信息

- 学习周期为4个月
- 上课时间为每周三、周六、周日 20:00-23:00
- 课程答疑时间为每周一、周二、周四、周五13:00-22:00，可通过微信或钉钉联系讲师
- 每堂课我们都会有作业布置给大家，请大家按照作业要求进行完成
- 作业会在钉钉群里发给大家，完成后将作业发送到艾编程邮箱：icodingedu@126.com

艾编程的使命

- 为每个互联网人提供高质量的终身学习平台

课程介绍

项目演进过程

历经15次架构演进过程实操



项目实施阶段

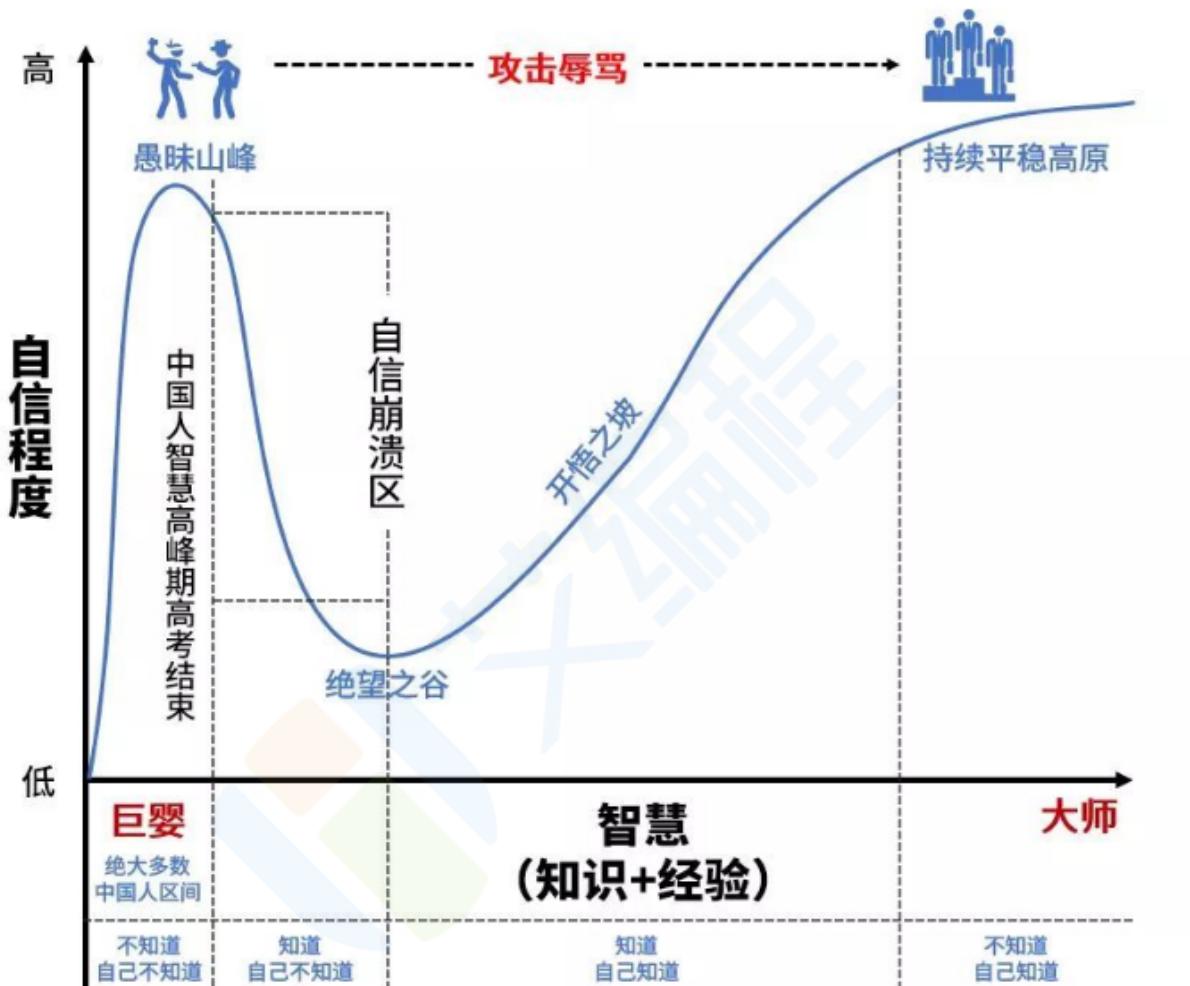


项目技术覆盖

Springboot、MyBatis、Redis、Nginx、ES搜索引擎、FastDFS、分布式会话、单点登录、RabbitMQ消息队列、MyCat、Sharding-Jdbc、分布式锁设计、分布式全局id设计、分布式事务和数据一致性、分布式限流、Springcloud G版微服务、Dubbo、Jenkins、Docker、k8s、容器弹性扩容、JVM性能调优、MySQL性能优化、Tomcat性能调优、Linux性能优化等，以上技术栈覆盖了市面上90%的技术内容。

我们开始今天的课程

首先我们来看一张图



上图是著名的邓宁-克鲁格效应

邓宁-克鲁格效应是指的是能力欠缺的人在自己欠考虑的决定的基础上得出错误结论，但是无法正确认识到自身的不足，辨别错误行为，是一种认知偏差现象。这些能力欠缺者们沉浸在自我营造的虚幻的优势之中，常常高估自己的能力水平，却无法客观评价他人的能力。

