# Java整体知识架构详解-之基础知识

- 锁
  - 死锁
    - 多线程会遇到的问题，两个线程同时争抢同一把锁造成死循环
  - 悲观锁
    - 悲观锁认为同一个数据并发操作一定会发生修改，所以要对一个数据的并发操作加锁，悲观锁是比较重量级的
  - 乐观锁
    - 乐观锁认为同一个数据的并发操作是不会发生修改的，可以通过版本机制，当更新数据的时候会进行不断尝试修改
  - 自旋锁
    - 自旋锁是指尝试获取锁的线程不会立即阻塞，二是会采用循环的方式不断尝试获取锁；好处就是减少了线程上下文的切换，缺点是消耗CPU，实际使用还是要根据实际情况
  - 可重入锁
    - 可重入锁指在同一个线程外层方法获取锁后，内层方法加同一个锁会自动获取锁
  - 偏向锁
    - 偏向锁指的锁的一种状态，当一段同步代码一直被同一个线程所访问的时候，那么线程会自动获取锁，减少获取锁的消耗
  - 轻量级锁
    - 当偏向锁被另一个线程所访问时，偏向锁会升级为轻量级锁，也就是自旋锁的方式，不断尝试获取，期待另一个线程马上释放锁
  - 重量级锁
    - 当轻量级锁一直自旋下去到一定时候还没有获取到锁，锁会升级为重量级锁，进入阻塞状态
  - 分段锁
    - 这是ConcurrentHashMap在jdk7中用到机制，将数组划分更细颗粒度进行锁控制
  - cas
    - cas有三个字母分别表示v：想要更新的值，e：预期值，n：新值，只有当v=e的时候，才会更新值为n
  - 分布式锁
    - 这属于分布式架构的讨论，移步分布式架构->分布式架构解决方案->分布式锁
- Java虚拟机
  - 什么是Java虚拟机
    - 一个可以执行Java字节码文件的虚拟机进程
  - 内存结构
    - 堆
      - 存放java对象的地方，线程公有

- 栈
  - 存放临时的局部变量，为线程私有
- 方法区
  - 存放静态变量，类的加载信息等，线程公有
- 本地方法栈
  - 存放本地方法的相关信息
- 程序计数器
  - 存放程序运行位置记录

- 垃圾收集器GC
  - 概述
    - 负责清除没有引用的失效对象的java后台线程
  - 相关概念
    - 并发（Concurrent）
      - 指用户线程与垃圾收集线程根据cpu轮询机制交替运行，使得看上去像同时运行
    - 并行（Parallel）
      - 充分利用CPU资源，多个任务在多个CPU上各自运行，一个任务会占用一个CPU资源
  - 收集器类型
    - 新生代收集器
      - Serial收集器
        - 串行收集器，采用单线程模式，复制算法，进行垃圾收集时会停止其他工作线程（Stop The World）
      - ParNew收集器
        - Serial收集器多线程版，和Serial有相同问题
      - Parallel Scavenge收集器
        - 并行收集器，通过控制吞吐量减少停顿时间
    - 老年代收集器
      - Serial Old收集器
        - 单线程收集器，使用标记整理算法，存在停顿时间长问题
      - Parallel Old收集器
        - 多线程模式，使用标记整理算法
      - CMS收集器
        - 概述
          - 以获取最短回收停顿时间为目标的收集器，基于标记清除算法实现
        - 工作过程
          - 初始标记
            - 标记下GC Roots能直接关联到的对象，耗时很短，会Stop The World
          - 并发标记
            - GC Roots Tracing过程，最耗时，但是并发操作，工作线程不会停止
          - 重新标记

