本期视频的效果其实比我预想的要好很多，我预料到大家整体评价会比较不错，但是没想到一觉醒来就上了热门，我还得意的跟挚友炫耀。

人民网跟我互相关注了你敢信？虽然是重庆的哈哈，不过也很开心，微信也灰度到我的付费阅读了，好事都是接二连三呀。

本期视频我前后的时间大概是做了2周左右，因为是工作之余出的，大家发现我其实文章也在出，还做了个缓存的技术视频，所以几乎极限的压缩了自己的业余时间了吧，不然也不至于几天都是11点去跑步减肥。

不过都是值得的嘛，很开心。

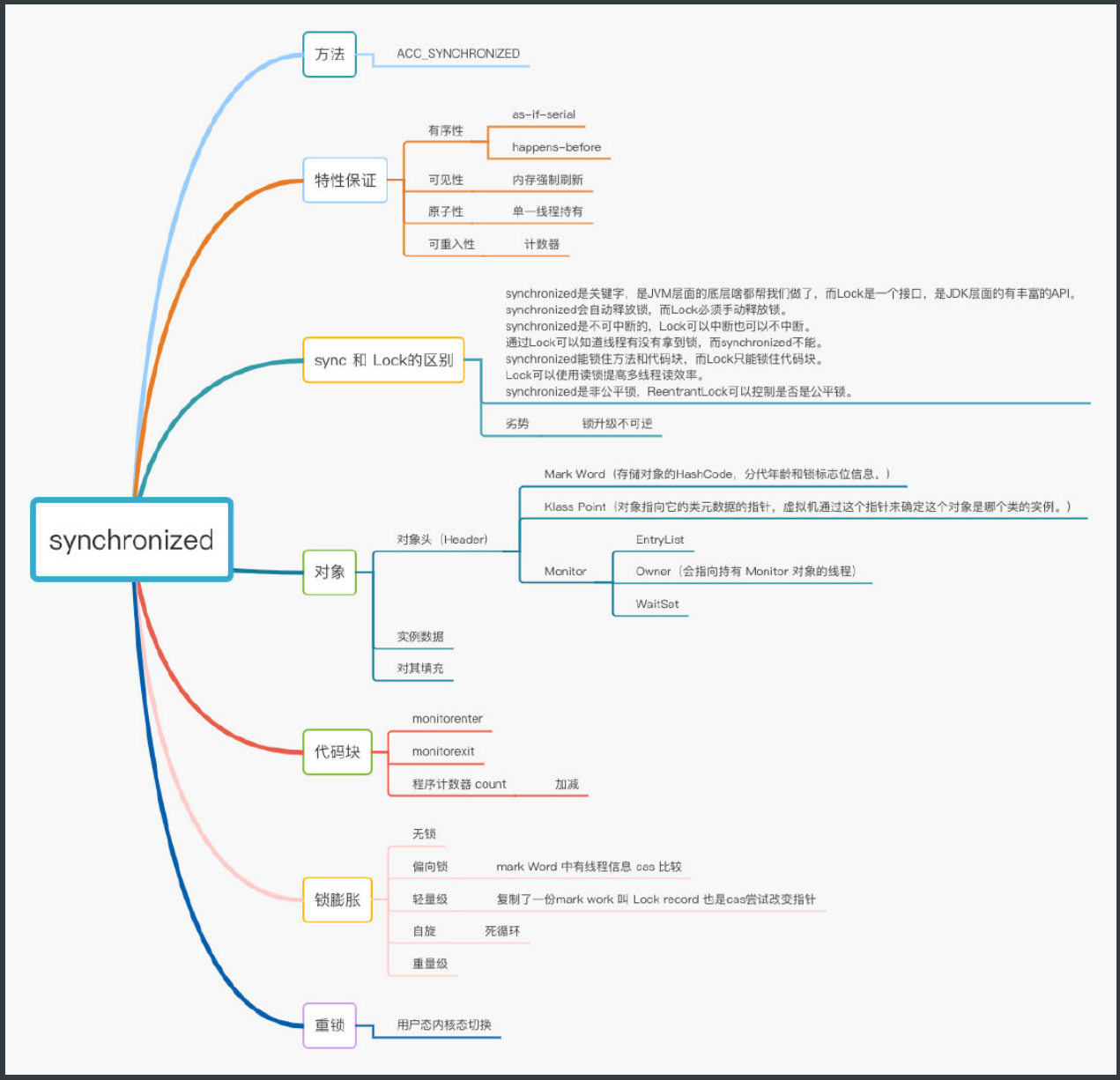
内容的所有脑图我都会放出来，我甚至做了文字版的复习PDF，就是那个我怼了一个月的知识点汇总的PDF，再肝一次回馈大家也可。

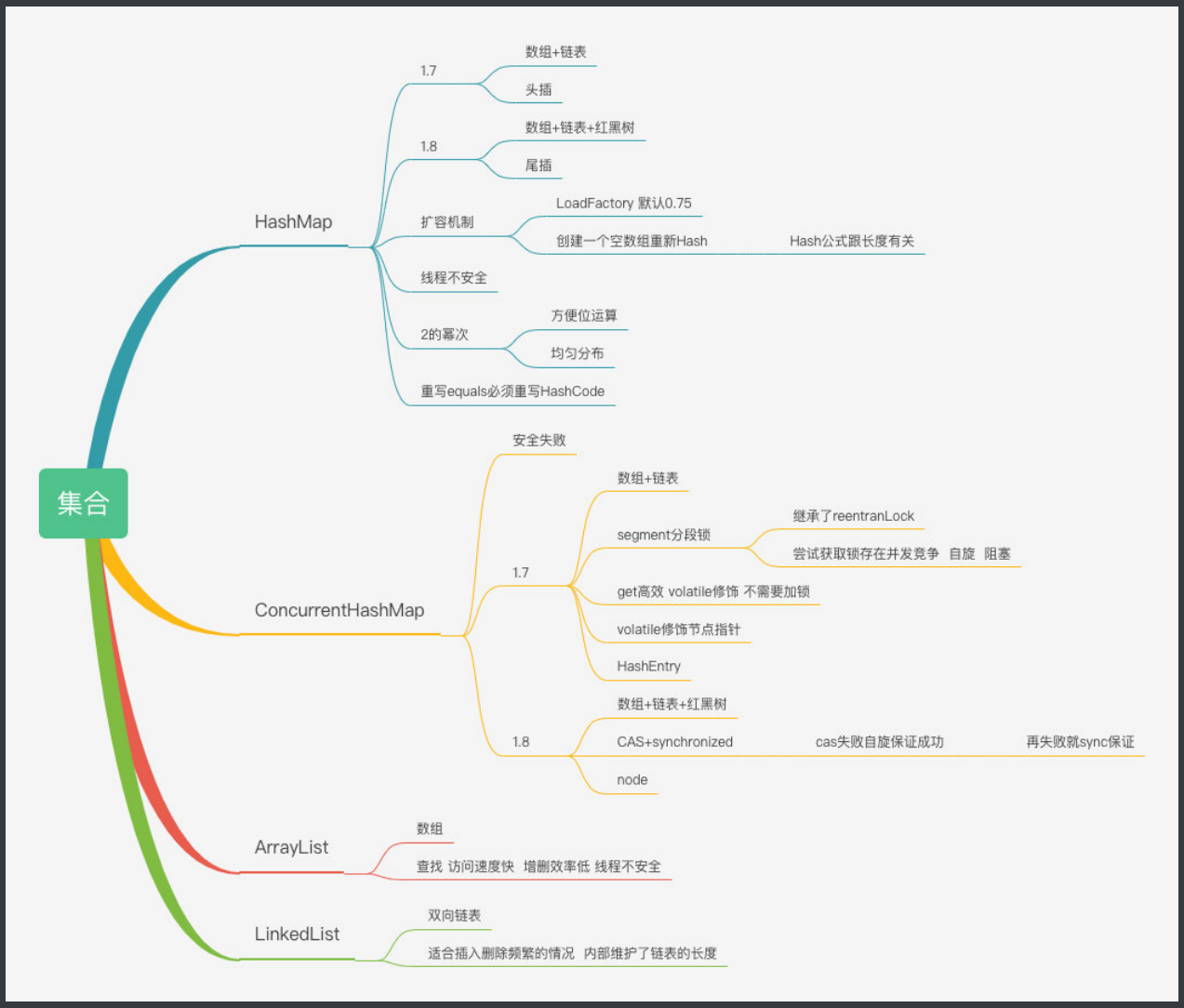
接下来的时间呢输出可能会少点，因为618嘛，会多少忙一点，忙完了我继续输出，我想好了很多很多点子，你们放心，一个不落，我都会输出的。

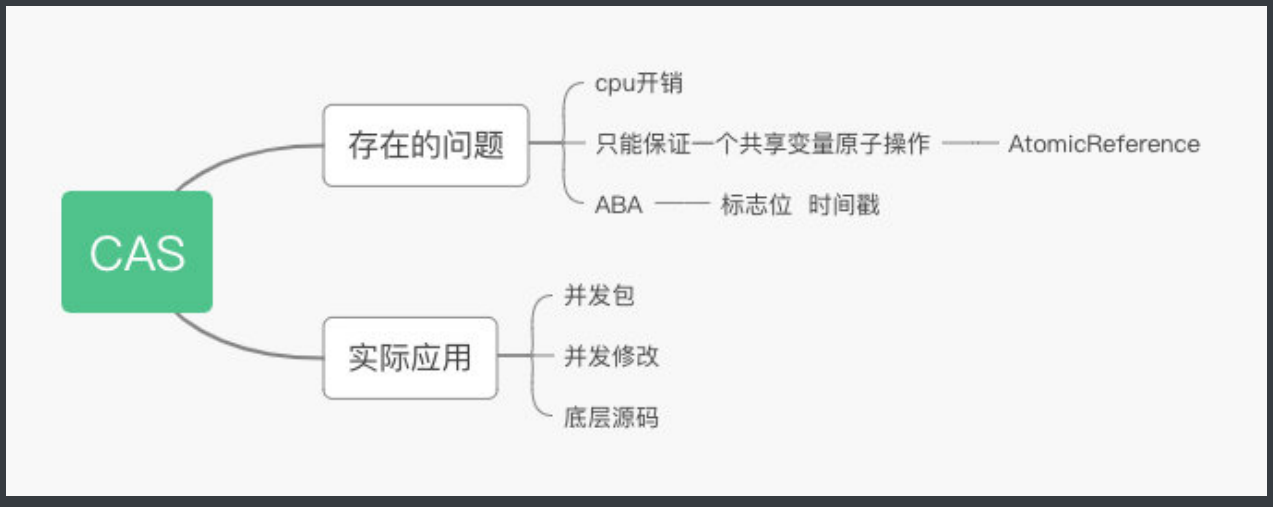
我的计划里面不知道有没有你喜欢看的，没有可以给我留言，好了，视频里面的脑图，我会直接贴在这里，最后我也会把所有的资料打包放在我的网盘，GitHub老地方也会有（新来的朋友可能不会知道，大家去找一次就知道了 https://github.com/AobingJava/JavaFamily）。