邓宁-克鲁格效应有几种现象：

- 能力越差的人越容易高估自己。
- 能力越差的人发现不了自己和别人的差距。
- 人类有虚幻的优越感，总觉得自己比别人棒

邓宁和克鲁格做了很多研究之后指出：

在很多复杂工作上，越是能力一般的普通人越觉得自己和高手差不多。

在很多复杂情绪上，越是情绪控制能力差的人越觉得自己做得特别好。

所以有些考试后，成绩好的学生有时候会感觉考砸了；而成绩一般般的竟然觉得不错。

有些行业优秀者轻松而完美地完成一些高难度的技术操作后，往往会认为这些事情是很容易的事，有时候也理所当然地以为别人也必须这样完成才是正常的，完成不了就是别人很笨、很差劲。

有时候某人完成一件事，明明做得很好，不容易了。有些人却偏偏要鸡蛋里挑骨头，说还有某方面做得不好，不这样就不显示出自己的能耐。其实鸡蛋里挑骨头的人往往会觉得比别人差很多。

什么是系统架构？

我们先看看Baidu百科给出的一个解释

系统架构师

 编辑

 讨论

同义词 架构师一般指系统架构师

本词条由“科普中国”科学百科词条编写与应用工作项目 审核。

系统架构师是一个最终确认和评估系统需求，给出开发规范，搭建系统实现的核心构架，并澄清技术细节、扫清主要难点的技术人员。主要着眼于系统的“技术实现”。因此他/她应该是特定的开发平台、语言、工具的大师，对常见应用场景能给出最恰当的解决方案，同时要对所属的开发团队有足够的了解，能够评估自己的团队实现特定的功能需求需要的代价。系统架构师负责设计系统整体架构，从需求到设计的每个细节都要考虑到，把握整个项目，使设计的项目尽量效率高，开发容易，维护方便，升级简单等。

系统架构的使命和责任

● 兼容过去问题：历史数据和业务的兼容，老系统过度

- 1、空中加油：难度最大、历史数据兼容复杂、业务分支多、发布过程中的灰度要非常短，切换速度再快也要防止脏数据的产生
- 2、新旧分割：老用户用老系统，新用户用新系统，数据互相隔离，待新系统完全稳定后可以对老系统用户做后端迁移，常用于APP版本共存阶段
- 3、休克疗法：一刀切，制定好停机时间，老系统关闭，历史数据迁移，新系统开启使用，升级失败会处于两难的境地，多用于垄断性APP或强大的甲方

● 解决当前问题：新需求的和业务功能的扩展

- 1、投入产出比高：需求估时中，对系统流程和架构的了解时间比例要低于20%
- 2、日常维护成本低：系统功能性问题不影响业务需求的发展
- 3、可扩展性强：避免出现增加一个小功能就进行大动干戈的整体回归测试
- 4、系统稳定、故障率低：SLA至少在四个9，服务停机时间不能超过5分钟以上

● 适度解决未来问题：don't over design

- 1、充分分析：时间成本、人力成本、机会成本
- 2、未来2-3年用户规模的扩大程度
- 3、未来2-3年现有用户的自我成长和进化方向
- 4、竞争对手商业竞争的残酷性

