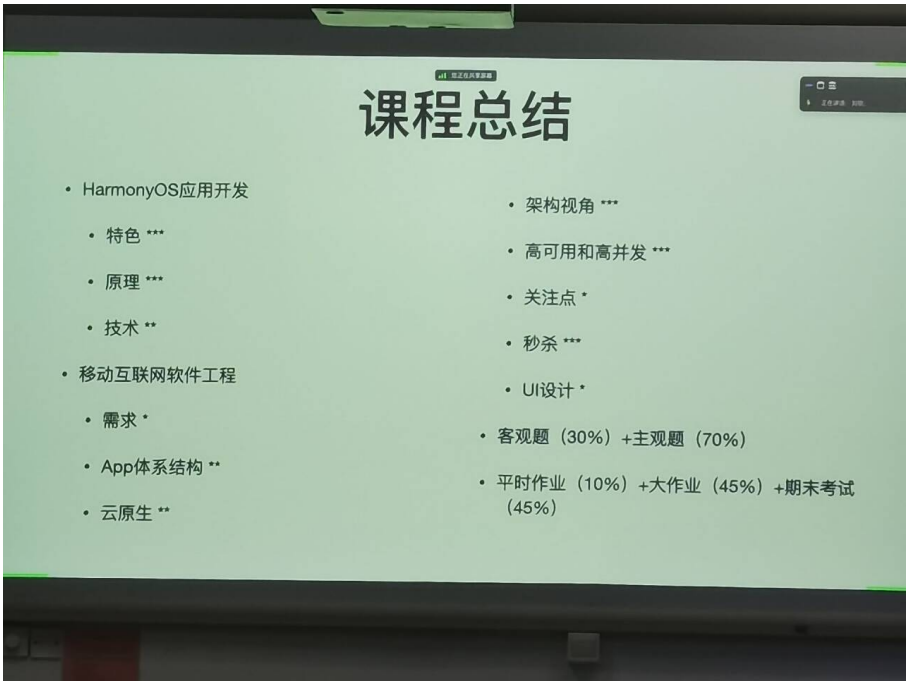


一、架构视角

1. 架构设计愿景

1.1 架构目标

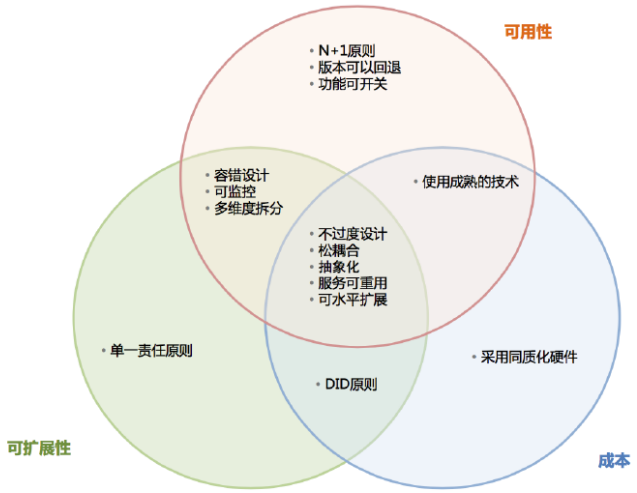
- 高可用性
- 高可扩展性
- 低成本
- 多快好省

1.2 质量要求

- 设计质量：概念完整性、可维护性、可重用性
- 运行时质量：可用性、互操作性、可管理性、性能、可靠性、可扩展性、安全性
- 系统质量：可支持性、可测试性
- 用户质量：易用性

1.3 总体架构原则

可用性、可扩展性、成本：



2. 架构设计

组成和关键：

- 组成：业务架构、应用架构、数据架构、技术架构
- 关键：解耦、拆分、抽象、集成、复用、治理
- 整体横向分层抽象，局部纵向贯穿分解

2.1 业务架构

2.1.1 设计原则

- 业务平台化
- 核心业务、非核心业务分离

- 区分主流程、辅流程
- 隔离不同类型业务

2.1.2 实例：基础业务下沉

2.2 应用架构

2.2.1 设计原则

- 稳定性原则
- 解耦/拆分
- 抽象化
- 松耦合
- 容错设计

2.2.2 架构分解原则

- 水平扩展（复制）：多级集群，提高并发能力
- 垂直拆分（不同业务拆分）：按业务域划分系统
- 业务分片（同业务分片）：按功能特点分开部署，如秒杀
- 水平拆分（稳定与易变分离）：服务分层，功能和非功能分开（冷热数据、历史数据分离）

