Java整体知识架构详解-之进阶篇二

- JVM细节
  - 类何时被回收
    - 1. 该类在堆中不存在实例对象
    - 2. 该类的ClassLoader已被回收
    - 3. 对该类的对象没有任何引用
  - 方法中的对象何时被回收
    - 当方法执行完成后，方法中定义的对象，需要等待垃圾回收才销毁
  - 软引用和弱引用的区别
    - 软引用在内存不够时才回收，内存足够不会回收；而弱引用，只能撑到下次垃圾回收之前就会被回收
  - 新生代到老年代的条件
    - 我们的一般印象是新生代GC到一定年龄后，对象才会进入老年代，但实际并不是只有这一种情况
    - 当survivor区中相同年龄的对象之和超过survivor空间的一半，年龄大于或等于该年龄的对象就可以直接进入老年代，无需等到要求的最大年龄
  - JVM在什么情况下会加载一个类
    - 简单的说就是在代码中用到这个类的时候
  - JVM类加载过程
    - 1. 加载
      - 读取类的.class文件
    - 2. 验证
      - 需要检验这个.class文件是否服务规范，也就是能不能运行，防止被篡改，不能运行我还干嘛继续呢
    - 3. 准备
      - 根据类中的属性等，分配内存空间，并赋予默认值
    - 4. 解析
      - 将符号引用替换为直接引用
    - 5. 使用
    - 6. 卸载
  - JVM深入
    - 我们知道JVM按区域划分可分为：堆，虚拟机栈，方法区，程序计数器，本地方法栈，这次更深入的描述下各组成部分
    - 程序计数器
      - 或许有些人会直接解释为程序运行指针的位置信息，更具体的来说是**每个线程当前执行的字节码指令的位置，所以每个线程都有自己的程序计数器，属线程私有的**
    - 虚拟机栈

- 用来存放局部变量，操作数栈，动态链接，方法出口等，更具体来讲，是因为线程每执行一个方法会创建一个栈帧压入栈，而栈帧中有前面说的这些东西，属线程私有的
  - 什么是动态链接
- 堆
  - 存放创建对象，线程共享
- 方法区
  - 存放类的相关信息，常量，静态变量等，线程共享
    - jdk8以后用元空间（MetaSpace）取代，存储在物理内存（本地内存）
- 本地方法栈
  - 和虚拟机栈类似，不过调用的是本地操作系统里面的底层类库，线程共享

- SpringCloud原理
  - Eureka
    - Eureka Client
      - 负责将服务的信息注册到Eureka Server中
    - Eureka Server
      - 注册中心，服务维护注册表，保存了各个服务器所在的机器和端口号
  - Feign
    - feign是用来代替httpClient，通过服务名的方式，动态代理接口接口，发起请求响应
    - 代码例子
      - 图片
    - 缺点
      - 通过Feign的方式调用服务，就像上面的例子一样，需要设置调用服务的名称在@FeignClient中，这样也导致了，当服务B部署了B1，B2，B3这三个不同名但功能相同服务，那么我就要写三个类似的FeignClinet客户端代码，之后如果再加个B4，又要改代码；或者Feign还提供了动态创建实例的功能（见链接），但存在问题，服务降级逻辑不知道写哪了
  - Ribbon
    - 作用
      - 作为负载均衡，比如一个服务部署在5台机器上，该组件的作用可以把请求均匀发送到各台机器
      - 默认算法为Round Robin轮询算法
  - Hystrix
    - 作用
      - 隔离、熔断和降级
        - 流程图
  - zuul
    - 作用
      - 微服务网关，根据请求的特征进行不同服务的转发，还可以做**统一的降级、限流、认证授权、安全**
- 优秀第三方组件
  - easyexcel