- 5、未来2-3年技术栈的更新迭代
- 6、研发团队的自身因素（团队稳定、结构、成本）

明确架构的使命和责任后接下来：Just do it

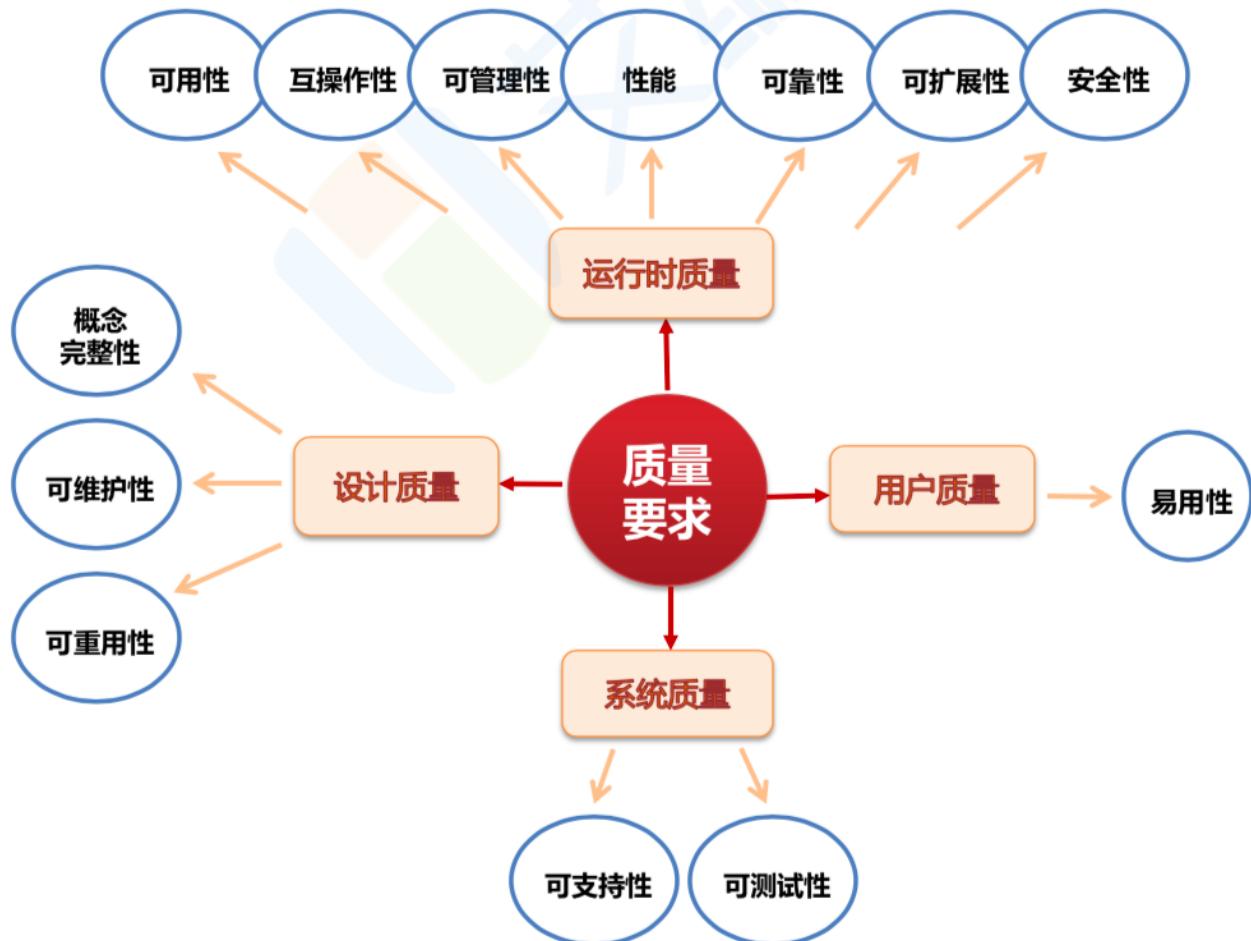
制定架构设计的目标

系统架构不是无目的性的设计和规划，一定是基于业务本身去做设计和分析，系统架构也不是一蹴而就的，是随着业务的发展逐步优化和生长起来的，但在前期设计的时候就需要给后期做好预留工作，这就需要对系统架构的相关设计理论和原则进行掌握和学习，第一步先明确架构设计的目标

- 1、高可用性：系统整体可用性SLA至少99.99%，整个系统全年不超过50分钟，单个系统故障不超过5分钟；
SLA 99.99% 的计算方式 $99.99\% = 8760 * 0.0001 = 0.876$ 小时 $= 0.876 * 60 = 52.6$ 分钟
- 2、高可扩展性：架构简单清晰，应用系统间耦合低，容易水平扩展；
- 3、低成本：提高服务重用性，降低人力成本，减少服务器成本；
- 4、多快好省：构建平台兼顾效率和性能，以高时效低成本为目标。