- 修正并发标记期间因程序继续运行导致的标记变动，耗时相对很短，会Stop The World
- 并发清除
  - 清除标记对象，也是并发操作，耗时相对较长，工作线程不会停止
- 优点
  - 并发收集，低停顿
- 缺点
  - 对CPU资源非常敏感，CMS默认启动回收线程数（CPU数量+3）/4，在CPU本来就少的情况下，会占用很多的CPU资源，导致用户线程变慢
  - 无法清除浮动垃圾，也就是在并发清理阶段，用户线程还在运行，这时产生的垃圾，本次清理过程是不会处理掉的
  - 产生空间碎片，CMS采用了标记清除算法，虽然更快，但也会产生大量空间碎片
- G1收集器
  - 特点
    - 并行和并发，充分利用多CPU的优势缩短停顿时间
    - 空间整合，整体采用标记整理算法，局部看采用复制算法，更加合理利用空间
    - 可预测的停顿，使用者可以明确规定垃圾处理器GC消耗不得超过N毫秒
  - 内存划分
    - G1收集器将整个Java堆划分成大小相等的独立区域（Region），虽然保留了新生代老年代的概念，但不再是物理隔离，代表了一部分Region的集合

- gc类型
  - Minor GC
    - 年轻代满是触发，具体值Eden代满
  - Full GC
    - 当老年代满时触发，会回收年轻代和老年代

- 垃圾收集算法
  - 引用计数法
    - 概念
      - 用来表示对象存在一个引用，计数器加1，到计数器为0时表示对象没有引用则可以回收
    - 问题
      - 该算法存在循环引用的问题，比如A引用B，B引用C，C引用D，D引用B，当A不在引用B时，其实BCD对象都没存在必要了，但他们循环引用导致引用计数器不会清零，这部分对象就清理不掉，所以现在基本不用该算法了
  - 标记清除法
    - 概述
      - 对GC Roots可达的对象进行标记，再统一清理没被标记的对象
    - 缺点
      - CMS垃圾收集器就采用了该算法，缺点是会产生大量的内存碎片，会导致后续大对象存储更容易失败

- 标记整理法
  - 概述
    - 和标记清除类似，但它对内存碎片进行了整理，将存活对象都移动到内存一端，G1垃圾收集器就使用了该算法
- 复制算法
  - 概述
    - 复制算法在新生代有广泛应用，复制算法需要两块相同大小的内存AB，当A内存中存活对象标记完毕，会统一把存活对象复制到B内存块，然后清空整个A内存，如此往复；新生代survivor的to和from就是代表
- 分代算法
  - 这是一个笼统的概称，它将内存根据生命周期划分成年轻代老年代，然后不同的区域使用不同的回收算法进行垃圾回收

- 内存屏障
  - 概述
    - 每个CPU都有自己的缓存，在多线程环境下工作内存和主内存不能实时进行信息交换，为了防止指令的乱序和数据错误，有了内存屏障，不同平台实现手段不同，JVM屏蔽了这些差异，统一由JVM来生成内存屏障指令
  - 作用
    - 防止屏障两侧的指令重排序
    - 强制把工作内存中的数据写会主内存，使得信息保持一致

- 算法和基础概念
  - 拜占庭将军问题
    - 概述
      - 在假设通信不会被中断且消息可靠的前提下，需要找到一种分布式协议，协调多支军队选择共同进攻还是撤退
    - 两军问题
      - 由拜占庭将军引申出来的问题，通信兵A和通信兵B互相通信问题，A发消息通知B后，B回复消息收到了，此时B要确认A收到了我的回复消息，所以A要再发个消息给B表示我收到了你的回复；但此时A又不确定B是否真的收到了，B还得发确认消息给A，这样会导致消息的不可靠性
      - 由拜占庭将军引申出来的问题，通信兵A和通信兵B互相通信问题，A发消息通知B后，B回复消息收到了，此时B要确认A收到了我的回复消息，所以A要再发个消息给B表示我收到了你的回复；但此时A又不确定B是否真的收到了，B还得发确认消息给A，这样会导致消息的不可靠性
    - TCP协议
      - 由上述两军问题可以看出，TCP协议就是为了解决这种消息不可靠性产生的；用白话对应就是A发送随机数x给B，B收到后把x+1，在生产随机数y，把两数都发给A,A收到后确认了B收到了消息，在把y+1返回给B，这样B也确认A收到了消息，这就是TCP协议重点。但它任然不能绝对保证消息可靠性，因为传输途中不能保证是否被拦截修改内容
      - 由TCP协议引申出来还有三次握手和四次挥手