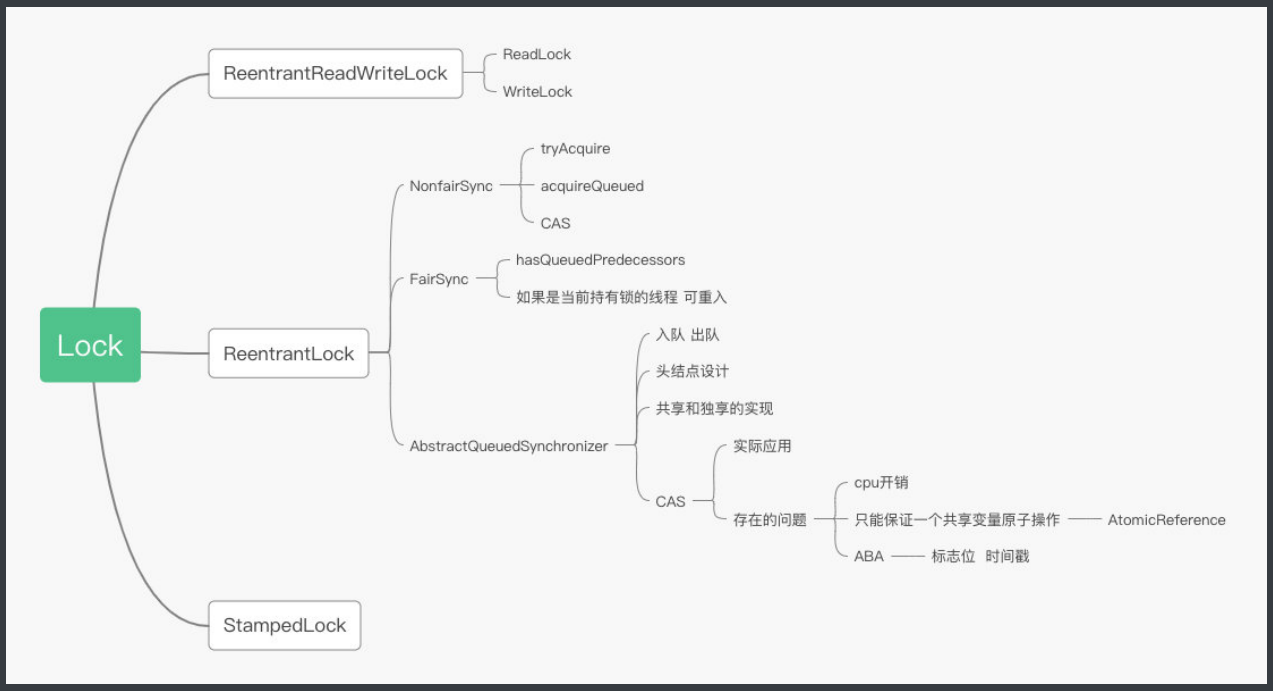

昂，我24岁了
福利
Java/后端学习路线
整理的书单(附个人喜欢的文学书)
整理好用的工具集
通用的学习方法
IDEA破解(请勿传播)
电子书(请勿传播)
面试资料(持续更新)
简历模板(欢迎补充)
概要设计模板
MyAll

视频所有脑图：

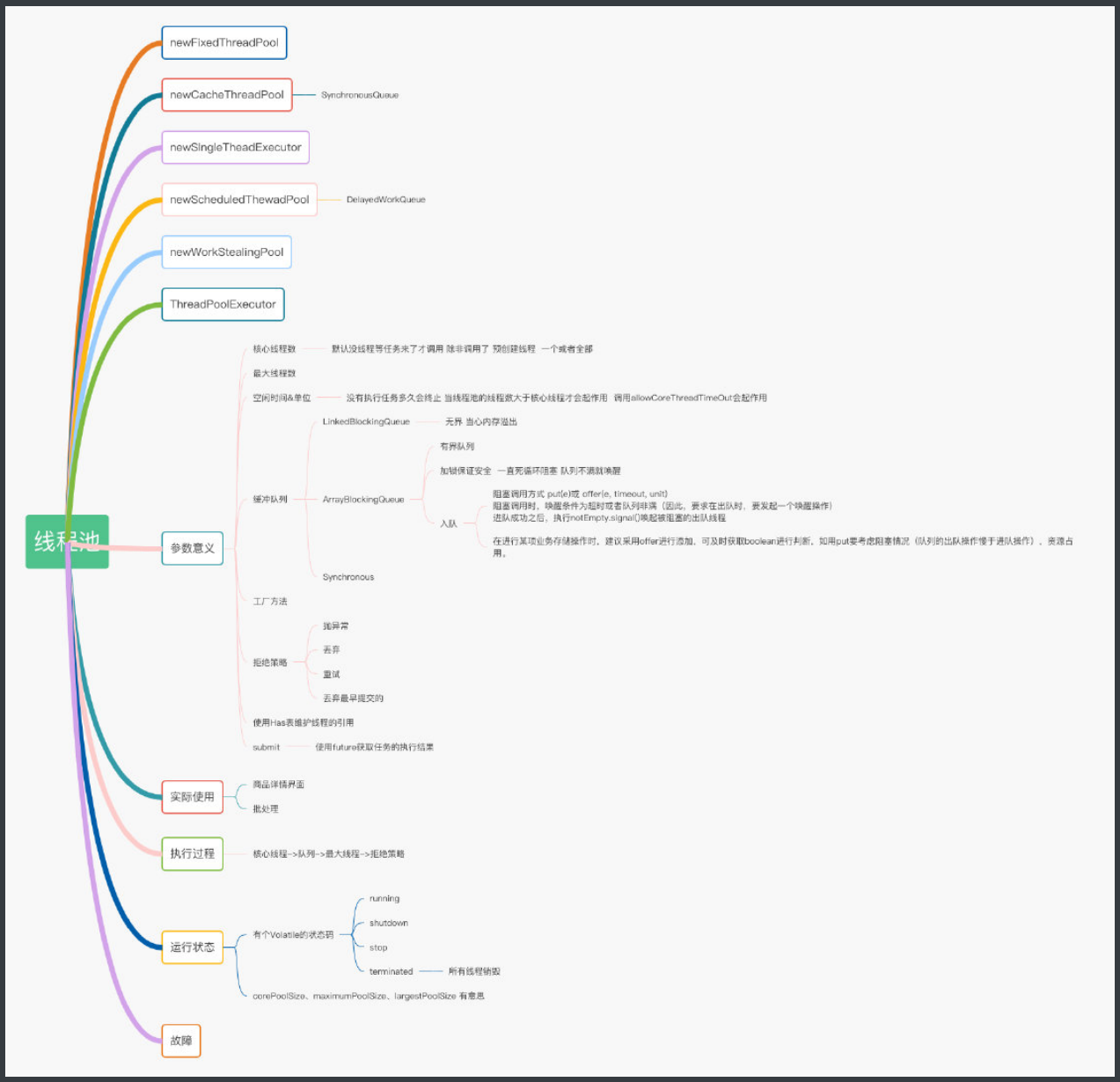

synchronized
方法
ACC_SYNCHRONIZED
特性保证
有序性
as-if-serial
happens-before
可见性
内存强制刷新
原子性
单一线程持有
可重入性
计数器
sync 和 Lock的区别
synchronized是关键字，是JVM层面的底层啥都帮我们做了，而Lock是一个接口，是JDK层面的有丰富的API。
synchronized会自动释放锁，而Lock必须手动释放锁。
synchronized是不可中断的，Lock可以中断也可以不中断。
通过Lock可以知道线程有没有拿到锁，而synchronized不能。
synchronized能锁住方法和代码块，而Lock只能锁住代码块。
Lock可以使用读锁提高多线程读效率。
synchronized是非公平锁，ReentrantLock可以控制是否是公平锁。
劣势
锁升级不可逆
对象
对象头（Header）
Mark Word（存储对象的HashCode，分代年龄和锁标志位信息。）
Klass Point（对象指向它的类元数据的指针，虚拟机通过这个指针来确定这个对象是哪个类的实例。）
Monitor
EntryList
Owner（会指向持有 Monitor 对象的线程）
WaitSet
实例数据
对其填充
代码块
monitorenter
monitorexit
程序计数器 count
加减
锁膨胀
无锁
偏向锁
mark Word 中有线程信息 cas 比较
轻量级
复制了一份mark work 叫 Lock record 也是cas尝试改变指针
自旋
死循环
重量级
重锁
用户态内核态切换

