微服务培训纪要

# 基础概念

1、单体jvm->分布式（多jvm）

2、微服务：网络通信，各个服务之间的调用；多实例服务（负载均衡）

3、单体软件-缓存（hashmap，锁等）；分布式缓存/锁，多个服务实例都在用（需要在多实例间同步）

4、增删改一般没有性能问题，select可能出现性能问题，并发更容易导致；使用缓存来解决（空间换时间）；时间换空间：gzip压缩静态文件

5、jvm缓存，redis缓存（网络），本地缓存，分布式缓存

6、redis缓存：标配了，受到物理内存大小的限制；

哨兵模式-主从复制，最多内存的80%；

集群模式，路由表（hash一个key）

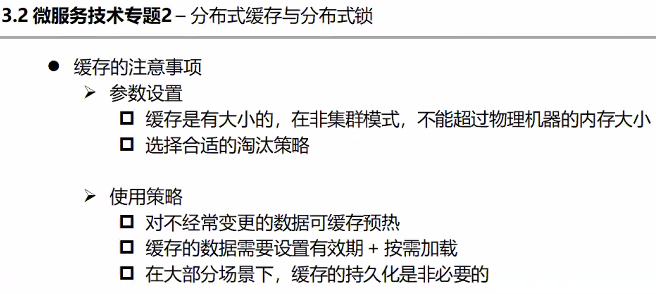
支持很多丰富的数据结构，set、list、map（打卡，布隆过滤器）

单线程模型：命令执行，一个命令执行完才执行下一个命令；程序也是多线程的（发请求、接受数据；以前读写数据包是单线程），高性能：避免了线程上下文切换的时间消耗，提高了性能；充分发挥多核cpu的能力

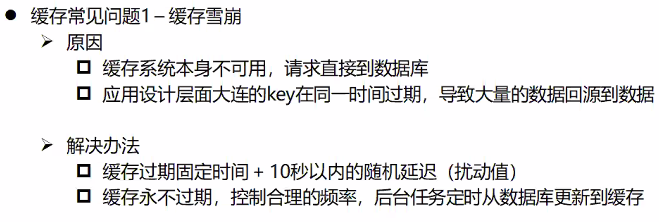
持久化：需要权衡使用它，一般是用来查询做缓存，很少用到，比如下次启动redis直接读取持久化文件加载到内存；开启了会耗性能，数据一致性要求高的环境不适合使用；配置文件中一般是关掉的

存储层：追加模式AOF、快照模式RDB

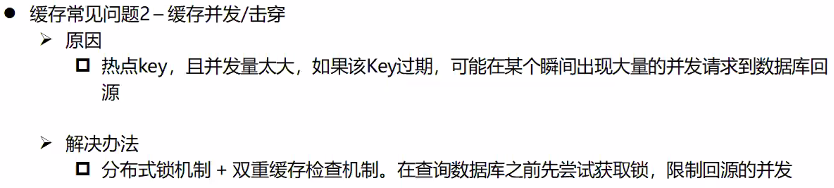
网络层：epoll网络框架，只有一个线程去轮训到达的:请求->处理；io多路复用



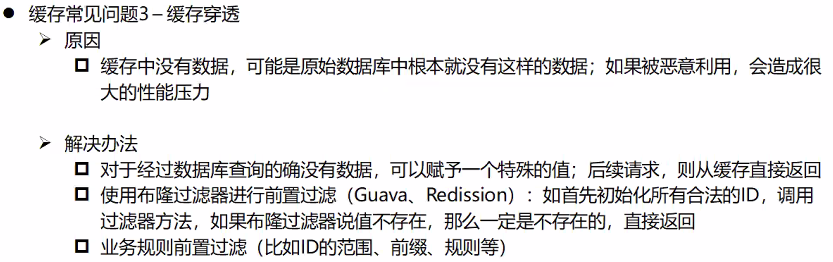
缓存：大小<内存，数据使用策略，预缓存（恶汉），要设置有效期+按需加载；大部分情况下不使用持久化，它是有一批数据后才一次性flush到磁盘，可能在这之间断电，那么缓存就丢失了