- gossip协议
  - 概述
    - 由子节点发起状态更新，随机朝周围子节点散播消息，收到消息的子节点也会重复该过程，直到所有节点都收到消息；这过程会有一定时间延迟，理论上节点越多延迟越高，这是一个最终一致性协议
  - 优点
    - 扩展性好，允许网络任意增加减少节点
    - 容错性好，网络中任何节点的宕机和重启不会影响gossip消息传播
    - 去中心化，避免了把压力都堆积到中心节点
    - 一致性收敛，消息传播速度快，消息是以指数形式传播，以一传十十传百这样，传播速度达到logN
    - 协议实现较为简单，复杂性低
  - 缺点
    - 消息有一定延迟，不使用在实时性要求极高的场合
    - 消息冗余，根据gossip协议，一个节点向周围传播消息可能会使一个节点收到多个节点的消息传播
- Raft算法
  - 概述
    - 共识算法，对拜占庭将军问题简化，假设没有叛军，但信息可能传递不到，有Follower，Candidate，Leader三个角色，通过一个随机时间倒计时的选举模式进行选主，获得票数过半选举成功

- Java基本数据类型
  - byte
    - 1字节
  - boolean
    - 1字节
  - short
    - 2字节
  - char
    - 2字节
  - int
    - 4字节
  - float
    - 4字节
  - long
    - 8字节
  - double
    - 8字节
- 常用Set
  - HashSet

- 不要以为是新的数据结构，**该方法的内部实现原理实际用到的是HashMap**
- 常用List
  - ArrayList
    - 数组实现，擅长查询
  - LinkedList
    - 链表实现，擅长增删改
  - CopyOnWriteArrayList
    - 并发情况下，读场景远远大于写场景的时候使用，它的写场景会复制一份相同的数组去这备份数组去写，读写互不干扰，所以读更快，且会更耗内存
- 常用Map
  - HashMap
    - 线程不安全的，在不用考虑线程安全的情况下推荐使用
    - 结构
      - 数组加链表的结构
    - 扩容
      - 参数中有个默认的负载因子0.75，表示数组占用超过这个比值就会进行扩容变为原来2倍
    - hash
      - hashmap的key值存放在数组的位置是根据key的hash值的高16位和低16位进行异或运算
    - transfer
      - 重哈希，在数组扩容后数据重新hash计算进行数据复制移动的过程
    - 为什么数组长度总为2的n次方
      - 优化取模速度，当长度为2的n次方时，h&(length-1)等于h对length取模，而与运算会比直接取模块
      - 不同的hash值发生碰撞的概率会比较小，尽量使数据在table中均匀分布
  - ConcurrentHashMap
    - 线程安全且高效的，jdk1.7使用segment分段锁机制，jdk1.8时改用cas无锁机制，锁的颗粒度更细了
  - HashTable
    - 线程安全但效率不高，基本没人用的，全程sychronized同步锁
- Java多态
  - Overload重载
    - 同一个类中方法名相同，参数列表不同，体现了一个类中方法的多态
  - Override重写
    - 子类可以重写父类的方法并进行覆盖，体现了对象实现方式的多态
- 源码分析
  - 常用设计模式
    - Proxy代理模式
      - 静态代理
        - 将对象A作为对象B的私有引用对象，由对象B代理细化对象A的实现，比如你要相亲（动作），你妈代理了这个动作，帮你找媒婆，打听情况等等（前置动作），再把你推出去