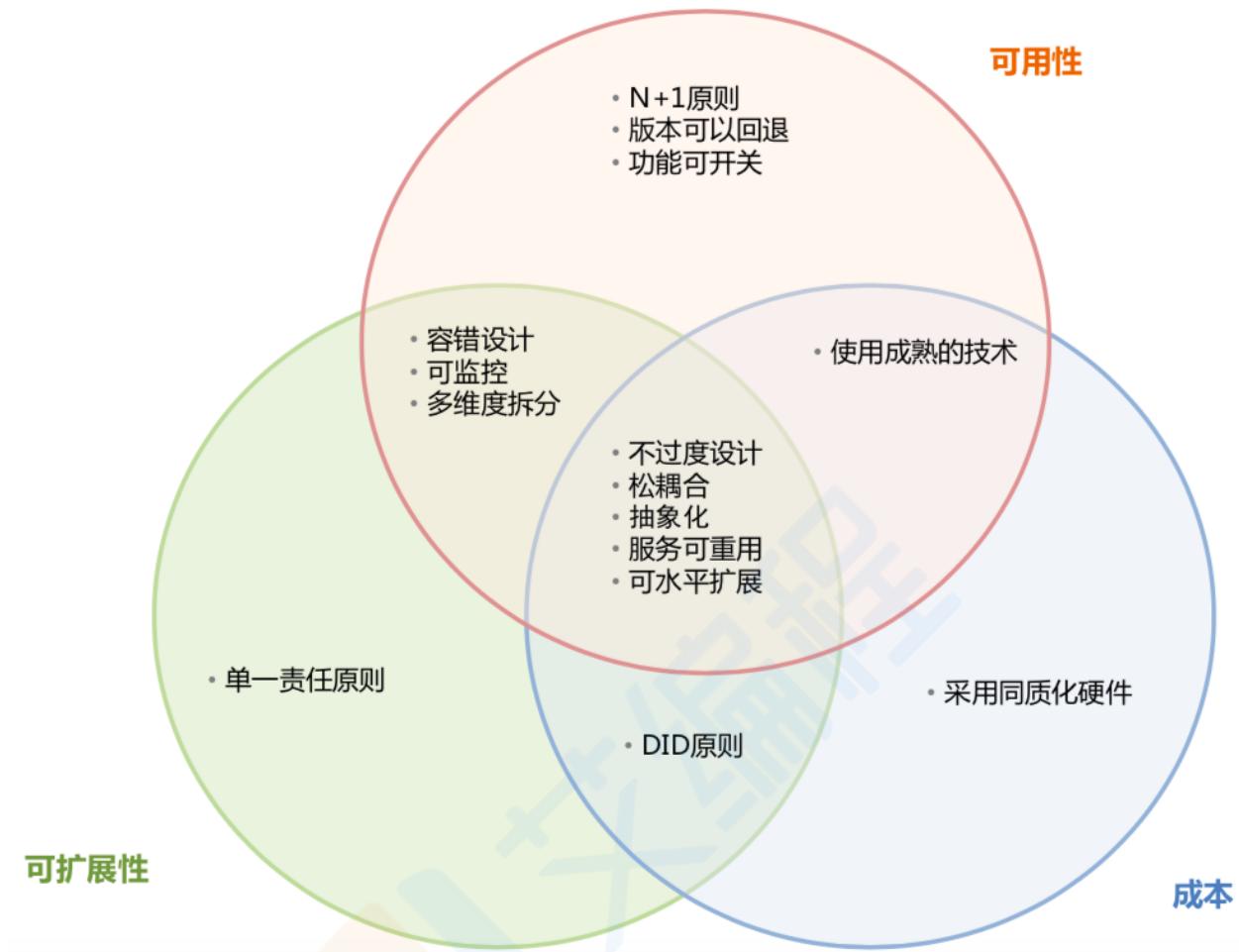
制定架构质量要求

一个好的架构要考虑各种影响因素，架构的质量要求是衡量架构的一个立体因素，贯穿了系统架构的各个方面，同样也从结果上为我们



掌握系统架构设计的方法论和原则

系统架构过程中有很多实践得出的设计原则帮助我们在架构中提前规避问题，协助我们更好的实现系统架构目标，对于一个初涉系统架构设计的研发人员具有很好的指导意义，虽然原则的内容有些枯燥，但明白了原则所阐述的标准和法则后就能在设计架构的时候尽量少走弯路（虽然大家在架构设计的过程中所需要走的弯路一步都不能少，但我至少希望通过我的讲解大家能知道自己正在走弯路）



1、单一责任原则；

这个大家在设计模式里应该都接触过，一个接口、方法、类，都应该只承担一个任务或责任，而不是完成多个任务功能，这样复杂度降低后便于功能维护、风险控制、变更管理等，而对于一个系统功能模块或服务也是一样，如果功能复杂，系统耦合度很高就不便于后续进行分布式扩展，功能模块互相影响也会导致业务节点崩溃（eg：业务系统和数据库放在同一台设备上就是严重违背单一责任的架构方式，这也就是俗称单一应用架构）

2、DID原则；

Design(D)设计20倍的容量；Implement(I)实施3倍的容量；Deploy (D) 部署1.5倍的容量
(eg：在未来2-3年时间你设计的系统架构在不进行系统化重构的情况下可以通过设备或节点扩展正常支撑业务发展）

3、N+1原则；

任何业务服务节点永远不要少于两个可独立运行设备（eg：集群服务的前提条件）

4、版本可回退；

一单线上版本发布后出现问题，可以回退至上一次正常的业务状态

5、功能可开关；

系统的一些功能入口可以进行关闭并做出友好提示 (eg: X宝双11当天，无法查询个人历史订单和商品物流信息等)

6、系统容错；

一个系统总是或多或少出现一些未知的系统问题，但对于用户则不要将类似404、500、502等错误展示在前端 (eg: 通过技术手段将错误展示页面转至提前准备的友好提示页面，并能从该页面继续进行业务操作)

7、服务可重用；

这是系统服务化的前提

8、服务可水平扩展；

水平扩展实现服务集群化以及水平服务能力随时增加，系统要水平扩展非垂直升级。永远不要依赖更大、更快的服务器系统。

9、松耦合、抽象化；

开发层面的松耦合大家都很熟悉了，系统架构层面最典型的应用就是模块之间的服务调用，而不是将模块功能内嵌，现在的微服务架构就很好的实现了这个原则

10、不过度设计；

杜绝浪费是在任何时候都要考虑到的，开源节流中的节流是系统架构设计中需要着重考虑的，虽然前面有DID原则但一个系统的技术架构也不要为未来5年甚至是10年来买单。

11、使用成熟的技术；

使用成熟技术好处有三点 1、不是吃螃蟹的第一人，没有太多技术坑，有问题也能很快的找到解决方案 2、学习资料充足，应用成本低 3、市场上技术人员保有量高，人好招

12、采用同质化硬件或云服务。

现在企业很少自建IDC机房或进行服务器租用，基本都上云了，整个系统的云服务采用同一家服务商，可以避免网络消耗，出现问题也好统一服务解决，避免跨服务商出现不可知的问题

架构设计组成的关键层级

系统架构不仅仅是技术层面的设计，首先要从业务角度出发，先完成业务层级的架构，根据业务层级的需要对技术和数据层进行设计，我们以电商业务为切入点，带领大家按照业务层级关系进行对应的架构设计。

首先我们来看一下各架构层级间关系



业务架构的设计方法与实现

业务架构的设计原则

1. 业务平台化

- 业务平台化，相互独立。如交易平台、仓储平台、物流平台、支付平台、广告平台等
- 基础业务下沉，可复用。如用户、商品、类目、促销、时效等



2. 核心业务、非核心业务分离

- 电商核心业务与非核心业务分离，核心业务精简（利于稳定），非核心业务多样化。如，主交易服务、通用交易服务

4. 区分主流程、辅流程

- 分清哪些是电商的主流程。运行时，优先保证主流程的顺利完成，辅流程可以采用后台异步的方式。避免辅流程的失败导致主流程的回滚。如，下单时，同步调用快照，异步通知台账、发票

3. 隔离不同类型的业务

- 交易业务是签订买家和卖家之间的交易合同，需要优先保证高可用性，让用户能快速下单
- 履约业务对可用性没有太高要求，可以优先保证一致性
- 闪购业务对高并发要求很高，应该跟普通业务隔离

