1. 高并发原则

无状态

- 无状态应用，便于水平扩展
- 有状态配置可通过配置中心实现无状态
- 实践: Disconf、Yaconf、Zookpeer、Consul、Confd、Diamond、Xdiamond等

拆分

- 系统维度：按照系统功能、业务拆分，如购物车，结算，订单等
- 功能维度：对系统功能在做细粒度拆分
- 读写维度：根据读写比例特征拆分；读多，可考虑多级缓存；写多，可考虑分库分表
- AOP维度： 根据访问特征，按照AOP进行拆分，比如商品页可分为CDN、页面渲染系统，CDN就是一个AOP系统
- 模块维度：对整体代码结构划分Web、Service、DAO

服务化

- 服务化演进: 进程内服务-单机远程服务-集群手动注册服务-自动注册和发现服务-服务的分组、隔离、路由-服务治理
- 考虑服务分组、隔离、限流、黑白名单、超时、重试机制、路由、故障补偿等
- 实践：利用Nginx、HaProxy、LVS等实现负载均衡，ZooKeeper、Consul等实现自动注册和发现服

消息队列

- 目的: 服务解耦(一对多消费)、异步处理、流量削峰缓冲等
- 大流量缓冲： 牺牲强一致性，保证最终一致性(案例：库存扣减，现在Redis中做扣减，记录扣减日志，通过后台进程将扣减日志应用到DB)
- 数据校对: 解决异步消息机制下消息丢失问题

数据异构

- 数据异构: 通过消息队列机制接收数据变更，原子化存储
- 数据闭环: 屏蔽多从数据来源，将数据异构存储，形成闭环

缓存银弹

- 用户层:
  - DNS缓存
  - 浏览器DNS缓存

    - 操作系统DNS缓存
    - 本地DNS服务商缓存
    - DNS服务器缓存
    - 客户端缓存
    - 浏览器缓存(Expires、Cache-Control、Last-Modified、Etag)* App客户缓存(js/css/image...)
- 代理层：
    - CDN缓存(一般基于ATS、Varnish、Nginx、Squid等构建,边缘节点-二级节点-中心节点-源站)
- 接入层：
    - Nginx为例：
    - Proxy_cache： 代理缓存,可以存储到/dev/shm或者SSD
    - FastCGI Cache
    - Nginx+Lua+Redis: 业务数据缓存
    - PHP为例：
        - Opcache： 缓存PHP的Opcodes
- 应用层：
    - 页面静态化
    - 业务数据缓存(Redis/Memcached/本地文件等)
    - 消息队列
- 数据层：
    - NoSQL： Redis、Memcache、SSDB等
    - MySQL： Innodb/MyISAM等Query Cache、Key Cache、Innodb Buffer Size等
- 系统层：
    - CPU : L1/L2/L3 Cache/NUMA
    - 内存
    - 磁盘：磁盘本身缓存、dirty_ratio/dirty_background_ratio、阵列卡本身缓存

并发化

2. 高可用原则

降级

- 降级开关集中化管理：将开关配置信息推送到各个应用
- 可降级的多级读服务：如服务调用降级为只读本地缓存
- 开关前置化：如Nginx+lua(OpenResty)配置降级策略，引流流量；可基于此做灰度策略
- 业务降级：高并发下，保证核心功能，次要功能可由同步改为异步策略或屏蔽功能

限流

- 目的: 防止恶意请求攻击或超出系统峰值
- 实践：
  - 恶意请求流量只访问到Cache
  - 穿透后端应用的流量使用Nginx的limit处理
  - 恶意IP使用Nginx Deny策略或者iptables拒绝

切流量

- 目的：屏蔽故障机器
- 实践:
  - DNS: 更改域名解析入口，如DNSPOD可以添加备用IP，正常IP故障时，会自主切换到备用地址;生效实践较慢
  - HttpDNS: 为了绕过运营商LocalDNS实现的精准流量调度
  - LVS/HaProxy/Nginx: 摘除故障节点