- 相亲（你执行动作），相完亲你妈还要各方打听你们两合不合适，对方父母怎么看等等（后置动作）
- 动态代理
  - 运行时代理，aop实现的核心，jdk动态代理需要对象有接口存在，原理是通过反射；cglib动态代理是通过类的继承创建子类来代理实现
- Factory工厂模式
  - 简单工厂模式
    - 一个工厂创建多种实例，耦合度高，每加一种实例需要修改原方法
  - 抽象工厂模式
    - 需要对工厂进行更高维度的抽象，也就是创建工厂的工厂，比如生产球的工厂接口，继承接口有生产篮球厂接口，生产排球厂接口等等，各自工厂实现生产不同球这个过程；当要新增一个乒乓球工厂的时候，只需要继承生产球工厂的接口，实现对应生产方法就行
- Singleton单例模式
  - 饿汉式
    - 程序启动，立即加载，线程安全的
  - 懒汉式
    - 需要用到时才加载，可能导致线程不安全，需要注意改造
  - 登记式
    - 非正式的称呼，取对象的时候初始化，并放入map容器中，第二次直接从map中获取
  - 枚举式
    - 基于枚举可以实现单例，因为枚举可以保证只有一个
- Delegate委派模式
  - 核心就是Leader干的事情，实际不是Leader在干，类内部有其它类实现了相同接口，Leader委派给员工类去实现对应的功能，但表面看起来像是Leader在干
- 原型模式
  - 浅拷贝
    - 基本数据类型直接赋值，引用数据类型进行引用传递，也就是内存地址的传递
  - 深拷贝
    - 基本数据类型复制过来，引用数据类型对象也进行复制
  - 实现方法
    - 对象实现Cloneable接口，若对象内成员变量有对象也要拷贝，也需要实现Cloneable接口
    - 通过序列化也可以实现拷贝
- 策略模式
  - 抽象策略接口，通过不同的实现类实现不同层面的相同功能，例如支付可以抽象一个支付方法的接口，其实现可以是支付宝、微信、网银等等，这就是不同的策略
- 模板模式
  - 父类实现抽象方法，延迟到子类实现，或者直接传入接口，由其实现类或实现方法去具体实现；比如JdbcTemplate中就用到了模板模式

- 适配器模式
  - 类适配器
    - 通过引用需要适配的类，进行组合实现
  - 对象适配器
    - 通过继承适配类进行实现
  - 接口适配器
    - 通过抽象类来进行适配
- 装饰器模式
  - 通过继承或引用对原有对象进行添加附加责任
- 观察者模式
  - 常用于监听模式，一个对象发生变化会使多个对象得到通知

- 设计原则
  - 单一职责原则（SRP）
    - 定义一个类，应该只有一个引起它变化的原因，该原因就是职责
  - 开闭原则（OCP）
    - 对扩展开放，对修改封闭
  - 里氏替换原则（LSP）
    - 任何基类可以出现的地方，其子类一定可以出现
  - 接口隔离原则（ISP）
    - 使用多个隔离的接口，比使用单个接口好，可以降低类间的耦合性
  - 依赖倒置原则（DIP）
    - 要求调用者和被调用者都是依赖抽象，两者没有直接的关联和接触，一方变动不会影响另一方，强调了抽象的重要性。一句话就是依赖于抽象而不依赖于具体
  - 迪米特原则（最少知道原则）
    - 一个实体类尽量少的和其它实体产生依赖，使模块功能独立
  - 合成/聚合复用（CARP）
    - 类的继承导致的耦合性也会比较高，比如父类接口增加一个方法，会导致所有子类编译出错；但如果只是通过组合聚合，引用类的方法，就可以降低风险，同时实现复用

- volatile
  - 概述
    - 当一个变量被volatile修饰时，它会保证修改的值立即被更新到主内存，保证其它线程课件
  - 性质
    - 可见性，见概述
    - 有序性
      - 防止指令重排序
    - 不能保证原子性