业务架构的设计实例



应用架构的设计方法与实现

应用架构设计原则



应用架构分析设计过程

- 应用分层



- 应用架构的分解设计原则



- 应用架构的依赖设计原则

2. 跨域弱依赖

- 跨业务域调用时，尽可能异步
弱依赖

1. 依赖稳定部分

- 稳定部分不依赖易变部分
- 易变部分可以依赖稳定部分
- 要求：避免循环依赖



3. 基本服务依赖

- 基本服务不能向上依赖流程服务
- 组合服务、流程服务可以向下依赖
基本服务
- 条件：基本服务稳定

6. 核心服务依赖

- 核心服务不依赖非核心服务
- 非核心服务可依赖核心服务
- 条件：核心服务稳定

5. 平台服务依赖

- 平台服务不依赖上层应用
- 上层应用可依赖平台服务
- 条件：平台服务稳定

4. 非功能性服务依赖

- 非功能性服务不依赖功能性服务
- 功能性服务可依赖非功能性服务
- 条件：非功能性服务稳定

• 应用架构的服务设计原则

无状态

- 尽量不要把状态数据
保存在本机
- 接口调用幂等性

可复用

- 复用粒度是有业务逻
辑的抽象服务，不是
服务实现细节
- 服务引用只依赖于服
务抽象

松耦合

- 跨业务域调用，尽可
能异步解耦
- 必须同步调用时，设
置超时和队列大小
- 相对稳定的基本服务
与易变流程服务分层

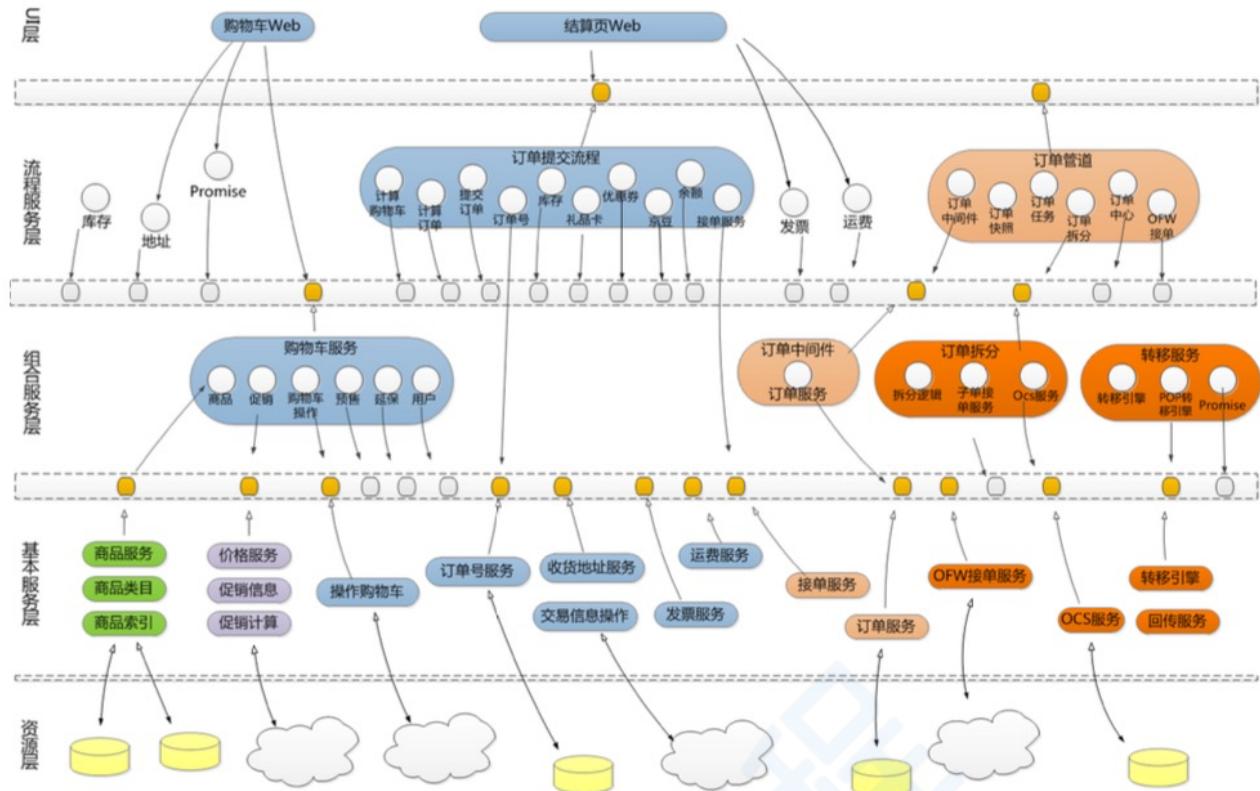
可治理

- 制订服务契约
- 服务可降级
- 服务可限流
- 服务可开关
- 服务可监控
- 白名单机制

基本服务

- 基础服务下沉、可复用。如时效、库存、价格计算等
- 基础服务自治，相互独立
- 基础服务的实现，要求精简、可水平扩展
- 基础服务实现物理隔离，包括基础服务相关的数据