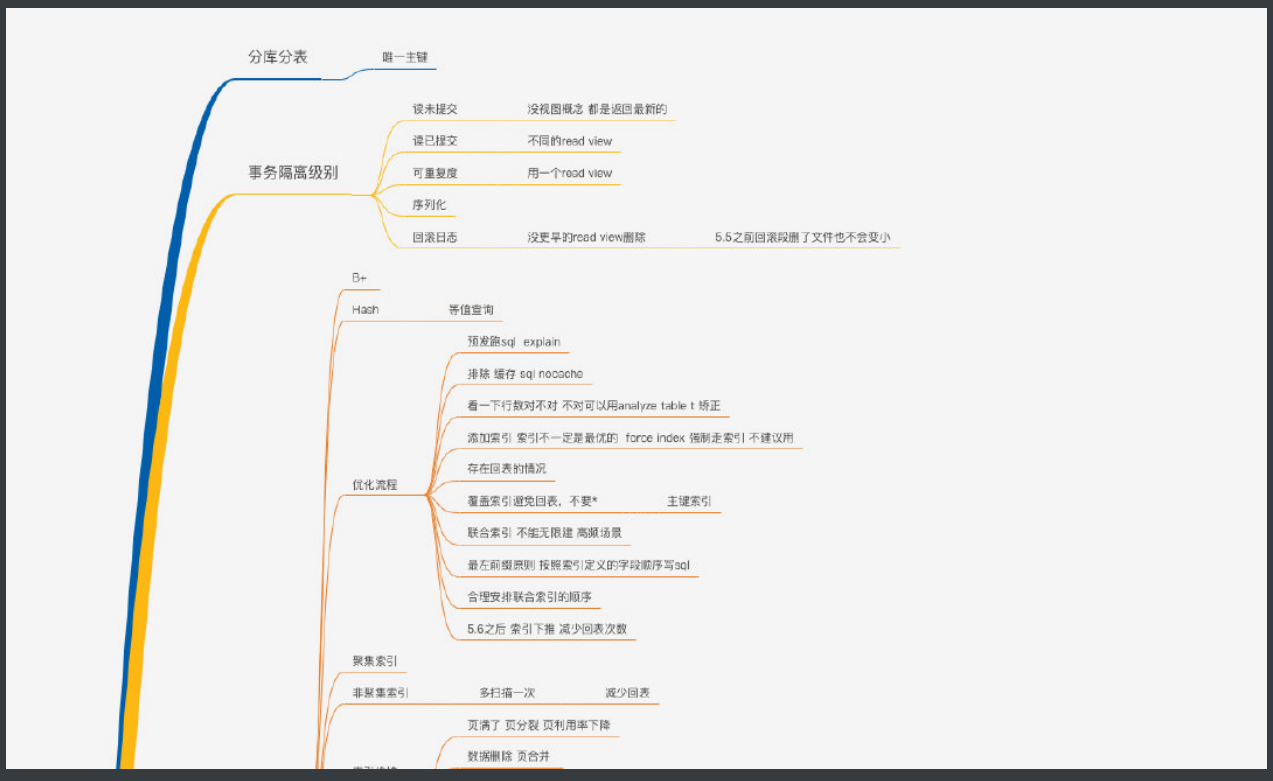

集合
HashMap
1.7
数组+链表
头插
1.8
数组+链表+红黑树
尾插
扩容机制
LoadFactory 默认0.75
创建一个空数组重新Hash
Hash公式跟长度有关
线程不安全
2的幂次
方便位运算
均匀分布
重写equals必须重写HashCode
ConcurrentHashMap
安全失败
1.7
数组+链表
segment分段锁
继承了reentranLock
尝试获取锁存在并发竞争 自旋 阻塞
get高效 volatile修饰 不需要加锁
volatile修饰节点指针
HashEntry
1.8
数组+链表+红黑树
CAS+synchronized
cas失败自旋保证成功
再失败就sync保证
node
ArrayList
数组
查找 访问速度快 增删效率低 线程不安全
LinkedList
双向链表
适合插入删除频繁的情况 内部维护了链表的长度

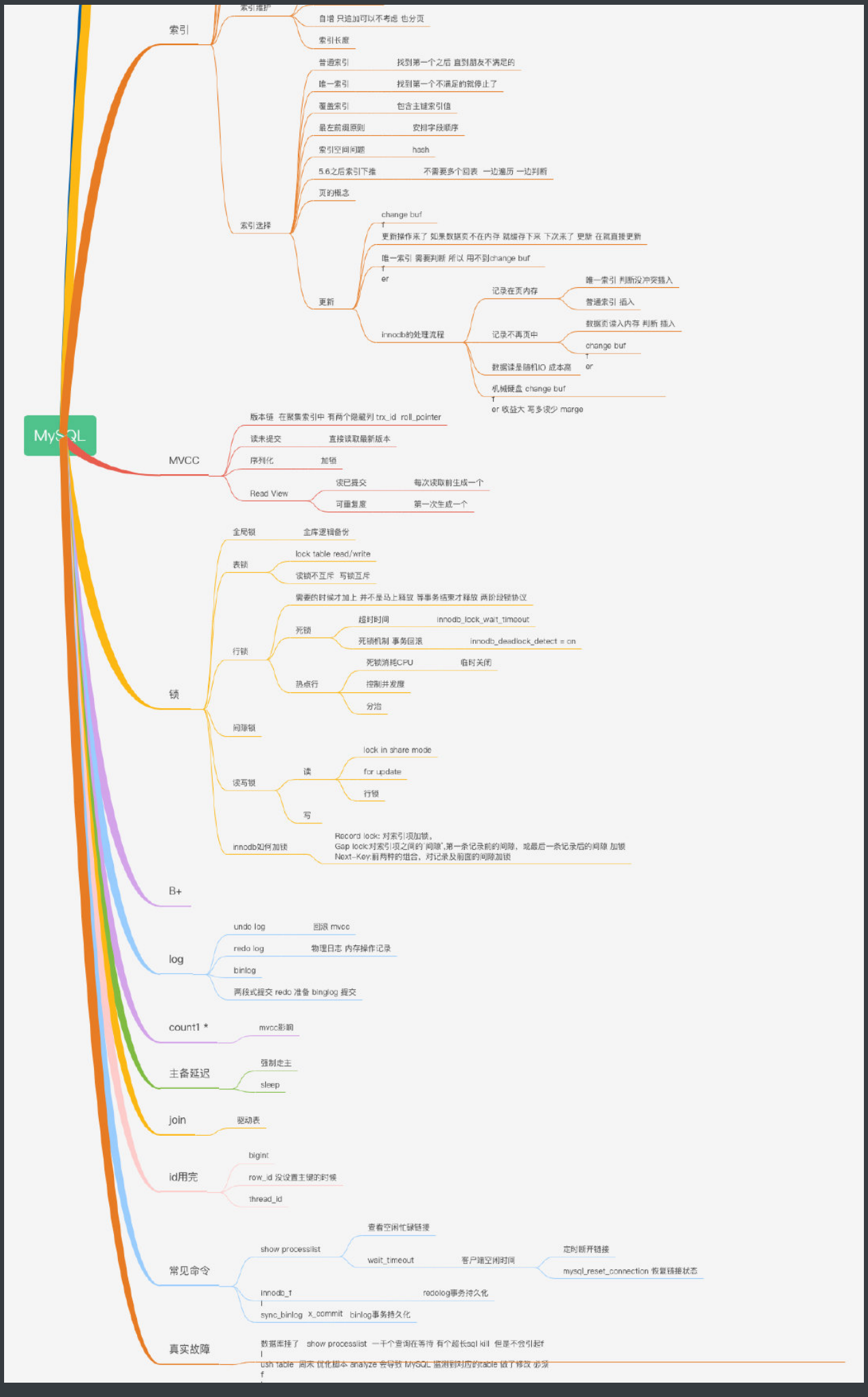

CAS
存在的问题
cpu开销
只能保证一个共享变量原子操作
AtomicReference
ABA
标志位 时间戳
实际应用
并发包
并发修改
底层源码

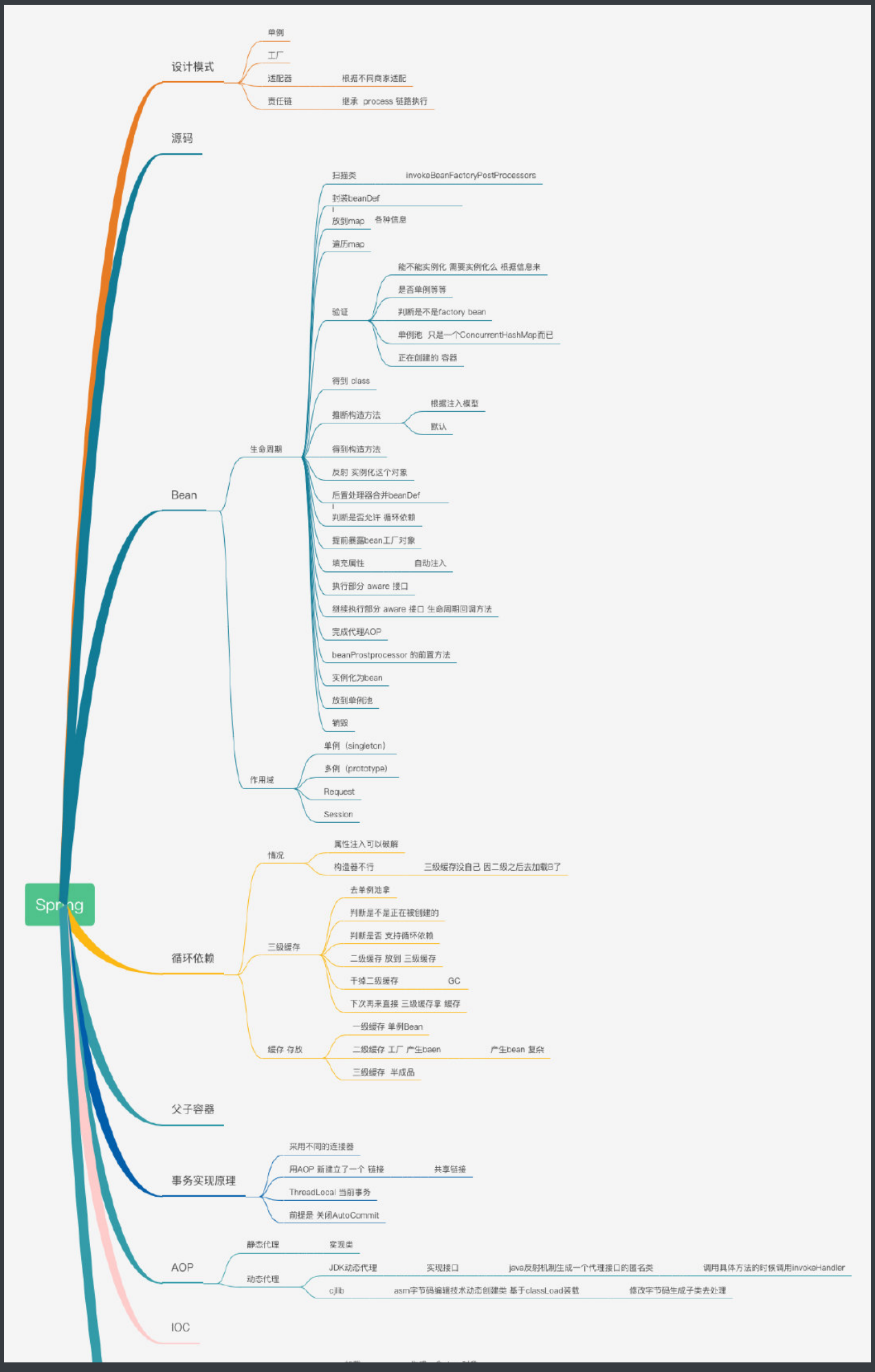

Lock
ReentrantReadWriteLock
ReadLock
WriteLock
ReentrantLock
NonfairSync
tryAcquire
acquireQueued
CAS
FairSync
hasQueuedPredecessors
如果是当前持有锁的线程 可重入
AbstractQueuedSynchronizer
入队 出队
头结点设计
共享和独享的实现
CAS
实际应用
存在的问题
cpu开销
只能保证一个共享变量原子操作
AtomicReference
ABA
标志位 时间戳
StampedLock