- JMM（Java内存模型）
  - 主内存
    - 存放静态字段、实例字段、数组元素等（线程公有）
  - 工作内存

- 存放局部变量和方法参数（线程私有）
- Happens-before
  - JMM模型规定了指令执行顺序，避免一些操作在工作内存改变变量后不能及时刷新到主内存，使其它线程不可见

- Spring
  - IOC
    - 依赖注入，本质是通过Spring帮你实例化对象并存储在容器中帮你管理Bean，当需要使用的时候进行对象注入
  - AOP
    - 使用JDK动态代理或CGlib动态代理对需要进行切面处理的对象进行代理
  - ApplicationContext
    - 是BeanFactory的子类，提供更多功能，比如AOP特性，载入多个上下文；容器启动时立即加载所有Bean
  - BeanFactory
    - 负责Bean配置文档的读取，bean加载，实例化，维护Bean之间的依赖关系；采用延迟加载
  - **SpringBean初始化过程**
    - 1. 实例化BeanFactoryPostProcessor实现类
    - 2. 执行BeanFactoryPostProcessor的postProcessBeanFactory方法
    - 3. 实例化BeanPostProcessor实现类
    - 4. 实例化InstantiationAwareBeanPostProcessor的postProcessBeforeInstantiation方法
    - 5. 执行Bean的构造器
    - 6. 执行InstantiationAwareBeanPostProcessor的postProcessPropertyValues方法
    - 7. 为Bean注入属性
    - 8. 调用BeanNameAware的setBeanName方法
    - 9. 调用BeanFactoryAware的setBeanFactory方法
    - 10. 执行BeanPostProcessor的postProcessBeforeInitialization方法
    - 11. 调用InitializatingBean的afterPropertiesSet
    - 12. 调用bean中init-method指定的初始化方法
    - 13. 执行BeanPostProcessor的postProcessAfterInitialization方法
    - 14. 结束后，调用DiposibleBean的destory方法
    - 15. 调用bean的destory-method指定的方法
  - Spring Bean的作用域
    - singleton
      - 单例，无论多少请求Bean实例只有一个，由BeanFactory维护
    - prototype
      - 与单例相反，每个请求提供一个实例，是多例的
    - request
      - 每一个客户端请求创建一个实例，请求完成后失效回收
    - Session

- 每一个会话有效期内保存一个实例，会话过期后失效，通常情况是用户登录开始算，到期后session失效需重新登录
- Java8 Stream
  - 对一串数据流Stream的各方面处理
  - map
    - 转换
  - filter
    - 过滤
  - sorted
    - 排序
  - foreach
    - 遍历
  - Collectors
    - averagingInt计算平均值
    - summarizingInt统计数量，最大，最小，平均，总和
  - FlatMap
    - 对象的多级处理，对象包含对象无线包含，流式处理
  - Reduce
    - 将多个元素组合起来得到一个元素，多元素的复杂操作
  - Parallel
    - 并行流计算，多线程运算
- MySQL
  - 主键和唯一索引
    - 1. 主键一定会创建唯一索引，但唯一索引不一定是主键
    - 2. 主键不允许为空，唯一索引列允许为空
    - 3. 一个表只能有一个主键，但可以有多个唯一索引（联合主键只能看成一个主键，因为是多个主键字段表示唯一行）
    - 4. 主键可以被其他表引用为外键，唯一索引不行
    - 5. 主键是一种约束，唯一索引是一种索引，是表的冗余数据结构，用来快速查找，两者有本质区别
- TCP和UDP
  - 1. TCP是面向连接的，UDP是无连接的
  - 2. TPC是可靠服务，保证数据的完整性，不重复，有序，无差错；UDP不保证数据完整性
  - 3. TCP是一对一连接，UDP可以有一对一，一对多多对一，多对多
  - 4. TCP提供了丢包重试机制，应答机制，有序接收机制保证数据可靠性
- 关于String
  - String是final类型的不允许修改，你看到的字符串相加产生的对象都是新对象
  - 在需要多个字符串相加的情况下使用**StringBuffer（线程安全的）或StringBuilder（线程不安全的）**效率更高
- JDK常用工具