应用架构设计实例：基于交易订单部分



数据架构的设计方法与实现

数据架构设计原则

2 数据、应用分离

- 应用系统只依赖逻辑数据库
- 应用系统不直接访问其它宿主的数据库，只能通过服务访问

1 统一数据视图

- 保证数据的及时性、一致性、准确性、完整性

3 数据异构

- 源数据和目标数据内容相同时，做索引异构。如商品库不同维度
- 内容不同时，做数据库异构。如订单买家库和卖家库。

数据架构

6 合理使用缓存

- 数据库有能力支撑时，尽量不要引入缓存
- 合理利用缓存做容灾

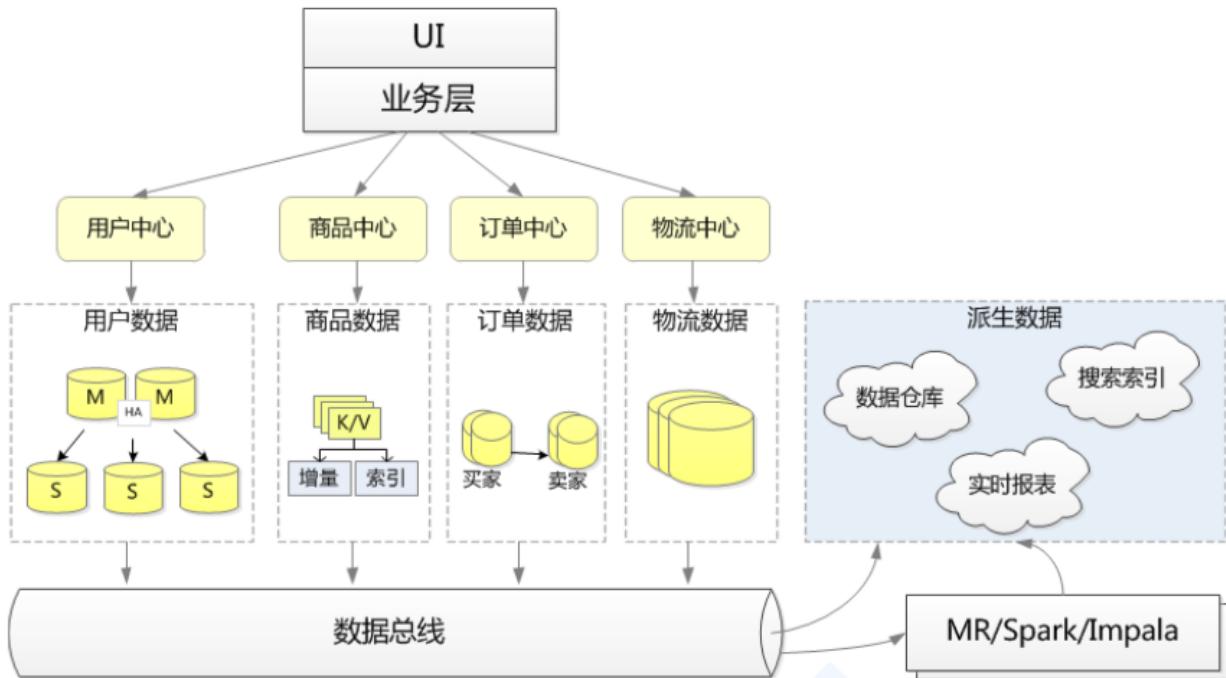
4 数据读写分离

- 访问量大的数据库做读写分离
- 数据量大的数据库做分库分表
- 不同业务域数据库做分区隔离
- 重要数据配置备库；

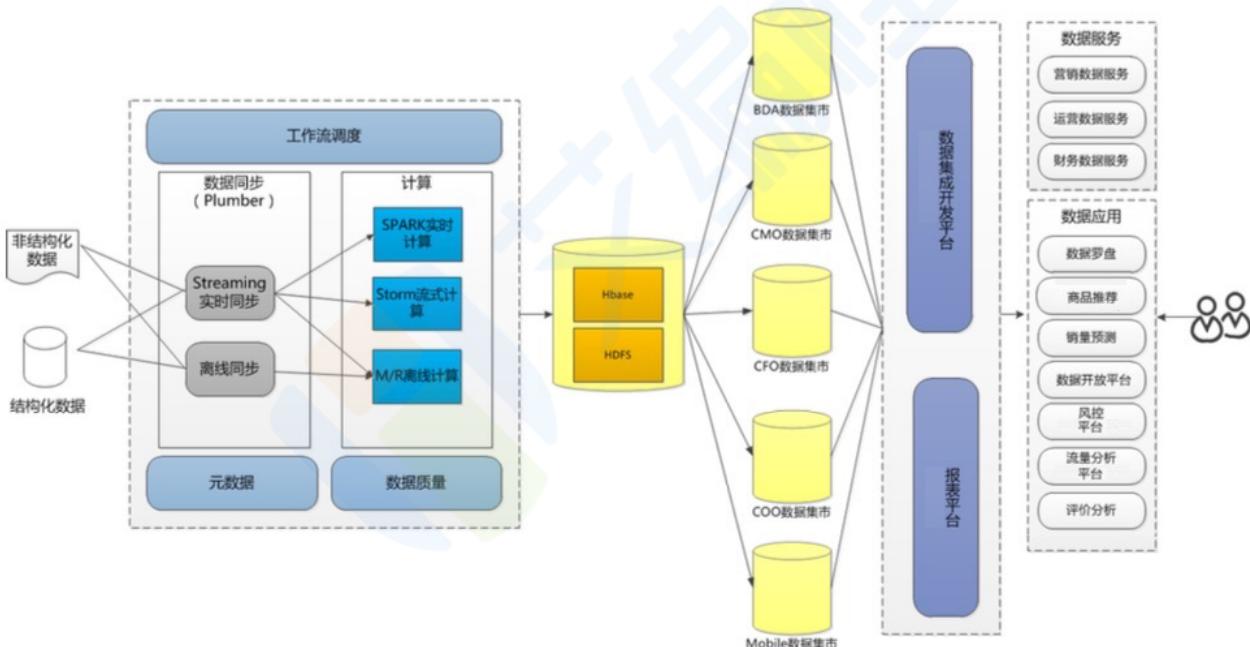
5 用Mysql数据库

- 除成本因素外，Mysql的数据库扩展性和支持高并发的能力较强，公司研发和运维在这方面积累了大量经验

典型的数据架构应用



典型的大数据平台架构应用



技术架构设计原则

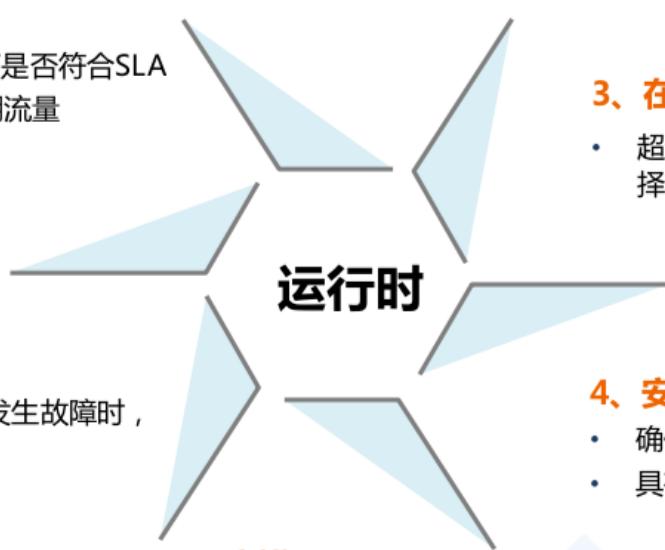
系统运行时原则