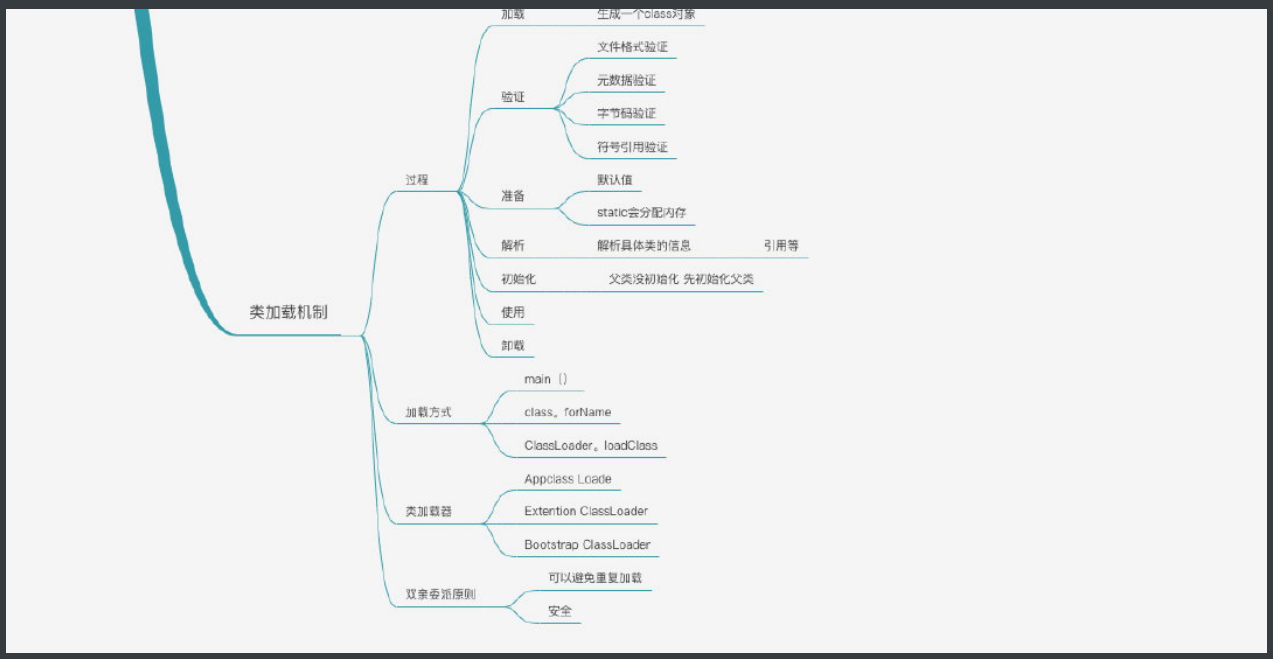

Volatile
MESI
当CPU写数据时，如果发现操作的变量是共享变量，即在其他CPU中也存在该变量的副本，会发出信号通知其他CPU将该变量的缓存行置为无效状态，因此当其他CPU需要读取这个变量时，发现自己缓存中缓存该变量的缓存行是无效的，那么它就会从内存重新读取
锁bus
volitale会一直嗅探 cas 不断循环无效交互 导致带宽达到峰值
总线风暴
Java内存模型JMM
高速缓存
可见性
嗅探机制 强制失效
处理器嗅探总线
有序性
禁止指令重排序
lock 前缀指令 内存屏障
源代码->编译器优化重排序->指令级并行重排序->内存系统重排序->最终执行的指令序列
happens-before
volatile变量规则：对一个volatile域的写，happens-before于任意后续对这个volatile域的读。
as-if-serial
跳出死循环
AtomicInteger

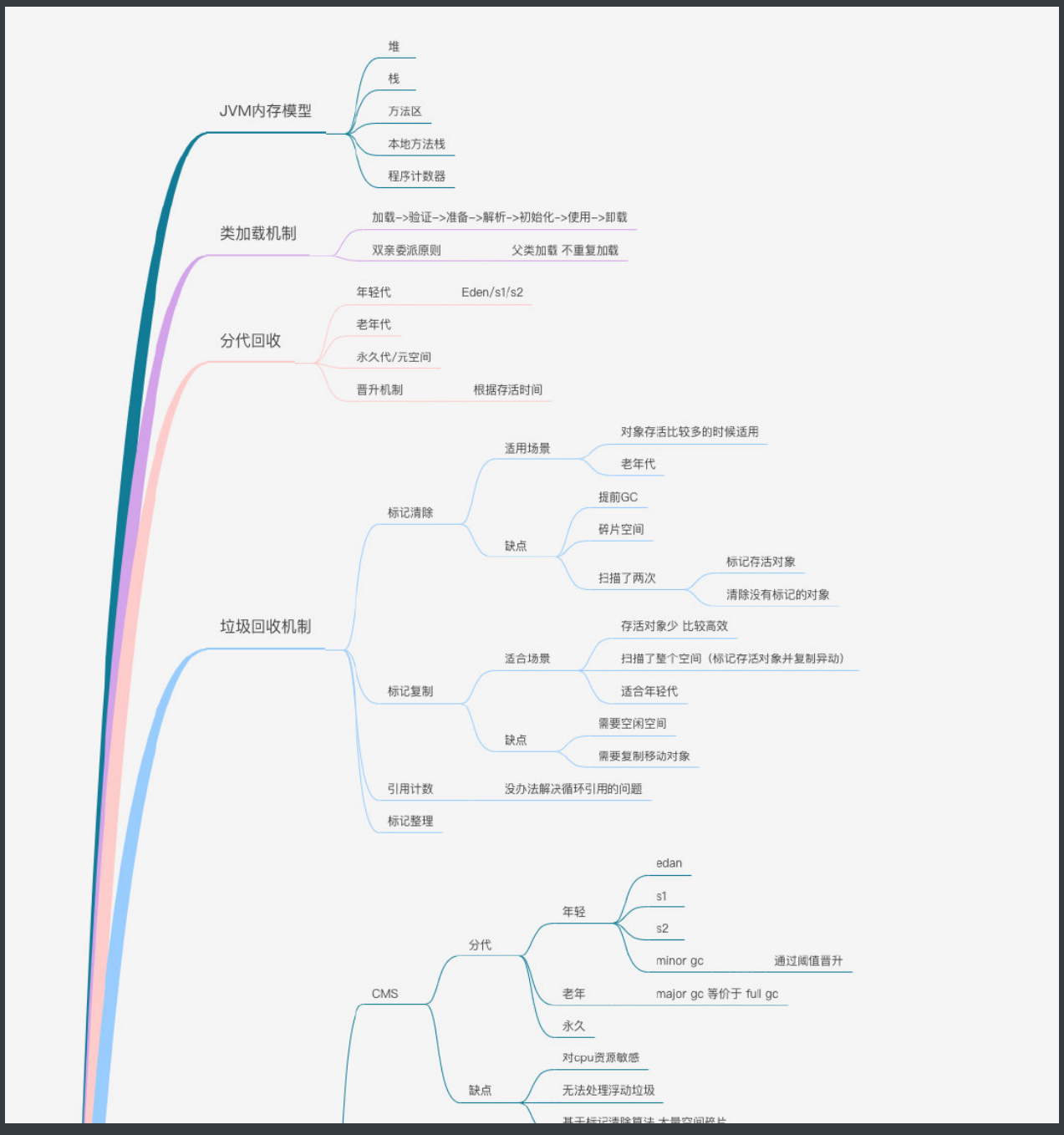

线程池
newFixedThreadPool
newCacheThreadPool
SynchronousQueue
newSingleTheadExecutor
newScheduledThewadPool
DelayedWorkQueue
newWorkStealingPool
ThreadPoolExecutor
参数意义
核心线程数
默认没线程等任务来了才调用 就非调用了 预创建线程 一个或者全部
最大线程数
空闲时间&单位
没有执行任务多久会终止 当线程池的线程数大于核心线程才会起作用 调用allowCoreThreadTimeOut会起作用
缓冲队列
LinkedBlockingQueue
无界 当心内存溢出
ArrayBlockingQueue
有界队列
加锁保证安全 一直死循环阻塞 队列不满就唤醒
入队
阻塞调用方式 put(e)或 offer(e, timeout, unit)
阻塞调用时，唤醒条件为超时或者队列非满（因此，要求在出队时，要发起一个唤醒操作）
进队成功之后，执行notEmpty.signal()唤起被阻塞的出队线程
在进行某项业务存储操作时，建议采用offer进行添加，可及时获取boolean进行判断，如用put要考虑阻塞情况（队列的出队操作慢于进队操作），资源占用。
Synchronous
工厂方法
拒绝策略
抛异常
丢弃
重试
丢弃最早提交的
使用Has表维护线程的引用
submit
使用future获取任务的执行结果
实际使用
商品详情界面
批处理
执行过程
核心线程->队列->最大线程->拒绝策略
运行状态
有个Volatile的状态码
running
shutdown
stop
terminated
所有线程销毁
corePoolSize、maximumPoolSize、largestPoolSize 有意思
故障

