一 设计原则

1. 高并发
2. 高可用

高并发的原则:

1. 拆分,对项目拆分,拆分可以划分不同的维度

> 系统维度: 按照系统功能/业务拆分 订单系统,商品系统,地址系统,优惠券.....
>
> 功能维度:优惠券拆分为 领券系统, 系统发券系统,用券
>
> 读写:: 读系统和写系统,读的量一定大于写的量,可以分库分表,数据聚合(数据比较大的情况下)
>
> 模块: controller,service,dao
>
> AOP: CDN,渲染页面

2. 拆分后保证应用无状态
3. 服务化
4. 消息队列
5. 大流量冲击---缓存--日志,牺牲强一致性
6. 数据校验---数据安全性的问题
7. 异构数据, 比如用户订单除了订单表中有之外,还可以按照用户的id分表进行放暑假
8. 缓存--银弹 CDN 接入层缓存,应用层缓存,JVM缓存,分布式缓存,静态化

高可用原则:

1. 集群
2. 限流
3. 熔断--可以是自动的,可用是手动的
4. 切换流量(流量分发)
5. 可回滚

数据原则

1. 幂等
2. 防重
3. 流程的自定义化
4. 状态,状态机
5. 备份
6. 数据安全审核
7. 日志

隔离

线程隔离

不同的请求分类按照不同的线程池处理

进程隔离

拆分,将不同的功能放到不同服务器,主要是区分单体和分布式

读写分离

机房分离

两地三中心

功能集群隔离

秒杀---->其实就是一个高并发的普通购买, 专门给秒杀部署一套集群

动静分离

动态资源和静态资源分离

限流机制

限流算法

不允许突然进来太多人,限制一个最大的数量

怎么知道同时当前进来了多少

加法 ,减法

令牌桶---->我有一个地方专门放一些数据的,比如线程池,我每隔500ms往线程池中放一个线程,不管你取出多少,我就是按照固定速度向桶中放令牌,如果出现特殊情况, 1个小时都没人访问,所以桶有上限

漏桶

你可以一次性进来很多个,但是我一次只拿两个处理,漏桶也有数量限制

应用限流

TPS/QOS阈值 tomcat可以配置

限制使用的资源数:当前应用最多允许建立100个数据库连接

限制某一个接口的访问量

一个接口 在固定时间只能最多访问你多少次

分布式限流

通过Redis中计数器去统计次数来限流

接入层或者应用层Nginx 限流

降级预案

自动和手动两种: 比如网络抖动导致的偶尔性的问题可以自动降级

警告,错误,严重错误的时候:可以使用自动或者人工降级 托底

如果缓存出现问题,降级到数据库取数据

如果数据库有问题,降级到缓存中取数据

写降级

页面降级,页面片段降级

配置中心

超时&重试

超时机制

接入层,应用层,tomcat,中间件,数据库,redis, js

回滚

事物回滚--->看情况

代码回滚

服务器版本的回滚

静态资源的回滚,尽量保证正确不要回滚,因为CDN deploy--->main-->1.0/

缓存

应用级缓存

缓存命中率

缓存的回收机制

基于空间---允许缓存最大使用空间,超过后回收, 基于容量的,缓存中最多放多少数据,超过也回收,基于时间

应用的缓存可以通过不同级别的引用对象来处理

回收算法:LRU ,LFU ,FIFO

多级缓存

数据过期问题

缓存更新

增量更新

缓存维度化

大的value 通过压缩或者拆分

keys *

缓存也要分布式

池化技术

连接池

线程池

MQ的消费者 可以多线程

异步调用

扩容

垂直扩容 :提升机器配置

水平扩容:多加机器

数据库:拆分 扩容

数据库表的结构: 没有外键, 反范式,允许适当的冗余数据

数据的备份表

队列

缓冲队列,用来削峰

任务队列

混合队列