2.2.3 依赖原则

- 依赖稳定部分
- 跨域弱依赖
- 基本服务依赖
- 非功能性服务依赖
- 平台服务依赖
- 核心服务依赖

2.2.4 **服务设计原则**

- 无状态
- 可复用
- 松耦合
- 可治理
- 基本服务：
    - 基础服务下沉、可复用
    - 基础服务自治、相互独立
    - 精简、可水平扩展
    - 物理隔离，包括其相关数据

2.2.5 **为何要服务化**

- 系统规模随着业务发展而增长，原有系统架构模式逻辑过于耦合不再适应
- 拆分后的子系统逻辑内聚，利于局部扩展
- 子系统间通过接口交互，可独立变化

> RPC（远程方法调用）：
>
> - 管道、信号、信号量、消息队列、共享内存、本地套接字
> - 基本问题：如何表示数据、如何传递数据、如何表示方法
>
> REST（表征状态转移）：
>
> - 资源、表征、状态、转移

2.3 数据架构

2.3.1 设计原则

- 统一数据视图
- 数据应用分离
- 数据异构
- 数据读写分离
- 用MySql数据库
- 合理使用缓存

2.4 技术架构

2.4.1 系统运行时原则

- 可监控
- 应用可回滚，功能可降级
- 在线扩容
- 安全保证
- 可容错
- 可故障转移

2.4.2 系统部署原则

- N+1原则：多搭建一套

- DID原则：设计20倍，实现3倍，部署1.5倍
- 支持灰度发布
- 虚拟化部署
- 业务子网

> JSF：HA&负载均衡、性能优化、配置、限流、降级、弹性云部署等

3. 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、非功能型需求	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

二、高可用和高并发

1. 高可用

1.1 负载均衡与反向代理

- 负载均衡：轮询、带权重的轮询、ip_hash同ip同服务器

- 反向代理：location用于匹配请求的URL

1.2 隔离

- 线程隔离、进程隔离、集群隔离、机房隔离
- 读写隔离、动静隔离、爬虫隔离、热点隔离
- 环境隔离、压测隔离、缓存隔离、查询隔离

1.3 限流

- 算法：令牌桶、漏桶、计数器
- 应用级：限流总请求/连接/并发数、限流总资源数、限流接口的请求数或时间窗口请求数、平滑限流接口请求数
- 分布式：限流服务原子化（Redis+Lua计数器或Nginx+Lua令牌环）
- 接入层：请求流量的入口，Nginx自带
- 节流：Throttle

1.4 降级

- 按日志级别、按是否自动化、按功能、按系统中处于的层次
- 自动开关降级：
    - 超时降级
    - 统计失败次数降级
    - 故障降级
    - 限流降级
- 读服务降级：降级到读缓存、降级到静态化

  写服务降级：Redis和DB的处理

1.5 超时与重试

- 代理层超时：Nginx客户端超时、DNS解析超时、代理超时
- Web容器超时：Tomcat，Jetty，

- 数据库客户端超时：MySQL、Oracle
- NoSQL客户端超时：MongoDB，Redis
- 业务超时：任务型、服务调用型
- 前端Ajax超时

1.6 回滚

- 事务回滚
- 代码库回滚
- 部署版本回滚
- 数据版本回滚
- 静态资源回滚

1.7 压测与预案

- 系统压测：线下和线上（读写压测、混合压测、仿真压测、引流压测、隔离集群压测、显示集群压测、全链路压测等）
- 系统优化与容灾
- 应急预案：分机、全链路分析、配置监控报警、其他应急预案