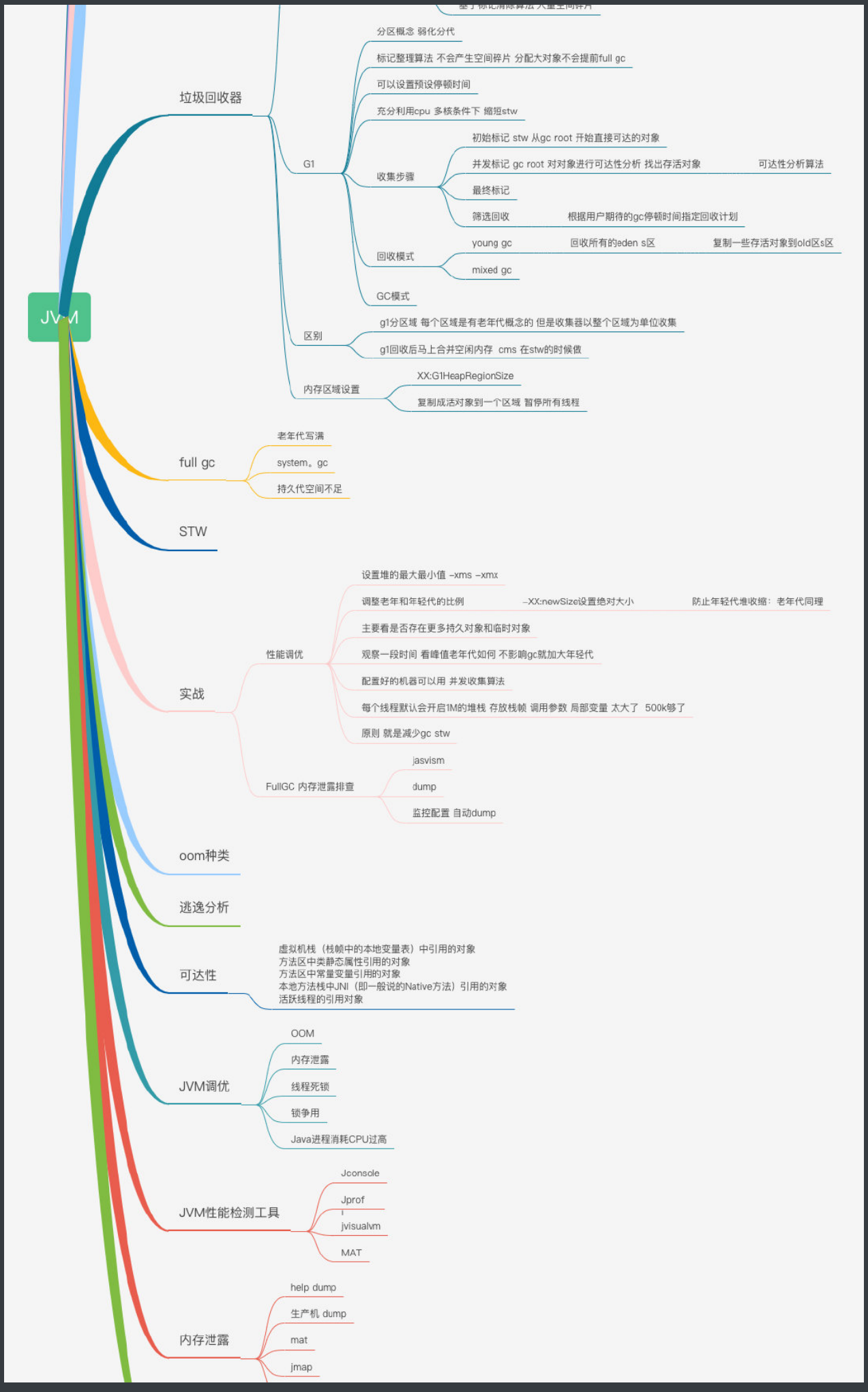

分库分表
唯一主键
事务隔离级别
读未提交
没视图概念 都是返回最新的
读已提交
不同的read view
可重复度
用一个read view
序列化
回滚日志
没更早的read view删除
5.5之前回滚段删了文件也不会变小
B+
Hash
等值查询
优化流程
预发跑sql explain
排除 缓存 sql noocache
看一下行数对不对 不对可以用analyze table t 矫正
添加索引 索引不一定是最优的 force index 强制走索引 不建议用
存在回表的情况
覆盖索引避免回表，不要*
主键索引
联合索引 不能无限建 高频场景
最左前缀原则 按照索引定义的字段顺序写sql
合理安排联合索引的顺序
5.6之后 索引下推 减少回表次数
聚集索引
非聚集索引
多扫描一次
减少回表
页满了 页分裂 页利用率下降
数据删除 页合并


MySQL
索引
索引维护
自增 只追加可以不考虑 也分页
索引长度
索引选择
普通索引 找到第一个之后 直到朋友不满足的
唯一索引 找到第一个不满足的就停止了
覆盖索引 包含主键索引值
最左前缀原则 安排字段顺序
索引空间问题 hash
5.6之后索引下推 不需要多个回表 一边遍历 一边判断
页的概念
更新
change buf
更新操作来了 如果数据页不在内存 就缓存下来 下次来了 更新 在就直接更新
唯一索引 需要判断 所以 用不到change buf
er
innodb的处理流程
记录在页内存
唯一索引 判断没冲突插入
普通索引 插入
记录不再页中
数据页读入内存 判断 插入
change buf
er
数据读是随机IO 成本高
机械硬盘 change buf
er 收益大 写多读少 merge
MVCC
版本链 在聚集索引中 有两个隐藏列 trx_id roll_pointer
读未提交 直接读取最新版本
序列化 加锁
Read View
读已提交 每次读取前生成一个
可重复度 第一次生成一个
锁
全局锁 全库逻辑备份
表锁
lock table read/write
读锁不互斥 写锁互斥
行锁
需要的时候才加上 并不是马上释放 等事务结束才释放 两阶段锁协议
死锁
超时时间 innodb_lock_wait_timeout
死锁机制 事务回滚 innodb_deadlock_detect = on
热点行
死锁消耗CPU 临时关闭
控制并发度
分治
间隙锁
读写锁
读
lock in share mode
for update
行锁
写
innodb如何加锁
Record lock: 对索引项加锁，
Gap lock:对索引项之间的'间隙',第一条记录前的间隙，或最后一条记录后的间隙 加锁
Next-Key:前两种的组合，对记录及前面的间隙加锁
B+
log
undo log 回退 mvcc
redo log 物理日志 内存操作记录
binlog
两段式提交 redo 准备 binglog 提交
count1 *
mvcc影响
主备延迟
强制走主
sleep
join
驱动表
id用完
bigint
row_id 没设置主键的时候
thread_id
常见命令
show processlist
查看空闲忙碌链接
wait_timeout 客户端空闲时间
定时断开链接
mysql_reset_connection 恢复链接状态
innodb_f redo log事务持久化
l
sync_binlog x_commit binlog事务持久化
真实故障
数据库挂了 show processlist 一千个查询在等待 有个超长sql kill 但是不会引起f
l
ush table 周末 优化脚本 analyze 会导致 MySQL 监测到对应的table 做了修改 必须
f
l


Spring
设计模式
单例
工厂
适配器
根据不同商家适配
责任链
继承 process 链路执行
源码
Bean
生命周期
扫描类
invokeBeanFactoryPostProcessors
封装beanDef
放到map
各种信息
遍历map
验证
能不能实例化 需要实例化么 根据信息来
是否单例等等
判断是不是factory bean
单例池 只是一个ConcurrentHashMap而已
正在创建的 容器
得到 class
推断构造方法
根据注入模型
默认
得到构造方法
反射 实例化这个对象
后置处理器合并beanDef
判断是否允许 循环依赖
提前暴露bean工厂对象
填充属性
自动注入
执行部分 aware 接口
继续执行部分 aware 接口 生命周期回调方法
完成代理AOP
beanProstprocessor 的前置方法
实例化为bean
放到单例池
销毁
作用域
单例 (singleton)
多例 (prototype)
Request
Session
循环依赖
情况
属性注入可以破解
构造器不行
三级缓存没自己 因二级之后去加载B了
三级缓存
去单例池拿
判断是不是正在被创建的
判断是否 支持循环依赖
二级缓存 放到 三级缓存
干掉二级缓存
GC
下次再来直接 三级缓存拿 缓存
缓存 存放
一级缓存 单例Bean
二级缓存 工厂 产生bean
产生bean 复杂
三级缓存 半成品
父子容器
事务实现原理
采用不同的连接器
用AOP 新建立了一个 链接
共享链接
ThreadLocal 当前事务
前提是 关闭AutoCommit
AOP
静态代理
实现类
动态代理
JDK动态代理
实现接口
java反射机制生成一个代理接口的匿名类
调用具体方法的时候调用invokeHandler
cjlib
asm字节码编辑技术动态创建类 基于classLoad装载
修改字节码生成子类去处理
IOC

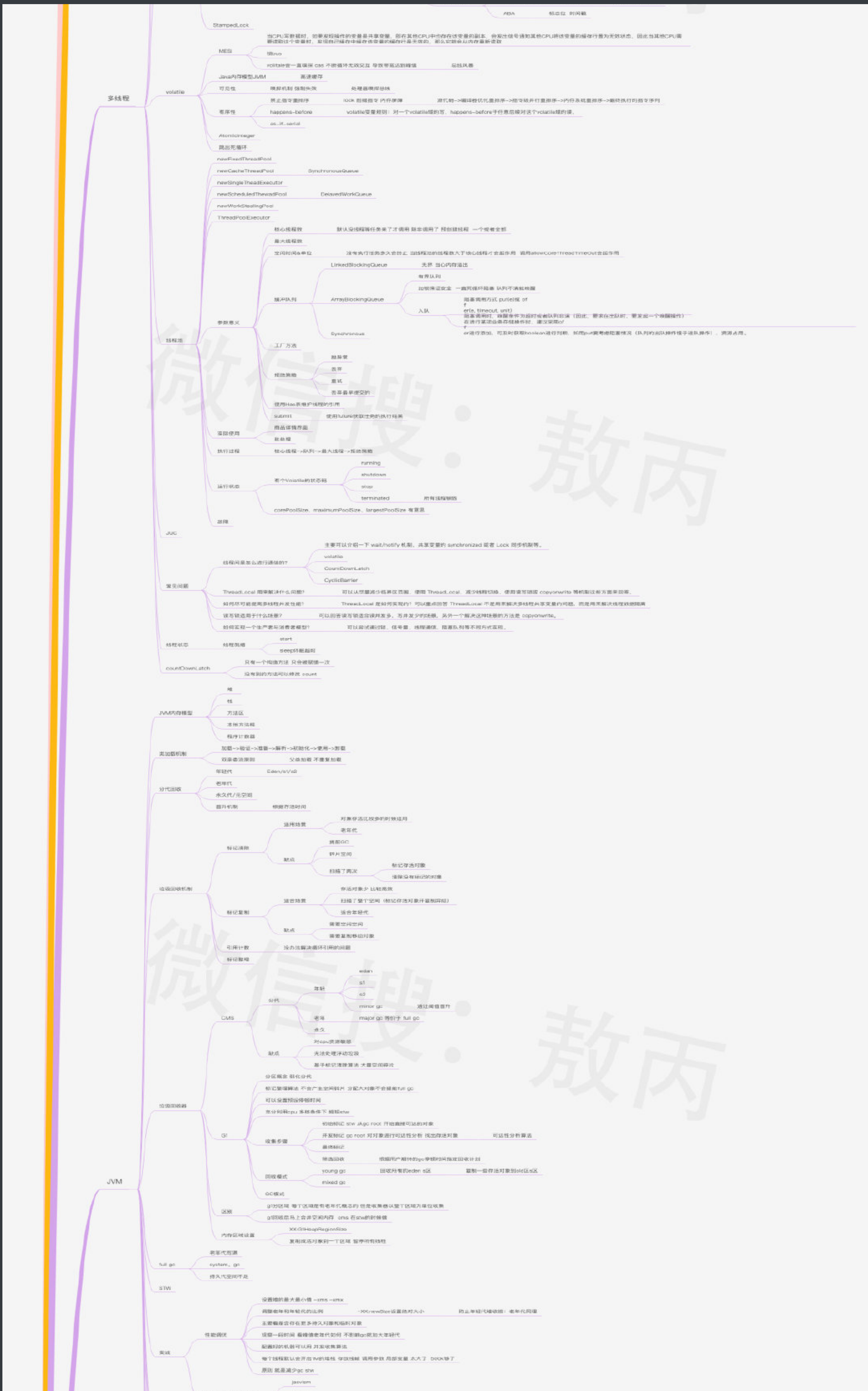

类加载机制
过程
加载
生成一个class对象
验证
文件格式验证
元数据验证
字节码验证
符号引用验证
准备
默认值
static会分配内存
解析
解析具体类的信息
引用等
初始化
父类没初始化 先初始化父类
使用
卸载
加载方式
main ()
class。forName
ClassLoader。loadClass
类加载器
Appclass Loade
Extention ClassLoader
Bootstrap ClassLoader
双亲委派原则
可以避免重复加载
安全