可回滚

- 发布版本失败时可随时快速回退到上一个稳定版本

3. 业务设计原则

- 防重设计
- 幂等设计
- 流程定义
- 状态与状态机
- 后台系统操作可反馈
- 后台系统审批化
- 文档注释
- 备份

4. 总结

先行规划和设计时有必要的，要对现有问题有方案，对未来有预案;欠下的

技术债，迟早都是要还的。

5. 分布式与集群的区别：

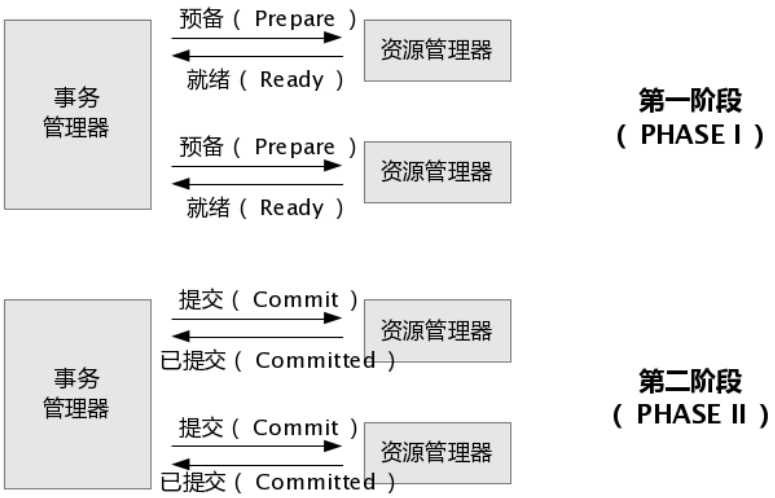
分布式是指将不同的业务分布在不同的地方。 而集群指的是将几台服务器集中在一起，实现同一业务

6. 分布式事务：

1. 二阶段提交：

a. 概念：参与者将操作成败通知协调者，再由协调者根据所有参与者的反馈情报决定各参与者是否要提交操作还是中止操作。

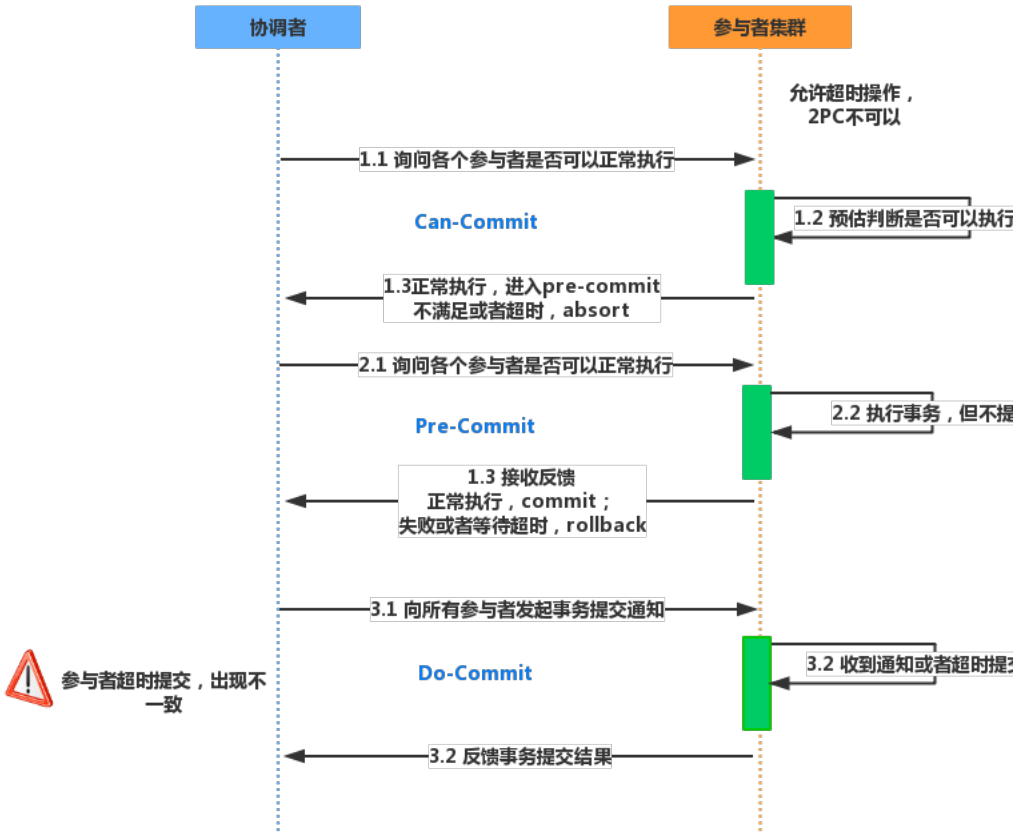
b. 作用：主要保证了分布式事务的原子性；第一阶段为准备阶段，第二阶段为提交阶段；



c. 缺点：不仅要锁住参与者的所有资源，而且要锁住协调者资源，开销大。一句话总结就是：2PC效率很低，对高并发很不友好。

2. 三阶段提交：

a. 概念：三阶段提交协议在协调者和参与者中都引入超时机制，并且把两阶段提交协议的第一个阶段拆分成了两步：询问，然后再锁资源，最后真正提交。这样三阶段提交就有CanCommit、PreCommit、DoCommit三个阶段。



b. 缺点：如果进入PreCommit后，Coordinator发出的是abort请求，假设只有一个Cohort收到并进行了abort操作，
而其他对于系统状态未知的Cohort会根据3PC选择继续Commit，此时系统状态发生不一致性。