- 概述
  - 快速、简单避免OOM的java处理Excel工具
- github

- EasyMock
  - 概述
    - 好用的第三方数据模拟工具，在后台开发没完成时，可以用它来模拟数据

- MyBatis-Plus
  - 概述
    - 此为为了更好更容易进行mybatis编程而设计的架构，囊括了从代码生成到代码简化，到各种插件的运用等
  - 功能概述
    - 优点
      - 通过集成pom，继承它提供的类可以做到大部分sql不需要再手动写入
      - 简化sql编写，抽象公共方法属性，使代码编写更简单
      - 支持jdk8的lambda表达式，语法编写更简洁明快
      - 支持主键自动生成策略
      - 支持分页
    - 缺点
      - 有一定学习成本（也不能叫缺点，复杂点的框架都有学习成本）
  - 插件
    - 分页
      - 通过添加分页插件，可以直接使用继承而来的分页方法进行分页查询
    - 逻辑删除
      - 添加该插件后，可以使删除sql的方法如`deleteByxxx`变成逻辑删除
    - 性能分析
      - 用于输出每条sql语句内容和执行时间
    - 乐观锁
      - 用于方便的实现乐观锁
    - ...

- 布谷鸟过滤器
  - 布隆过滤器缺点
    - 查询性能低
    - 空间利用率低
    - 不支持反向操作（删除）
    - 不支持计数
  - 布谷鸟过滤器缺点
    - 不能重复插入相同的数据，否则空间会严重浪费，还不如布隆过滤器
    - 删除会造成高概率误删，所以也基本不可用
    - 空间必须是2的指数倍，这样也导致了空间上可能也没有优势
  - 说明

- 结论，知道有这玩意就好，可以看下前面的连接文章具体介绍，用就不建议用了
- Java零碎知识点
  - synchronize 和 lock 的区别
    - synchronize是JVM内置关键字，lock是Java类
    - synchroinze锁可重入，不可中断，不公平，Lock可重入，可中断，可公平
    - synchronize可自动释放，Lock需要手动释放
    - Lock提供了Condition可控制notify哪个线程，synchroinze不能针对唤醒
    - 总结：一般可以使用synchronize的地方，用它更方便；Lock使用更加灵活，在需要灵活中断获取锁，唤醒特定线程等场合可以用Lock
  - volatile 关键字的作用
    - 保持内存可见性
      - 所有线程都能看到共享内存的最新状态
    - 防止指令重排序
      - 在单例模式中，用右侧的代码创建单例，仍可能造成问题，获得的是未初始化的对象，原因就是指令重排序，先分配了对象的内存地址，再初始化对象，还没初始化对象前，对象可能已经不为null了，这种情况就可以用volatile解决（建议看下前面的链接）
  - 二叉树
    - 深度优先遍历
      - 前序遍历
      - 中序遍历
      - 后序遍历
    - 广度优先遍历
  - 多线程
    - 如何保证线程顺序
      - 最基础的join
      - 使用锁等待机制，获取同一把锁，设置标志位，后续的await
      - CountDownLatch计数器，当设置多个时，进行await，类似于设置了标志位
  - java对象的内存占用
    - 前提说明
      - 初级程序员常会被问到一个问题，java的8种数据类型占用多少字节，这个看起来很好回答，但实际使用中，我们定义的对象是否是该大小？事实上我们在实际开发中，如int我们更多的是用Integer，double用Double，那么Integer或Double又占用多少内存？
    - 解释
      - java对象存在请求头，类型指针在64位JVM中各占64位，也就是16字节，经过JVM压缩指针可以降到12字节；同时JVM内存需要遵循内存对齐，也就是对象的起始地址需要对齐至8的倍数；如果一个字段占据C个字节，那么该字段的偏移量需要对齐至NC，这样就解释了为什么Long，Double类型对象要占据24字节，因为尽管压缩后对象头大小为12字节，但Long，Double为8字节，需要对齐至8的倍数，也就是最少16字节，加上本身的8字节为24字节；同理Integer类型对象头12字节，本身4字节，满足对齐条件，加上本身4字节，总共占据16字节
      - 上述内容我根据链接的博客推导而来，若需要验证可以运行右方代码查看对象占用字节