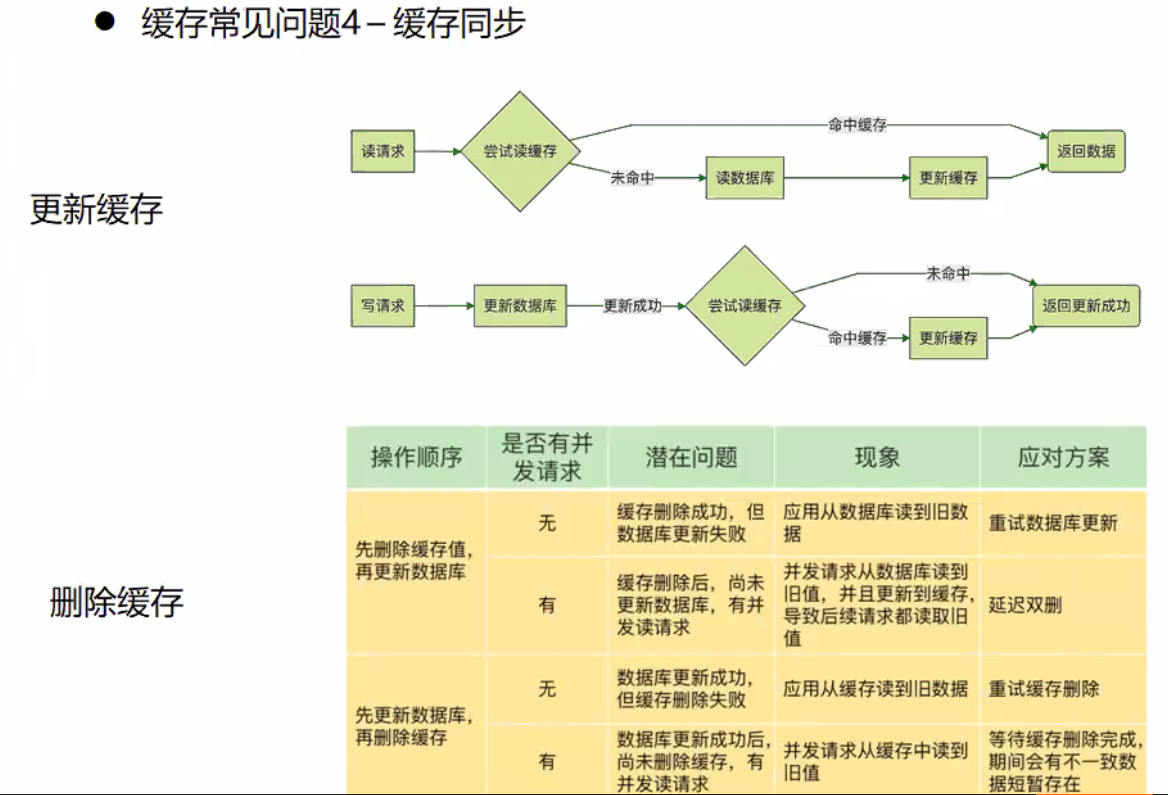
缓存雪崩：缓存不可用了，数据直接到数据库，没有缓冲作用了，容易导致数据库崩掉



双重缓存检查机制：先查缓存，没有则先去获取锁，然后去数据库拿数据写到缓存；其他线程也是先去获取锁，获取到了后还要再判断一次缓存有没有数据，有的话直接返回



分布式锁：单机-Guava，分布式-Redission



缓存同步：更新数据后更新缓存，删除数据

# SpringCache

添加缓存：@Cacheable(value=**"deeCache"**,key=**"#authcode+'-'+#authpwd"**) *//获取的数据加入缓存*

@CacheEvict(value=**"deeCache"**,allEntries=**true**)*// 清空 deeCache 缓存  
@CacheEvict(value="deeCache",key="#trustBean.getCode()+'-'+#trustBean.getPassword()")// 删除指定key的 deeCache 缓存*

# Redis

安装->启动

# Spring整合redis

依赖对应的starter，配置redis的信息；

序列化：RedisTemplate配置

# Redisson分布式锁

工程加入一个redission配置文件redisson.ymal（内容很多），锁就通过redission.getXXLock()来获取各种锁