3. 柔性事务：

a. 概念：所谓柔性事务是相对强制锁表的刚性事务而言。流程入下：服务器A的事务如果执行顺利，那么事务A就先行提交，如果事务B也执行顺利，则事务B也提交，整个事务就算完成。但是如果事务B执行失败，事务B本身回滚，这时事务A已经被提交，所以需要执行一个补偿操作，将已经提交的事务A执行的操作作反操作，恢复到未执行前事务A的状态。

b. 缺点：业务侵入性太强，还要补偿操作，缺乏普遍性，没法大规模推广。

4. 消息最终一致性解决方案之RabbitMQ实现：

a. 实现：发送方确认+消息持久化+消费者确认。

7. 什么时候用到分布式开发：

a. 优点：

i. 模块解耦：把模块拆分,使用接口通信,降低模块之间的耦合度.

ii. 项目拆分，不同团队负责不同的子项目：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.

iii. 提高项目扩展性：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。

iv. 分布式部署：可以灵活的进行分布式部署.

v. 提高代码的复用性：比如service层,如果不采用分布式rest服务方式架构就会在手机wap商城,微信商城,pc,android，ios每个端都要写一个service层逻辑,开发量大,难以维护一起升级,这时候就可以采用分布式rest服务方式,公用一个service层。

a. 缺点：

i. 系统之间的交互要使用远程通信,接口开发增大工作量；

ii. 网络请求有延时；

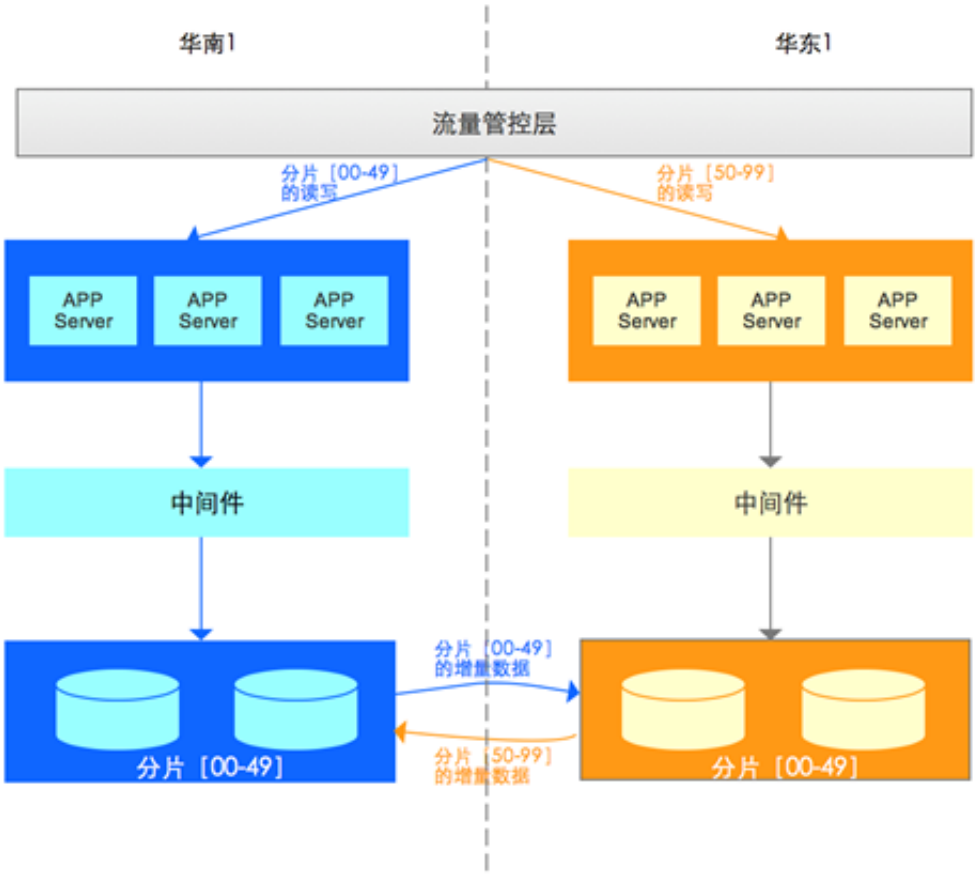
iii. 事务处理比较麻烦，需要使用分布式事务。

8. cdn（异地多活）

1、异地多活：异地多活指分布在异地的多个站点同时对外提供服务的业务场景。异地多活是高可用架构设计的一种，与传统的灾备设计的最主要区别在于“多活”，即所有站点都是同时在对外提供服务的。