JVM内存模型
堆
栈
方法区
本地方法栈
程序计数器
类加载机制
加载->验证->准备->解析->初始化->使用->卸载
双亲委派原则
父类加载 不重复加载
分代回收
年轻代
Eden/s1/s2
老年代
永久代/元空间
晋升机制
根据存活时间
垃圾回收机制
标记清除
适用场景
对象存活比较多的时候适用
老年代
缺点
提前GC
碎片空间
扫描了两次
标记存活对象
清除没有标记的对象
标记复制
适合场景
存活对象少 比较高效
扫描了整个空间（标记存活对象并复制异动）
适合年轻代
缺点
需要空闲空间
需要复制移动对象
引用计数
没办法解决循环引用的问题
标记整理
CMS
分代
年轻
edan
s1
s2
minor gc
通过阈值晋升
老年
major gc 等价于 full gc
永久
缺点
对cpu资源敏感
无法处理浮动垃圾

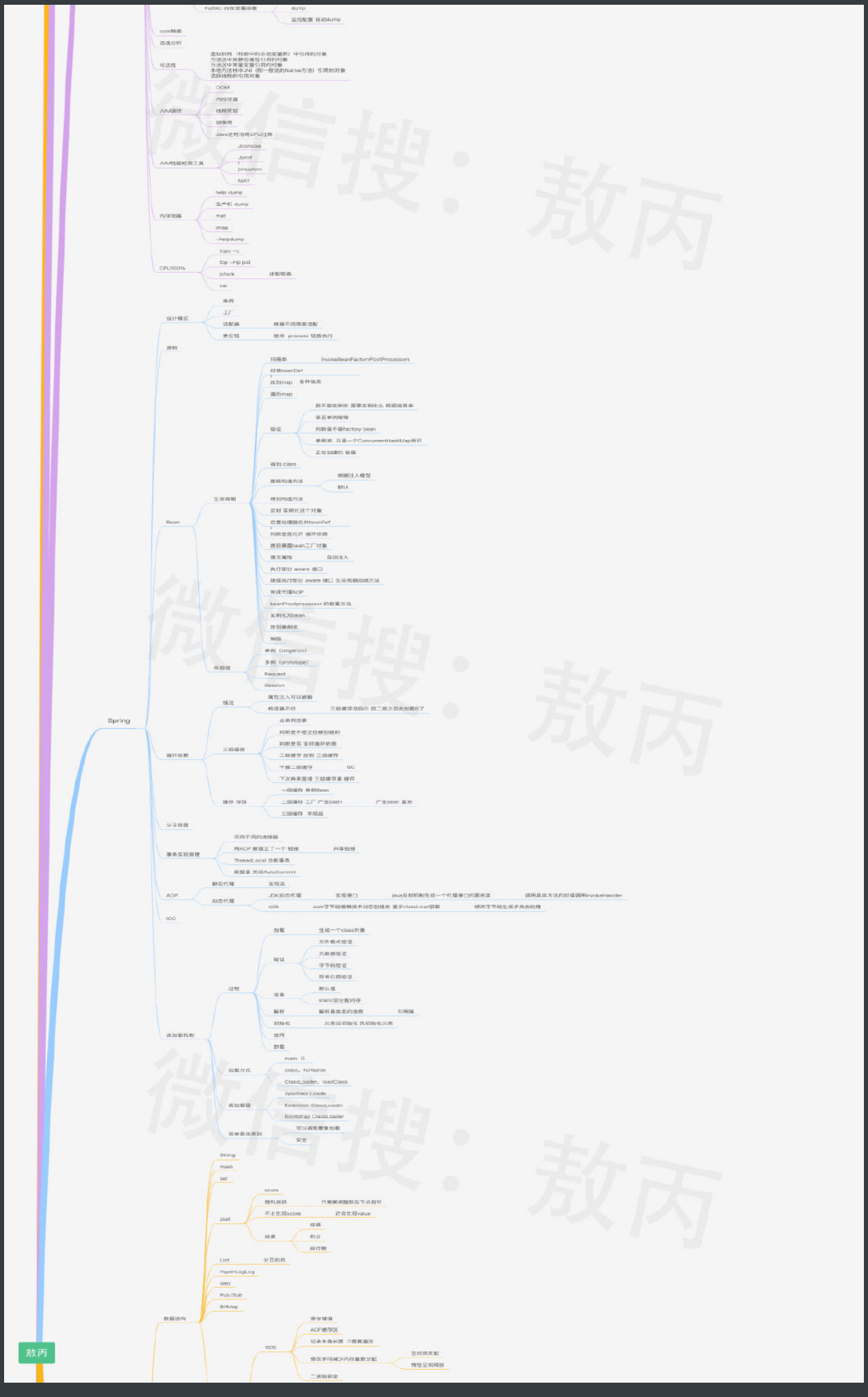

JVM
垃圾回收器
G1
分区概念 弱化分代
标记整理算法 不会产生空间碎片 分配大对象不会提前full gc
可以设置预设停顿时间
充分利用cpu 多核条件下 缩短stw
收集步骤
初始标记 stw 从gc root 开始直接可达的对象
并发标记 gc root 对对象进行可达性分析 找出存活对象
可达性分析算法
最终标记
筛选回收
根据用户期待的gc停顿时间指定回收计划
回收模式
young gc
回收所有的eden s区
复制一些存活对象到old区s区
mixed gc
GC模式
区别
g1分区域 每个区域是有老年代概念的 但是收集器以整个区域为单位收集
g1回收后马上合并空闲内存 cms 在stw的时候做
内存区域设置
XX:G1HeapRegionSize
复制成活对象到一个区域 暂停所有线程
full gc
老年代写满
system。gc
持久代空间不足
STW
实战
性能调优
设置堆的最大最小值 -xms -xmx
调整老年和年轻代的比例
-XX:newSize设置绝对大小
防止年轻代堆收缩：老年代同理
主要看是否存在更多持久对象和临时对象
观察一段时间 看峰值老年代如何 不影响gc就加大年轻代
配置好的机器可以用 并发收集算法
每个线程默认会开启1M的堆栈 存放栈帧 调用参数 局部变量 太大了 500k够了
原则 就是减少gc stw
FullGC 内存泄露排查
jasvism
dump
监控配置 自动dump
oom种类
逃逸分析
可达性
虚拟机栈（栈帧中的本地变量表）中引用的对象
方法区中类静态属性引用的对象
方法区中常量变量引用的对象
本地方法栈中JNI（即一般说的Native方法）引用的对象
活跃线程的引用对象
JVM调优
OOM
内存泄露
线程死锁
锁争用
Java进程消耗CPU过高
JVM性能检测工具
Jconsole
Jprof
jvisualvm
MAT
内存泄露
help dump
生产机 dump
mat
jmap

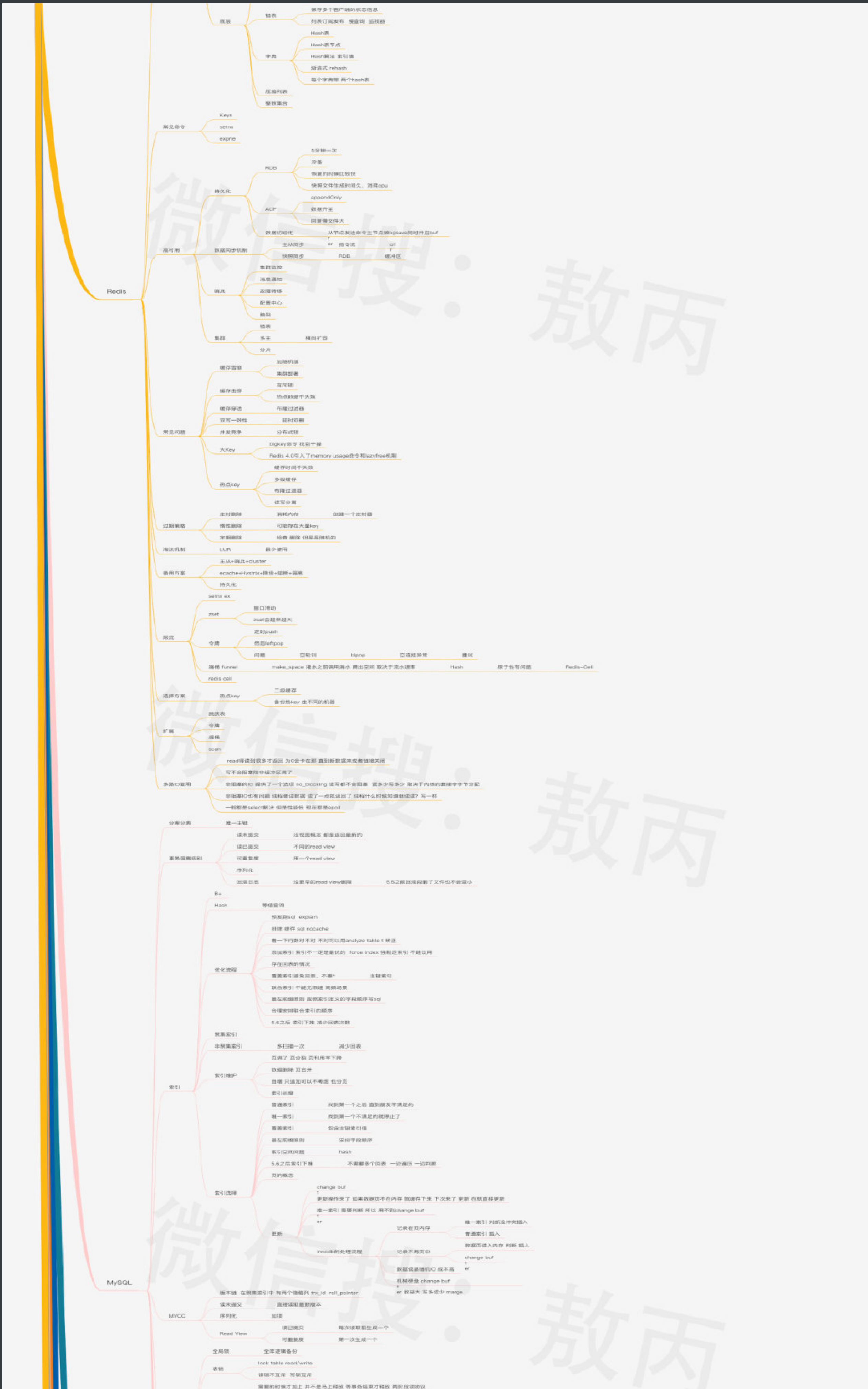

–helpdump
CPU100%
topc –c
top –Hp pid
jstack
进制转换
cat


集合
HashMap
ConcurrentHashMap
ArrayList
LinkedList
基础
UDP
TCP
HTTP
HTTPS
BIO
NIO
select
poll
epoll
synchronized
ACC_SYNCHRONIZED
monitorenter
monitorexit
ThreadLocal
Lock
ReentrantReadWriteLock
ReadLock
WriteLock
ReentrantLock
NonfairSync
FairSync
AbstractQueuedSynchronizer
CAS