2、应用可回滚，功能可降级

- 应用出现问题时，要求能回滚到上一版本，或做功能开关或降级

1、可监控

- 服务的TPS和RT是否符合SLA
- 是否出现超预期流量



6、可故障转移

- 多机房部署，发生故障时，能即时切换

3、在线扩容

- 超预期流量时，应用系统可选择在线水平扩展

4、安全保证

- 确保系统的保密性和完整性
- 具有足够的防攻击能力

5、可容错

- 核心应用要求多活，避免单点设计，并且自身有容错和修复能力。故障时间TTR小

系统部署原则

2 D-I-D原则

- 设计20倍的容量 (Design)
- 实现 3 倍的容量 (Implement)
- 部署1.5倍的容量 (Deploy)

1 N+1原则

- 确保为故障多搭建一套系统，避免单点问题。例如，多机房部署、应用系统集群、数据库主备等
- 功能开发与运维分开。系统开发完成后，交给专业的运维团队管理和运营。

3 支持灰度发布

- 系统新上线，要求支持“灰度”发布，分步切流量，故障回滚

5 业务子网

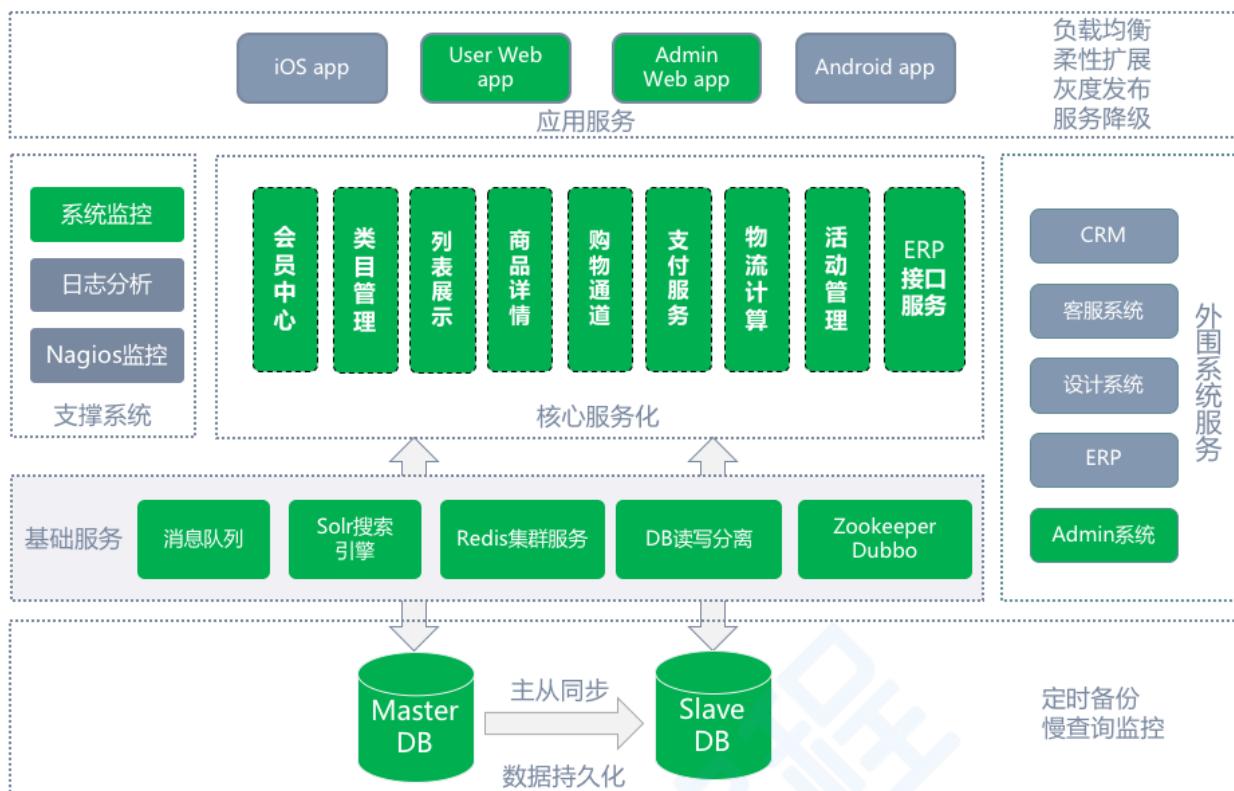
- 机房部署以业务域划分：基本服务和数据库，相同业务域的服务器部署在一起；不同业务域的服务器物理隔离