2、两地容灾切换方案：

容灾是异地多活中最核心的一环， 以两个城市异地多活部署架构图为例：



- 在两个城市（城市1位于华南1地域、城市2位于华东1地域）均部署一套完整的业务系统。
- 下单业务按照“user_id”% 100 进行分片，在正常情况下：
  - [00~49]分片所有的读写都在城市1的数据库实例主库。
  - [50～99]分片所有的读写都在城市2的数据库实例主库。
- “城市1的数据库实例主库”和 “城市2的数据库实例主库”建立DTS双向复制。

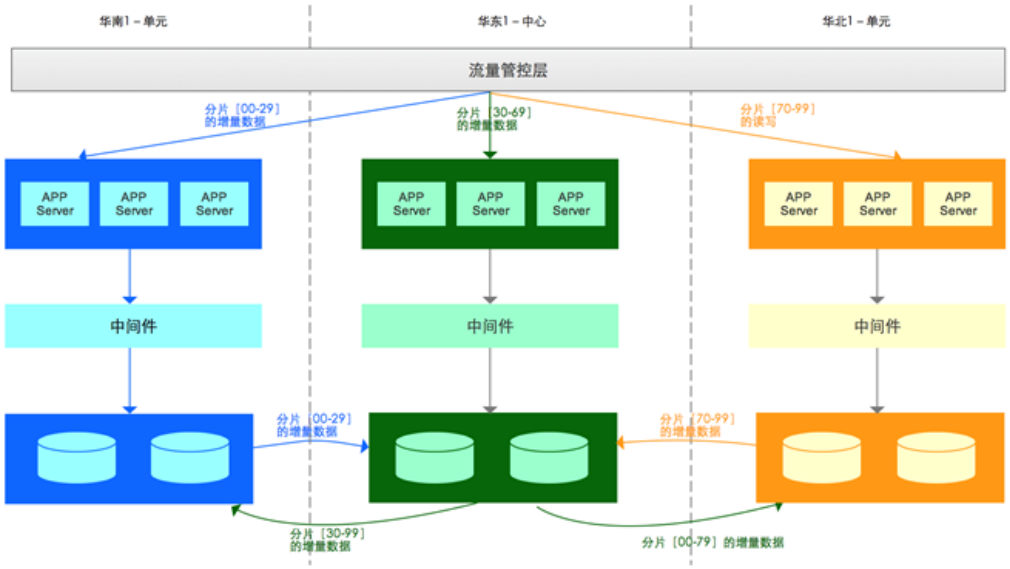
当出现异常时，需要进行容灾切换。可能出现的场景有以下4种：

序号	异常情况	操作
1	城市1数据库主库故障	1. 数据库引擎完成主备切换 2. DTS自动切换到城市1新主库读取新的增量更新，然后同步到城市2的数据库实例
2	城市1所有APP Server故障	有两种处理方案： • 方案1：数据库层无任何操作，APP Server切换到城市2，并跨城市读写城市1的数据库 • 方案2：APP Server和数据库都切换到城市2
3	城市1所有数据库故障	有两种处理方案： • 方案1：数据库层切换到城市2，APP Server跨城市读写城市2的数据库 • 方案2：APP Server和数据库都切换到城市2
4	城市1整体故障（包括所有APP Server ＋数据库等）	1. 城市1的全部数库流量切换到城市2 2. 城市1数据库到城市2数据库的DTS数据同步链路停止 3. 在城市2中，DTS启动，保存［00-49］分片的变更 4. 城市1故障恢复后，［00-49］的增量数据同步到城市1的数据库实例 5. 同步结束后，将［00-49］的数据库流量从城市2切回到城市1启动［00-49］分片从城市1到城市2的DTS同步

将第2种、第3种异常情况，全部采用第2种方案进行处理，那么不管是所有的APP Server异常、所有的数据库异常、整个城市异常，就直接按照城市级容灾方案处理，直接将APP Server、数据库切换到到另一个城市。

3、多城异地多活：

多城市异地多活模式指的是3个或者3个以上城市间部署异地多活。该模式下存在中心节点和单元节点：



- 中心节点：指单元节点的增量数据都需要实时的同步到中心节点，同时中心节点将所有分片的增量数据同步到其他单元节点。
- 单元节点：即对应分片读写的节点，该节点需要将该分片的增量同步到中心节点，并且接收来自于中心节点的其他分片的增量数据。

下图是3城市异地多活架构图，其中华东1就是中心节点，华南1和华北1是单元节点。

9. 分布式环境下宕机的处理方案?

1、dubbo：服务器宕机，zk临时被删除；

2、springcloud：每30s发送心跳检测重新进行租约，如果客户端不能多次更新租约，它将在90s内从服务器注册中心移除。

3、apm监控：

10.