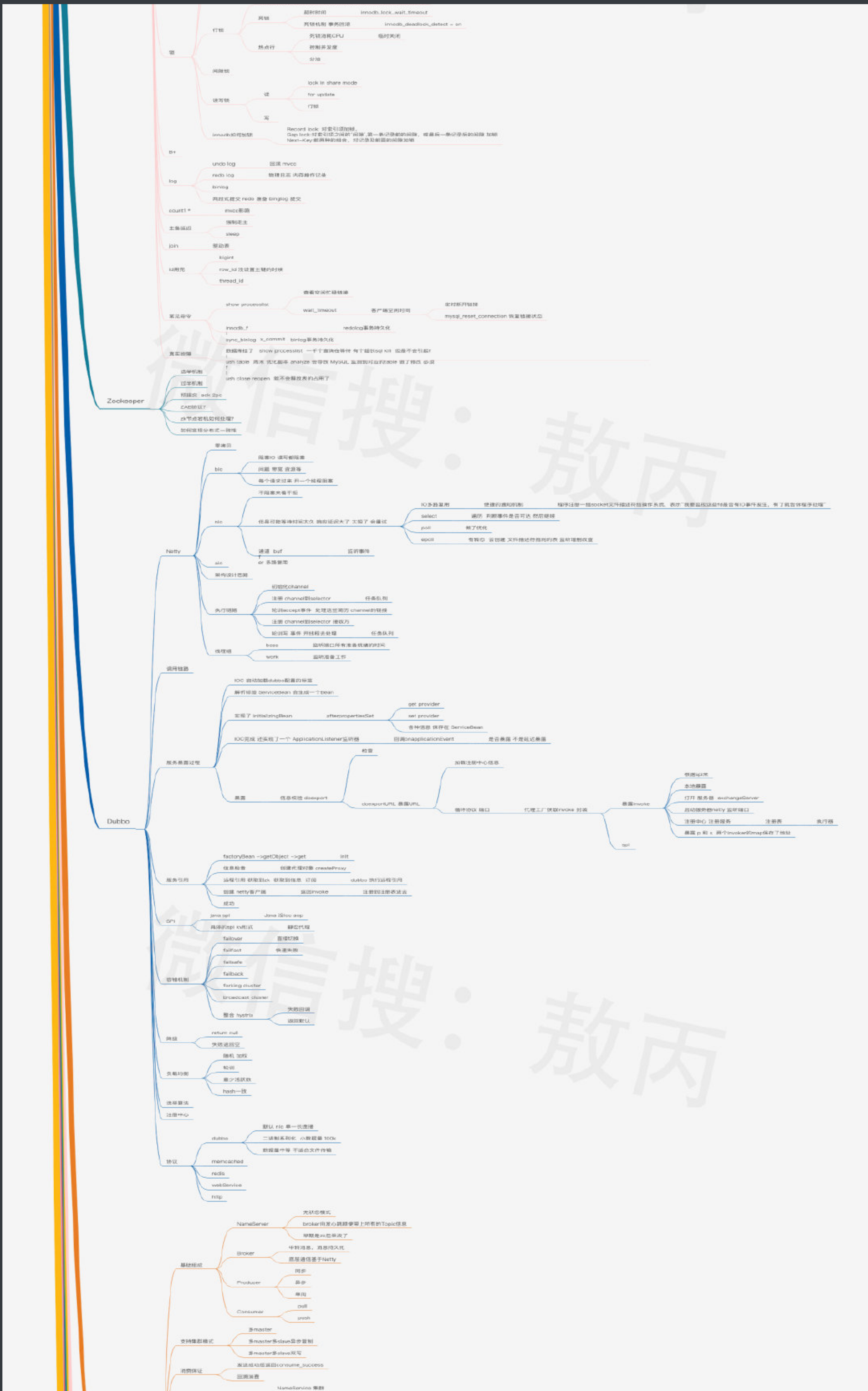


ABA
StampedLock
多线程
volatile
MESI
Java内存模型JMM
高速缓存
可见性
有序性
happens-before
as-if-serial
AtomicInteger
跳出死循环
newFixedThreadPool
newCacheThreadPool
SynchronousQueue
newSingleTheadExecutor
newScheduledThewadPool
DelayedWorkQueue
newWorkStealingPool
ThreadPoolExecutor
核心线程数
最大线程数
缓冲队列
LinkedBlockingQueue
ArrayBlockingQueue
有界队列
入队
er(e, timeout, unit)
Synchronous
参数意义
工厂方法
拒绝策略
线程池
实际使用
执行过程
核心线程->队列->最大线程->拒绝策略
运行状态
running
shutdown
stop
terminated
corePoolSize、maximumPoolSize、largestPoolSize 有意思
故障
JUC
常见问题
线程间是怎么进行通信的?
主要可以介绍一下 wait/notify 机制，共享变量的 synchronized 或者 Lock 同步机制等。
volatile
CountDownLatch
CyclicBarrier
ThreadLocal 用来解决什么问题?
如何尽可能提高多线程并发性能?
读写锁适用于什么场景?
可以回答读写锁适合读并发多，写并发少的场景，另外一个解决这种场景的方法是 copyonwrite。
如何实现一个生产者与消费者模型?
可以尝试通过锁、信号量、线程通信、阻塞队列等不同方式实现。
线程状态
start
sleep终能超时
countDownLatch
只有一个构造方法 只会被赋值一次
没有别的方法可以修改 count
JVM内存模型
堆
栈
方法区
本地方法栈
程序计数器
类加载机制
加载->验证->准备->解析->初始化->使用->卸载
双亲委派原则
父类加载 不重复加载
分代回收
年轻代
Eden/s1/s2
老年代
永久代/元空间
晋升机制
根据存活时间
垃圾回收机制
标记清除
适用场景
对象存活比较多的时候适用
老年代
缺点
提前GC
碎片空间
扫描了两次
标记存活对象
清除没有标记的对象
标记复制
存活对象少 比较高效
适合年轻代
需要空闲空间
需要复制移动对象
引用计数
没办法解决循环引用的问题
标记整理
JVM
垃圾回收器
CMS
分代
年轻
eden
s1
s2
minor gc
通过阈值晋升
老年
major gc 等价于 full gc
永久
缺点
对cpu资源敏感
无法处理浮动垃圾
基于标记清除算法 大量空间碎片
G1
分区概念 弱化分代
可以设置预设停顿时间
收集步骤
最终标记
筛选回收
回收模式
young gc
mixed gc
GC模式
区别
内存区域设置
XX:G1HeapRegionSize
full gc
老年代写满
system。gc
持久代空间不足
STW
实战
性能调优
原则 就是减少gc stw
jasvism