4 虚拟化部署

- 虚机部署：二级系统、三级系统采用虚拟机部署，节省资源和管理成本
- 虚拟化部署：一级系统应用服务器，采用虚拟化部署

根据业务、应用、数据架构进行整体技术架构输出

电商系统应用技术架构

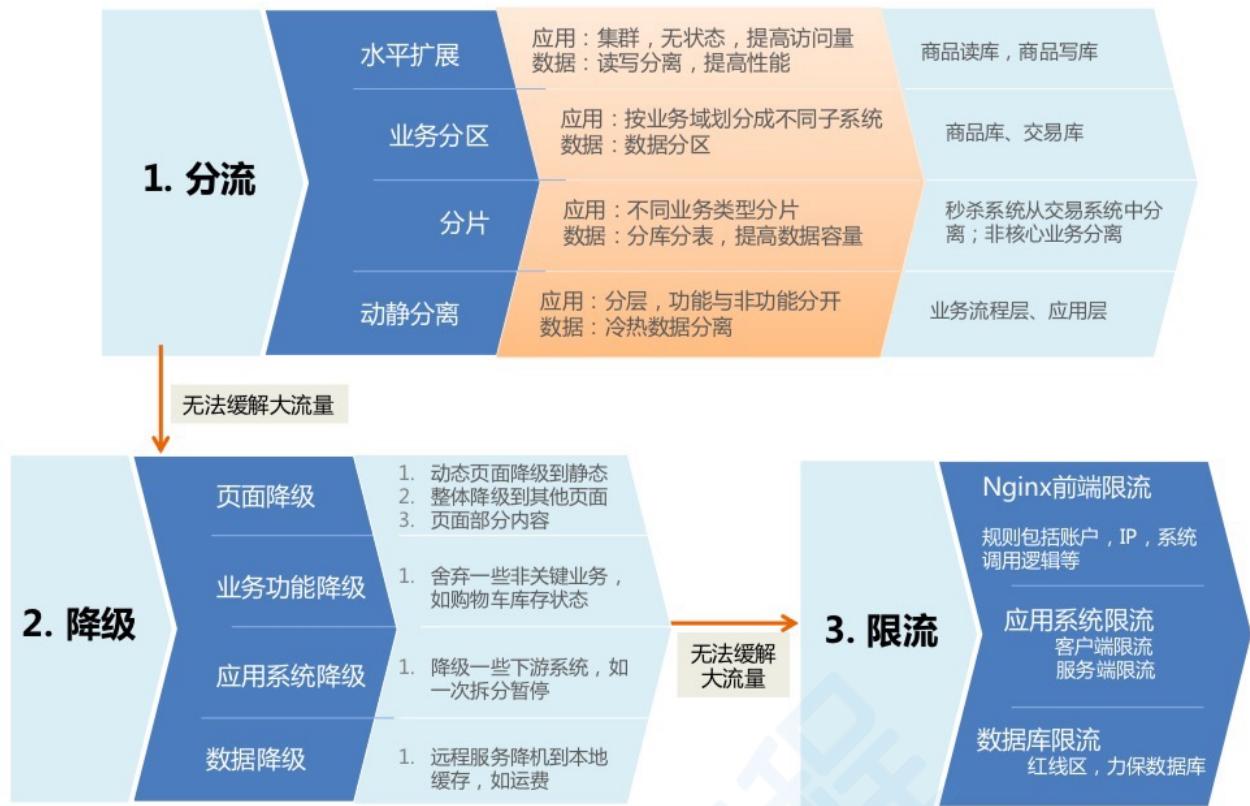


日均亿级流量应对方案

系统准备阶段



大流量高并发过程中的流量应对措施



架构设计总结

	解耦/拆分	抽象	集成	复用	治理
业务	1. 电商业务域 2. 核心、非核心业务 3. 主流程、辅流程 4. 业务规则分离		1. 跨业务域调用异步 2. 非核心业务异步	1. 基础业务下沉，可复用	1. 厘清业务边界、作用域
应用	1. 应用集群水平扩展 2. 按业务域分离应用 3. 按功能分离应用 4. 按稳定性分离应用	1. 服务抽象，服务调用不依赖实现细节 2. 应用集群抽象，应用位置透明	1. 易变依赖稳定 2. 流程服务依赖基础服务 3. 非核心应用依赖核心应用	1. 复用粒度是有业务逻辑的抽象服务	1. 服务自治 2. SLA 3. 可水平扩展 4. 可限流 5. 服务可降级 6. 容错设计 7. 服务白名单
数据	1. 读写分离 2. 按业务域分库 3. 分库分表 4. 冷热数据分离	1. 数据库抽象。应用只依赖逻辑数据库	1. 数据库只能通过服务访问 2. 统一的元数据管理 3. 统一的主数据管理		1. 重要数据做主备 2. 合理利用缓存容灾 3. 双写要做补偿
技术	1. 功能开发与运维分离 2. 业务子网 3. 分离功能、非功能性需求	1. 服务器资源抽象。应用只依赖虚拟化资源	1. 同步调用时，设置超时和任务队列长度 2. 利用回调异步化 3. 利用MQ、缓存、中间件异步化	1. 代码提共通，可复用 2. 非功能性服务，可复用 3. 基础配置、基础软件复用	1. N+1设计 2. 灰度部署 3. 版本可回滚 4. 可监控 5. 可容灾

本文档仅供艾编程架构师VIP学员内部学习，严禁传播或其他商业用途，同学们切记！！！

