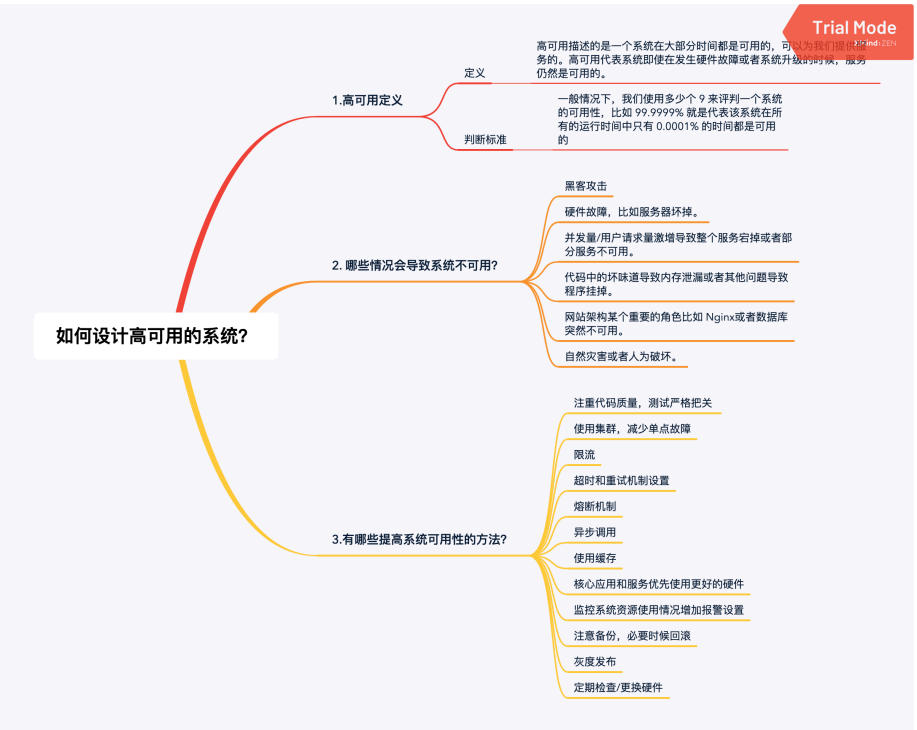
高可用分布式

集群

缓存

限流、削峰、熔断

异步调用(消息队列)



redis

为什么用redis

1 数据库，磁盘IO及网络开销大。Reids 是基于内存的读写操作，内存肯定比传统磁盘IO数据库快。

2 redis单线程 好处就是节省线程切换的开销

3 Reids 核心是基于非阻塞的IO多路复用机制

redis 底层结构

1 String：动态字符串

2 List：双向链表

3 sort set：HashMap和跳跃表(SkipList)来保证数据的存储和有序

HashMap里放的是成员到score的映射

跳跃表里存放的是所有的成员，排序依据是HashMap里存的score

跳跃表：通过简单的多层索引结构，实现简单，支持范围查找

```
ZADD runoobkey 4 'xxx'
```

redis持久化

RDB 在指定的时间间隔内，执行指定次数的写操作，则会将内存中的数据写入到磁盘中，添加一个时间点快照的数据，dump.rdb 文件，Redis 重启会通过加载 dump.rdb 文件恢复数据。

Redis 提供了 ***SAVE*** 和 ***BGSAVE*** 两个命令来生成 RDB 文件，区别是前者是阻塞的，后者是后台 ***fork*** 子进程进行，不会阻塞主进程处理命令请求

启动快、适合大规模、但备份时占内存

AOF 详解：采用日志的形式实时记录每个写操作，每次修改将日志追加到文件末尾

实时性好，但文件较大

redis集群Redis-cluster 无中心结构

Master-slave 模式中，Master 成为集群中至关重要的一个节点，Master 的稳定性决定整个系统的稳定性

为解决这一问题：主从不在同一个机器上。

数据分片 数据按照槽 (slot) 存储分布在多个 Redis 实例上

Redis Cluster **实现的功能：**

- 将数据分片到多个实例 (按照 slot 存储)；
- 集群节点宕掉会自动 failover；
- 提供相对平滑扩容 (缩容) 节点。

优点：

1. 无中心架构：三机房部署，其中一主一从构成一个分片，之间通过异步复制同步数据，异步复制存在数据不一致的时间窗口，保证高性能的同时牺牲了部分一致性一旦某个机房掉线，则分片上位于另一个机房的 *slave **会被提升为** master* 从而可以继续提供服务
2. 可扩展性：可线性扩展到 1000 多个节点，节点可动态添加或删除。
3. 降低运维成本，提高系统的扩展性和可用性。