2. 高并发

2.1 应用级缓存

- 缓存命中率、回收策略（基于空间、基于容量、基于时间、基于Java对象应用）、回收算法（FIFO、LRU、LFU）
- 缓存使用模式：Cache-Aside（直接维护Cache和SoR，适合AOP），Cache-As-SoR（业务只看到Cache看不到SoR）

2.2 HTTP缓存

- 浏览器请求直接取Last-Modified
- 分布式缓存：单机全量缓存+主从

2.3 连接池

- 数据库、HTTPClient、线程池

2.4 异步并发

- Future、Callback

2.5 扩容

- 单体：水平口容、垂直扩容
- 应用拆分、数据库拆分
- 分库分表

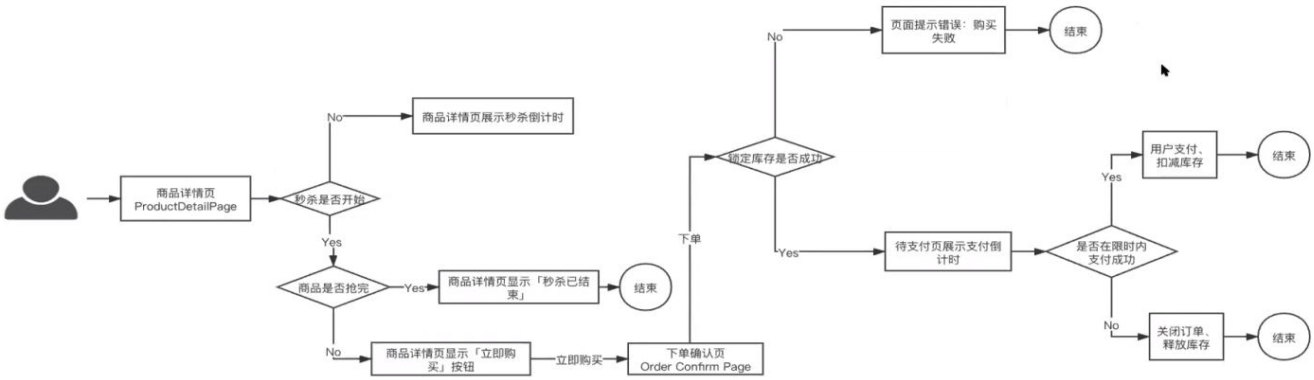
2.6 队列

- 缓冲队列、任务队列、消息队列
- 请求队列、数据总线队列、混合队列

三、秒杀

1. 业务需求

- 登录注册、商品管理、商品展示、秒杀



2. 架构原则（4要1不要）

- 数据要尽量少
- 请求数要尽量少
- 路径要尽量短
- 依赖要尽量少
- 不要有单点

3. 技术方案

3.1 动静分离

- 静态数据缓存到离用户近的地方（浏览器、CDN、服务端）
- 动态数据分离：URL唯一化、分离浏览者相关因素、分离时间因素、异步化低于因素、去Cookie
- 动态内容处理：ESI（Web代理服务器做动态内容请求并插入静态页面）、CSI（单独发起异步JS请求）

3.2 二八原则：热点数据

- 静态热点数据：能提前预测
- 动态热点数据：监控发现突发热点
- 处理思路：优化、限制、隔离

3.3 流量削峰

- 排队、答题、分层过滤
- 分层校验原则：
    - 动态请求的数据缓存在Web端，过滤无效数据读
    - 不做强一致性校验
    - 写数据基于时间合理分片，过滤国企请求
    - 对写做限流保护
    - 对写做强一致性校验

- 优化：减少编码、减少序列化、Java极致优化（直接用Servlet）、并发读数据

3.4 减库存

- 下单减库存（恶意）、付款减库存（多卖）、预扣库存
- 并发锁：应用层排队

3.5 高可用

3.6 其他业务需求实现

定时发放随机数从而实现卡时间

4. 方案对比

Java：Java+Nginx+Tomcat+Redis+ShardingSphere

一致性、单Redis性能（缓存击穿、缓存雪崩）

Go：Golang+RabbitMQ+Iris+MySQL