- 秒杀系统设计
  - 拦截
    - 拦截掉无效请求，比如秒杀抢货物，一般只会前几百到几万的有效订单，达到数量了拦截掉剩余的无效请求
  - redis暂替数据库
    - 若剩下还有数万的请求，接收请求的tomcat可以集群部署几百个，但数据库承受不了每秒数万，此时可以用redis代替，之后再写入数据库
  - 限流削峰
    - 在抢完订单后，可能需要把信息发给发货系统，一秒数万的信息发货系统也需要数百台tomcat来接，但是没有必要，发货不需要实时，这时可以用消息中间件，慢慢消费发送，达到限流削峰的作用
- 解决服务器资源耗尽原因
  - 1. 首先当然是top找到cpu占用最高的pid
  - 2. 查看具体cpu
  - 3. 找到耗时最长的线程TID
  - 4. 打印堆栈信息
- 开启查看慢sql日志

- 分布式锁
  - 作用
    - 保证分布式环境下的顺序执行，分布式环境下的锁机制
  - 常用锁类型
    - Redis
      - 问题
        - 加锁过程
          - 1. redis加锁后如果系统宕机，会导致锁不能够释放，所以为了解决这个问题，必须设置过期时间
          - 2. 但过期时间同样不好设置，假设过期时间30秒，但在30秒内服务没有运行完成，则会释放锁，导致其他系统获取锁
          - 3. 而设置过期时间也不能用两个命令，比如先设值再设过期时间，这样也可能导致设值后宕机没成功设置过期时间
        - redis的部署方式
          - 1. 单机部署容易导致宕机后不能获取锁
          - 2. 主从模式，当主宕机后主从切换过程也会在这短暂的时间不能正确获取锁
      - 建议
        - 建议使用Redission，这是个第三方的开源框架
        - Redission
          - 代码例子
          - redlock算法支持
          - 特性
            - 1. 所有指令采用lua脚本，保证原子性

- 2. 锁超时时间怎么办？它提供了个watchdog，会在获取锁以后每隔10秒把key设置过期时间为30秒，这样只要程序一直在运行就不会过期，也不会死锁
- zookeeper
  - zookeeper的基本特性
    - zk的节点可以是临时节点也可以是永久节点
    - zk的节点可以是有序节点，按序递增
    - zk的节点有事件监听机制，节点的创建、修改、删除、子节点变更都会触发事件监听
  - 获取锁的流程
    - 看图
  - 开源方案
    - Curator
      - 使用代码
  - 评价
    - zk获取锁的过程耗时一般比Redis久点，但安全性比Redis高
- 优缺点
  - redis
    - 缺点
      - 获取锁的方式只能通过不断轮询尝试获取，比较消耗性能
      - 不够健壮，Redis可能发生不安全的情况也就出现在主服务器获取到锁还没同步到从服务器，主服务器宕机，从服务器切换到主服务器的这个短暂的过程中
    - 优点
      - redis分布式锁是比较常用的，它的性能比较高，获取锁的速度快，所以并发效率高
      - 大部分情况下都不会遇到极端复杂场景导致锁获取错误
  - zookeeper
    - 优点
      - 定位就是分布式协调，强一致性，所以锁的模型比较健壮
      - 如果获取不到锁，主需要添加一个监听器，不需要一直轮询，性能消耗较小
    - 缺点
      - 如果有较多的客户端频繁申请加锁，释放锁，对zk集群的压力会比较大，获取锁的速度相对redis会稍慢

- 设计模式深入
  - 观察者模式
    - 观察者模式和发布订阅模式一样吗
      - 这可以是一个面试题，表面上看观察者模式是被观察者通知观察者消息，发布订阅模式生产者通知消费者消息，好像就名称不一样
      - 区别
        - **观察者模式是高耦合的关系**，被观察者触发事件后通知所有观察者，被观察者是知道有哪些观察者的，只是可以自由添加删除，所以是松耦合； **发布订阅模式则是完全解耦**，它还有个组成部分broker，也就像MQ中间件中的topic存储，生产者发送消息到topic，它并不知道有哪些消费者，消费者可以自由订阅哪个生产者的消息