异步同步复制数据，存在一些不一致，但保证了高性能

在宕机的 master 没有恢复前 Slave 要同时承担读写服务，虽然累一点，但是系统仍然能提供服务

弊端：

但是你会发现，单个机房如果距离很远， Master 1 的数据同步到 Slave2 上是跨机房，跨机房同步肯定不如同机房块，有延迟，牺牲掉一致性

CAP定理

一致性、可用性、分区容错性

Consistency、Availability、Partition Tolerance

在 Redis 去中心集群架构中，最优的解决方案还是满足可用性（A）和分区容错性（P）就要牺牲掉一致性（C），即跨机房同步数据存在延迟

使用消息队列有啥缺点？

消息的延迟会带来数据不一致问题

解决：最终一致性，不管消息延迟多久甚至丢失，设计一个离线定时任务，定期去扫描两个系统的数据，有不一致的情况就主动刷新同步，这样保证最终一致。

redis雪崩、穿透、击穿

雪崩

大量热点数据过期，请求涌入到后台，由于redis热点数据失效了，请求打入到数据库，引起数据库压力造成查询堵塞。

解决办法：

1 将缓存失效时间分散开，比如每个key的过期时间是随机，防止同一时间大量数据过期现象发生

2 让Redis数据永不过期

穿透

穿透是指，redis和数据库值都不存在，穿透到系统底层。

大规模发起key不存在，要检索的请求，导致系统压力大，最后故障

解决办法：

1 布隆过滤器，缓存无效的值，准确检索一个元素不在一个集合中。

2 返回空值，在Redis里set一个null value，目的在于同一个key再次请求时直接返回null，避免穿透

击穿

热点key的穿透，同一个key会被有成千上万次请求，压力更大直接压垮数据库。

解决办法：待补

布隆过滤器原理

用法：可以用于检索一个元素不在一个集合中。

原理：用一个足够好的Hash函数，将一个字符串映射到二进制位数组的某一位，

这样不管字符串如何长，都只有一位，因此存储空间就极大的提升了

解决冲突：一次采用多个Hash函数把数据映射到不同的位上

缺点是不能删除数据，因为hash了，删除数据可能会影响到其它数据

BloomFilter十分适合海量数据去重、过滤，尤其是当检测的字符串比较大时，极大地节省内存和存储空间，同时查询效率也十分高效

---

redis热key大value

Redis 热点数据问题，一般都是什么原因引起的？

1 访问量大的热key，热门的key比如热门商品，并发量大，QPS高

2 value大，导致占用内存大，网卡负载大

热点 key 或大 Value 会造成哪些故障呢

1 数据倾斜，热点key数据，会造成 Reids 集群负载不均衡（也就是数据倾斜）导致的故障，大量的写请求都会落到同一个 redis server 上。负载高

2 大value会造成网卡负载高，内存不足 服务器缓冲区不足导致get超时

Redis 缓存失效导致数据库层被击穿的连锁反应。

热点数据问题你是如何准确定位的

1 估计，活动统计

2 redis集群，使用一层代理统计

3 redis服务器收集 monitor命令监控单个分片的QPS，发现 QPS 倾斜到一定程度的节点进行 monitor，获取热点 key。

缺点：只能统计一个 Redis 节点的热点 key，对于集群需要汇总统计，统计时在内存暴涨和 Redis 性能的隐患。

如何解决热点数据问题

1 数据分片，让压力均匀分摊到集群的分片上，防止单个机器打挂

key拆分，二维的hash拆分field-value

2 迁移隔离

热点数据key迁移到独立的节点上，不会影响到整个集群的其他业务

其它：热点key限流、本地缓存(省去了一次远程的 IO 调用，但数据可能不一致)

如何解决存储的大 Value问题

大Value造成的问题复习：

由于 Redis 是单线程运行的，如果一次操作的 value 很大会对整个 redis 的响应时间造成负面影响

大 value 就是单个 value 占用内存较大，负载大，数据倾斜

解决方案——拆分：把value打散

1 将较大的key-value拆分成几个小的key-value，将操作压力平摊到多个 redis 实例中，降低对单个 redis 的 IO 影响

2 将分拆后的几个 key-value 存储在一个 hash 中，用get获取field属性

---

3 如果hash、set、zset、list 中存储过多的元素（field单独哈希，field分开存放）

核心思想就是将 value 打散，每次只 get 你需要的

开一个固定的桶大小，对field进行hash，用原来的hashKey + (hash(field) % 10000)