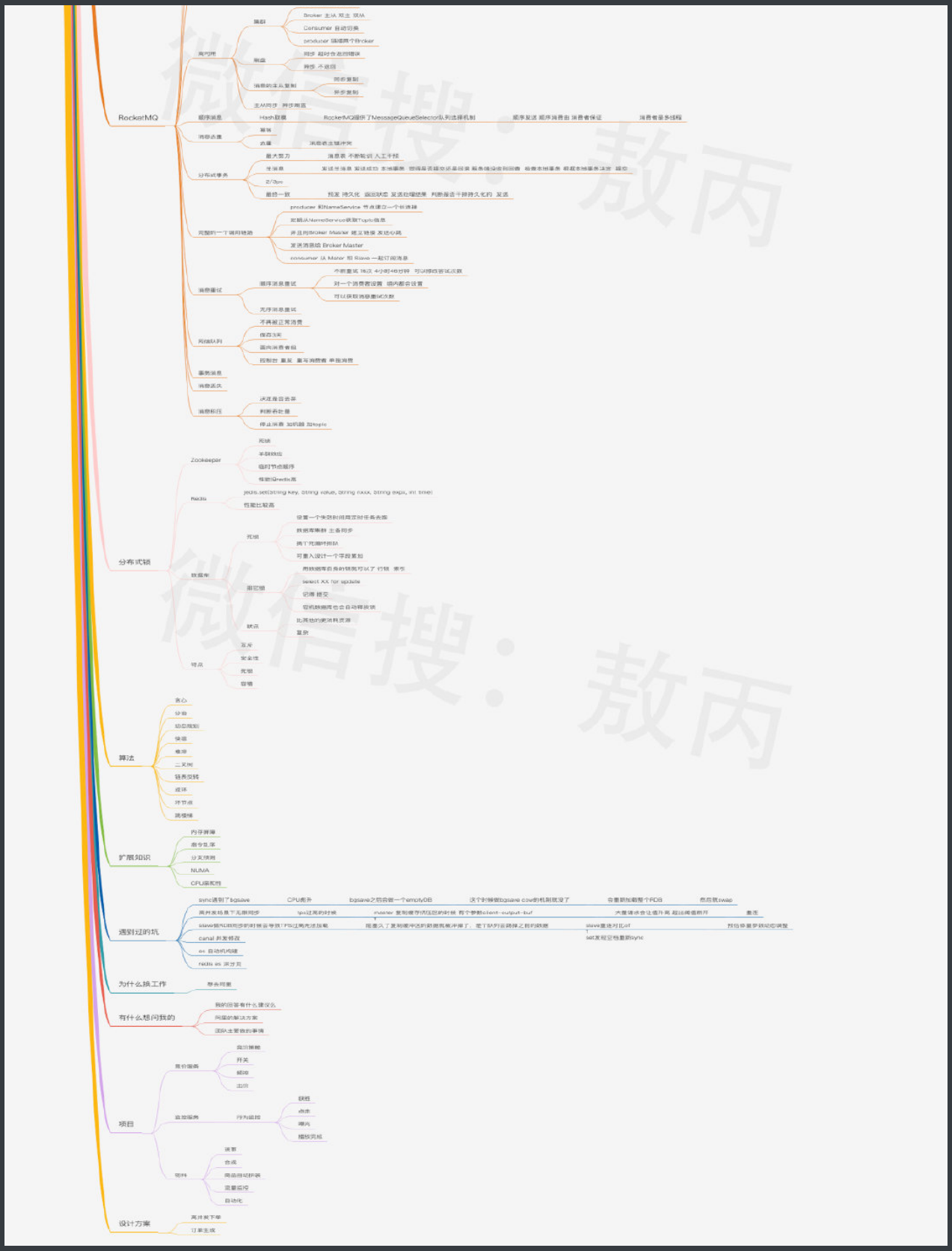

FullGC 内存溢出排查
dump
监控配置 自动dump
oom种类
逃逸分析
可达性
虚拟机栈（栈帧中的本地变量表）中引用的对象
方法区中类静态属性引用的对象
方法区中常量池引用的对象
本地方法栈中JNI（即一般说的Native方法）引用的对象
活跃线程的引用对象
JVM调优
OOM
内存泄露
线程死锁
锁事件
Java进程消耗CPU过高
JVM性能检测工具
Jconsole
Jprof
jvisualvm
MAT
内存溢露
help dump
生产机 dump
mat
jmap
-helpdump
CPU100%
topc -o
top -Hp pid
jstack
读懂堆栈
cat
设计模式
单例
工厂
适配器
根据不同需要适配
责任链
使用 process 链路执行
源码
Spring
Bean
生命周期
扫描类
invokeBeanFactoryPostProcessors
封装beanDef
放到map
各种信息
遍历map
验证
能不能实例化 需要实例化么 根据信息来
是否单例等等
判断是不是factory bean
单例池 只是一个ConcurrentHashMap而已
正在创建的 容器
得到 class
推断构造方法
根据注入模型
默认
得到构造方法
反射 实例化这个对象
放置处理器合并beanDef
判断是否允许 循环依赖
提前暴露bean工厂对象
填充属性
自动注入
执行部分 aware 接口
按照执行部分 aware 接口 生命周期回调方法
完成代理AOP
beanPostprocessor 的前置方法
实例化对象
放到单例池
销毁
作用域
单例（singleton）
多例（prototype）
Request
Session
循环依赖
情况
属性注入可以破解
构造器不行
三级缓存没启示 加二级之后表加载好了
三级缓存
去单例池拿
判断是不是正在被创建的
判断是否 支持循环依赖
二级缓存 放到 三级缓存
十层二级缓存
GC
下次再来直接 三级缓存拿 缓存
缓存 作用
一级缓存 单例Bean
二级缓存 工厂 产生bean
产生bean 半成品
三级缓存 半成品
父子容器
事务实现原理
采用不同的连接器
用AOP 就算上了一个 链接
共享链接
Thread_local 当前事务
前提是 关闭AutoCommit
AOP
静态代理
实现类
动态代理
JDK动态代理
实现接口
Java反射机制生成一个代理接口的匿名类
调用具体方法的时候调用invokeHander
cglib
asm字节码编辑技术动态创建类 基于ClassLoad加载
修改字节码生成子类去处理
IOC
类加载机制
过程
加载
生成一个class对象
验证
文件格式验证
元数据验证
字节码验证
符号引用验证
准备
默认值
static变量配内存
解析
解析具体类的信息
引用等
初始化
父类没初始化 先初始化父类
使用
卸载
加载方式
main 0
class。forName
ClassLoader。loadClass
类加载器
Appdraw Loade
Extention ClassLoader
Bootstrap ClassLoader
双亲委派原则
可以避免重复加载
安全
数据结构
String
Hash
Set
zset
score
跳表原理
只需要调整前后节点指针
不主要按score
还会比较value
场景
视频
积分
排行榜
List
分页的坑
HyperLogLog
Geo
Pub/Sub
BitMap
SDS
保存键值
AOF缓存区
记录本身长度 O需要遍历
修改字符减少内存重新分配
空间预支配
惰性空间释放
二进制安全
敖丙



Redis
底层
链表
保存多个客户端的状态信息
列表订阅发布 慢查询 监视器
字典
Hash表
Hash表节点
Hash算法 索引值
渐进式 rehash
每个字典带 两个hash表
压缩列表
整数集合
常见命令
Keys
setnx
exprie
高可用
持久化
RDB
5分钟一次
冷备
恢复的时候比较快
快照文件生成时间久，消耗cpu
AOF
appendOnly
数据齐全
回复慢文件大
数据初始化
从节点发送命令主节点做bgsave同时开启buf
数据同步机制
主从同步
指令流
快照同步
RDB
缓冲区
哨兵
集群监控
消息通知
故障转移
配置中心
脑裂
集群
链表
多主
横向扩容
分片
常见问题
缓存雪崩
加随机值
集群部署
缓存击穿
互斥锁
热点数据不失效
缓存穿透
布隆过滤器
双写一致性
延时双删
并发竞争
分布式锁
大Key
bigkey命令 找到干掉
Redis 4.0引入了memory usage命令和lazyfree机制
热点key
缓存时间不失效
多级缓存
布隆过滤器
读写分离
过期策略
定时删除
耗内存
创建一个定时器
惰性删除
可能存在大量key
定期删除
检查 删除 但是是随机的
淘汰机制
LRU
最少使用
备用方案
主从+哨兵+cluster
ecache+Hystrix+降级+熔断+隔离
持久化
限流
setnx ex
zset
窗口滑动
zset会越来越大
令牌
定时push
然后leftpop
问题
空轮训
blpop
空连接异常
重试
漏桶 funnel
make_space 灌水之前调用漏水 腾出空间 取决于流水速率
Hash
原子性有问题
Redis-Cell
redis cell
选择方案
热点key
二级缓存
备份热key 走不同的机器
扩展
跳跃表
令牌
漏桶
scan
多路IO复用
read得读到很多才返回 为0会卡在那 直到新数据来或者链接关闭
写不会阻塞除非缓冲区满了
非阻塞的IO 提供了一个选项 no_blocking 读写都不会阻塞 读多少写多少 取决于内核的套接字字节分配
非阻塞IO也有问题 线程要读数据 读了一点就返回了 线程什么时候知道继续读？ 写一样
一般都是select解决 但是性能低 现在都是epoll
MySQL
分库分表
唯一主键
事务隔离级别
读未提交
没视图概念 都是返回最新的
读已提交
不同的read view
可重复读
用一个read view
序列化
回滚日志
没更早的read view删除
5.5之前回滚段删了文件也不会变小
索引
B+
Hash
等值查询
优化流程
预发跑sql explain
排除 缓存 sql nocache
看一下行数对不对 不对可以用analyze table t 矫正
添加索引 索引不一定是最优的 force index 强制走索引 不建议用
存在回表的情况
覆盖索引避免回表，不要*
主键索引
联合索引 不能无限建 高频场景
最左前缀原则 按照索引定义的字段顺序写sql
合理安排联合索引的顺序
5.6之后 索引下推 减少回表次数
聚集索引
非聚集索引
多扫描一次
减少回表
索引维护
页满了 页分裂 页利用率下降
数据删除 页合并
自增 只追加可以不考虑 也分页
索引长度
索引选择
普通索引
找到第一个之后 直到朋友不满足的
唯一索引
找到第一个不满足的就停止了
覆盖索引
包含主键索引值
最左前缀原则
安排字段顺序
索引空间问题
hash
5.6之后索引下推
不需要多个回表 一边遍历 一边判断
页的概念
更新
change buffer
更新操作来了 如果数据页不在内存 就缓存下来 下次来了 更新 在就直接更新
唯一索引 需要判断 所以 用不到change buffer
innodb的处理流程
记录在页内存
唯一索引 判断没冲突插入
普通索引 插入
记录不再页中
数据页读入内存 判断 插入
change buffer
数据读是随机IO 成本高
机械硬盘 change buffer 收益大 写多读少 marge
MVCC
版本链 在聚集索引中 有两个隐藏列 trx_id roll_pointer
读未提交
直接读取最新版本
序列化
加锁
Read View
读已提交
每次读取前生成一个
可重复读
第一次生成一个
全局锁
全库逻辑备份
表锁
lock table read/write
读锁不互斥 写锁互斥
需要的时候才加上 并不是马上释放 等事务结束才释放 两阶段协议



死锁
超时时间
innodb_lock_wait_timeout
死锁机制 事务回滚
innodb_deadlock_detect = on
行锁
死锁消耗CPU
临时关闭
热点行
控制并发度
分治
锁
间隙锁
lock in share mode
for update
行锁
读写锁
读
写
innodb如何加锁
Record lock: 对索引项加锁。
Gap lock:对索引项之间的'间隙',第一条记录前的间隙, 或最后一条记录后的间隙 加锁
Next-Key:前两种的组合, 对记录及前面的间隙加锁
B+
undo log
回滚 mvcc
redo log
物理日志 内存操作记录
log
binlog
两段式提交 redo 准备 binglog 提交
count1 *
mvcc影响
主备延迟
强制走主
sleep
join
驱动表
bigint
id用完
row_id 没设置主键的时候
thread_id
show processlist
查看空闲忙碌链接
wait_timeout
客户端空闲时间
定时断开链接
mysql_reset_connection 恢复链接状态
常见命令
innodb_f
redolog事务持久化
sync_binlog
x_commit
binlog事务持久化
真实故障
数据库挂了
show processlist
Zookeeper
选举机制
过半机制
预提交
ack 2pc
CAB协议?
zk节点宕机如何处理?
如何实现分布式一致性
零拷贝
阻塞IO 读写都阻塞
bio
问题 带宽 资源等
每个请求过来 开一个线程阻塞
不阻塞来者不拒
IO多路复用
select
nio
poll
做了优化
epoll
通道
buf
监听事件
Netty
aio
架构设计思路
初始化channel
注册 channel到selector
任务队列
执行链路
轮训accept事件
注册 channel到selector 接收方
任务队列
线程组
boss
work
监听准备工作
调用链路
IOC 自动加载dubbo配置的标签
解析标签 ServiceBean 会生成一个bean
实现了 InitializingBean
afterpropertiesSet
get provider
set provider
各种信息 保存在 ServiceBean
IOC完成 还实现了一个 ApplicationListener监听器
回调onapplicationEvent
是否暴露 不是延迟暴露
服务暴露过程
检查
加载注册中心信息
暴露
信息校验 doexport
doexportURL 暴露URL
代理工厂 获取invoke 对象
暴露invoke
创建spi类
本地暴露
打开 服务器
exchangeServer
启动服务器netty 监听端口
注册中心 注册服务
注册表
执行器
spi
Dubbo
factoryBean ->getObject ->get
init
信息检查
创建代理对象 createProxy
服务引用
dubbo 执行远程引用
创建 netty客户端
返回invoke
注册到注册表进去
成功
java spi
SPI
failover
直接切换
failfast
快速失败
failsafe
failback
容错机制
forking cluster
broadcast cluster
整合 hystrix
失败回调
返回默认
return null
降级
失败返回空
随机 加权
轮训
负载均衡
最少活跃数
hash一致
选举算法
注册中心
dubbo
默认 nio 单一长连接
二进制系列化 小数据量 100k
数据量中等 不适合文件传输
memcached
协议
redis
webService
http
NameServer
无状态模式
broker向发心跳顺便带上所有的Topic信息
Broker
中转消息, 消息持久化
底层通信基于Netty
基础组成
同步
Producer
异步
单向
Consumer
pull
push
多master
支持集群模式
多master多slave异步复制
多master多slave双写
发送成功后返回consume_success
消费保证
回滚消费
NameService 集群



pdf网盘链接：链接：https://pan.baidu.com/s/1ySw0c5BqRuTVQP4ws5-zBg 密码:rdvd

我是敖丙，一个在互联网苟且偷生的程序员，大家周末愉快