- 使用层面讲，观察者模式多用于应用内部，对象间的通知机制；而发布订阅模式，则是中跨应用的模式，也就是常用的消息中间件

- 线程池深入
  - 如何优雅关闭线程池
    - 说明
      - 这个网上有太多的错误或表达不明确的答案，然后互相抄袭，都是笼统的介绍了下，还都是错误的；我这里会概括几句，能理解最好，不能理解可以看下前面的链接文章，可**不是简单的shutdown优雅关闭，shutdownNow直接关闭**
    - 线程中断
      - 这个需要着重说明，应该线程池中断也是要基于这个原理
      - 线程中断并不是调用Thread.interrupt，这里只是置了个标志位，具体的中断逻辑需要在自己的线程，用isinterrupt判断；
      - 这里有两种情况的中断逻辑需要判断，当interrupt标志置为true时，如果线程处于阻塞状态会抛InterruptedException异常；如果正在运行则需要自己线程代码中判断isinterrupt，这里我给出个我写的demo代码
    - 线程池关闭
      - shutdown
        - 一般情况下，我们使用shutdown是没什么问题的，当如果你存有永久循环处理线程，那就要注意了，shutdown是关不了这种永久while线程的
        - 它的解释说明是：线程池拒接收新提交的任务，同时等待线程池里的任务执行完毕后关闭线程池
      - shutdownNow
        - 它的解释是：线程池拒接收新提交的任务，同时立马关闭线程池，线程池里的任务不再执行
        - 实际执行过程是，当执行shutdownNow方法，它会取出剩下所有没有执行的任务队列并返回，而线程池是在不断循环任务队列的；此时又有两种情况，线程池正取到的任务阻塞，那么会直接抛出异常，和线程中断抛异常一样的逻辑处理；如果线程在执行状态，则会处理完当前任务然后继续循环，判断到任务为null了退出
        - 上面的我继续概括：执行该方法后，会删除所有未执行任务，正在执行的任务如果是阻塞的，则抛异常，否则置中断状态后，继续执行完成当前任务后退出；
      - 相同
        - shutdown和shutdownNow都是异步执行的
      - 不同
        - shutdownNow会强制置所有工作线程置中断状态，中断导致的后果需要通过自己代码判断isinterrupt处理；而shutdown需要trylock尝试获取锁，获取不到则不会中断
      - 结论
        - 当使用shutdwonNow需要捕获处理可能抛出的中断异常；当使用shutdown需要保证线程任务没有永久阻塞的逻辑，否则线程关闭不了

- 算法深入
  - 倒排索引

- 说明
  - 说起倒排索引，就能想到ElasticSearch，它通常用在搜索引擎用来快速查找
- 为什么叫倒排索引
  - 它是不是由记录来确定属性值，而是由属性值来确定记录
- 组成
  - 通常有单词词典和倒排文件组成
- 单词词典
  - 常用数据结构包含哈希加链表和树形词典结构
- 倒排文件
  - 存储在磁盘的文件，存储倒排索引
- 压缩算法
  - 为减小索引文件，使用了压缩技术，比如对数字压缩只保存与上一个值的差值，以减小数字的长度
- 一句话概括
  - 我们要查找n篇文章中，某个单词出现在哪篇文章的哪个位置，这时我们发现这n多的文章出现的单词都是有很多重复的，我们把所有单词都整理出来排序后作为词典，记录每个单词出现在哪篇文章的文章编号，出现位置，出现次数等，这就形成了倒排索引

- 递归
  - 说明
    - 递归一直是很多学算法的人难以灵活运用的算法，这里给出一篇我觉得很好的文章，希望对你有用

- SQL优化深入
  - 优化语句如何查看?
    - 大多数人知道EXPLAIN用来分析sql的索引命中情况，事实上我们可以进一步查看mysql优化过后的语句是什么样的，看右边图标
  - 何时建联合索引?
    - 根据的是区分度，当字段A=x能搜出数万条记录，字段B=y能搜出数万条记录，而字段A=x，B=y同时作为条件能搜出**数百条记录**，说明这个区分度是比较明显的，此时就可以把AB字段作为联合索引；若是AB同时作为条件仍然搜出了**数万条记录**，说明区分度不明显，就不必要建联合索引了

- 进阶面试题
  - wait和notify为什么在Object类，而不是Thread中
    - 在java中线程是不能被指定的，也就是说，在线程A你不能够对线程B进行入侵操作，而所有对象都持有锁，可以用来作为监视器，线程A可以通过监视器进入阻塞，也可以通过notify来唤醒其它线程，这样的好处是监视器是可以指定的，也没有入侵性