来存储value

```
newHashKey = hashKey + (hash(field) % 10000);
hset(newHashKey, field, value);
hget(newHashKey, field)
```

如何解决缓存与数据库双写一致性问题

根据业务需求是否需要一致性，

解决办法：把读请求和写请求串行化，放到先进先出的队列串行执行

如何解决**redis**的并发竞争可以问题

加分布式锁

1 基于redis 集群：可以使用setnx命令加锁

将 加锁 和 设置过期时间 用set命令一起设置，原子性

原理：setnx命令，意思就是 set if not exist:

命令在指定的 key 不存在时，为 key 设置指定的值

如果lockKey不存在，把key存入Redis。保存成功后如果result返回1，表示设置成功。如果非1，表示失败，别的线程已经设置过了

```
jedis.set(lockKey, requestId, SET_IF_NOT_EXIST,
SET_WITH_EXPIRE_TIME, expireTime))
```

```
Long result = jedis.setnx(lockKey, requestId);
if (result == 1) {
    // 第二步：设置过期时间
    jedis.expire(lockKey, expireTime);
}
```

del + 判断requestId来解锁

```
// 第一步： 使用 requestId 判断加锁与解锁是不是同一个客户端
if (requestId.equals(jedis.get(lockKey))) {
   // 第二步： 若在此时，这把锁突然不是这个客户端的，则会误解锁
   jedis.del(lockKey);
}
```

2 基于Zookeeper 集群模式。

负载均衡

什么是正向代理和反向代理

反向代理：用户请求，转发到不同的服务器

正向代理：公司内部员工去请求一个服务器，显示公司IP

代理有什么好处

1 控制流量分发，管理服务集群，起到负载均衡作用。

2 安全性和匿名性：通过拦截前端服务器的请求，反向代理服务器可以保护其身份

3 让 IO 和服务器分离，突破 IO 性能，提高服务器吞吐能力。

负载均衡算法

轮询(依次发放)、加权轮询

随机、加权随机

Hash(用ip去%服务器数量)保证同一个用户分配到相同的存有session的服务器

一致性hash，增加服务器数量，hash算法就失效了，对2^31-1取余数，环形

消息队列

消息队列的作用

消息队列主要解决应用耦合、异步消息、流量削锋等问题

实现高性能、高可用、可伸缩和最终一致性架构

比如：用户每消费一笔给用户增加一定积分，通知积分系统给用户加积分

比如：秒杀，通过消息队列去暂存秒杀请求

每个阶段依赖于多个子服务来共同完成，比如下单会依赖于购物车、订单预览、确认订单服务，这些子服务又会依赖于底层基础系统来完成其功能，异步去通知

消息队列的优点

1.异步处理：比如日志服务

2.应用系统解耦：单一职责原则，解耦，线程池就服务杂糅了

3.流量削峰：比如秒杀流量大，日志多

4.发布/订阅：比如用户付款成功，通知多个订阅者(订单系统、物流系统)

消息队列的缺点

1 消息丢失

2 消息重复

3 消息的顺序问题

4 消息的一致性问题

是否需要解决完全依赖业务场景

如果 Consumer 不能接受重复消息，那 Consumer 的接口可以设计成“**幂等**”

如何解决消息重复问题

一句话：消费端接口设计成幂等

比如订单系统支付送积分，根据用户订单流水号做强幂等，保存下来，每成功依次加了积分就记录下来，即使消息重复，只需要用流水号判记录有没有过。

加积分，和保存记录，要做成事务，ACID

如何解决消费的顺序问题

一句话：让同一个消息不要分区，并且设置成单线程处理

三个角度

1 生产者角度：一个一个同步发送，前一个消息确认后再发送，即消息顺序入队

2 消费端串行消费

3 topic不分区，让同一个topic主体都进入一个队列。

或则单通道，生产者—— 队列——消费者 一对一服务

在分布式环境下如果同一个topic进入多个分区，那多个分区之间肯定无法保证消息顺序了。

都会牺牲掉系统的性能和稳定性

如何做到topic不分区

RocketMQ：

Topic 用**Hash取模法**，相同Topic的Hash值肯定是一样的。

让同一个 Topic 同一个队列中，再使用同步发送，就能保证消息在一个分区有序

再用**RocketMQ提供的MessageQueueSelector队列选择机制**，选择不同队列

总结很多地方用到hash，主要目的在于分配到某一个相同的区域

1 如MQ，topic用hash每次分配到相同的区域，解决消息顺序问题

2 如负载均衡，hash，把用户请求分配到相同的服务器节点

3 如redis解决大value问题，对hash、list多值 把不同的value二次hash到不同位置