- 常用命令
  - jps
    - jps -lvm输出JVM运行进程
  - jstack
    - 用来检测方法栈，比如检查内存溢出1、top -Hp pid1查出最消耗CPU线程 2、printf "%x\n" pid2 对该线程进行16进制转换 3、jstack pid1 | grep pid2 查看可能出问题的栈信息
  - jmap
    - 查看堆内存相关信息
  - jstat
    - 查看gc相关信息
- 网络七层协议OSI
  - 物理层
    - 定义物理设备标准进行比特流传输
  - 数据链路层
    - 数据检查纠错，保证正确传输帧数据
    - 数据检查纠错，保证正确传输帧数据
  - 网络层
    - 数据的路由寻址，可以是IP，ICMP等
  - 传输层
    - 提供端对端的接口，TCP，UDP
  - 会话层
    - 会话的建立与结束，数据传输同步
  - 表示层
    - 数据格式的转化
  - 应用层
    - 与用户应用程序的接口，比如实现Http，ftp，smtp，dns，telnet等
- JVM类加载机制
  - 1. 加载
    - 内存中生成类的Class对象，作为方法区这个类的数据入口
  - 2. 验证
    - 确保Class文件字节流包含信息符合虚拟机要求
  - 3. 准备
    - 为类变量分配内存并设置初始值，这里的初始值指的基本类型的初始值，比如int类型初始值0
  - 4. 解析
    - 将常量池中的符号引用替换为直接引用的过程
  - 5. 初始化
    - 真正执行类中定义的Java程序代码，执行构造器方法
  - 6. 使用
    - 使用对象处理业务逻辑
  - 7. 卸载

- 销毁对象类
- 类加载器
  - Bootstrap classloader启动类加载器，属于根加载器，加载jdk的JAVA_HOME\lib目录下类
  - Extension classloader扩展类加载器，加载JAVA_HOME\lib\ext扩展类
  - Application Classloader应用加载器，负责加载用户路径classpath上的类库
  - JVM双亲委派模型
    - 当一个类加载器收到类加载任务，会先交给其父类加载器去完成，最终会把任务传到启动类加载器，只有当父类加载器无法完成加载任务是，才会交给其子类
    - 保证了可信任类的优先级，比如用户自己建了一个java.lang.Object对象，该机制会保证加载系统认可的lib包下的对象，保证系统安全性
- http
  - 简介
    - 超文本传输协议，基于TCP/IP通信协议来传递数据
  - 特点
    - 简单快速
      - 客户端向服务器请求服务时，只需传达请求方法和路径，请求方法常用GET，HEAD，POST。由于HTTP协议简单，程序规模小，因而通信速度很快
    - 灵活
      - Http允许传递任意类型的数据对象，传输类型由Content-Type加以标记
    - 无连接
      - 无连接的含义是指每次连接只处理一个请求，处理完后自动断开，为节省传输时间和资源
    - 无状态
      - http协议是无状态协议，指的是请求不会依赖于前一个请求的状态参数等，所以如果要用到前一个请求的参数，需要再次携带前一次的参数进行请求
  - http状态码
    - 1xx：指示信息--请求已接收，继续处理
    - 2xx：成功--表示请求已被成功接收、理解、接收
    - 3xx：重定向--请求需要更进一步的操作，通常是转发到别的地址或页面
    - 4xx：客户端错误--请求有语法错误或请求无法实现
      - 400：请求语法错误，不能被服务器所理解
      - 401：未经授权，这个状态码必须WWW-Authenticate报头域一起使用
      - 403：服务器收到请求，但拒绝提供服务
      - 404：请求资源不存在，很可能是输错了URL地址
    - 5xx：服务器端错误--服务器未能实现合法请求
      - 500：服务器发生不可预期的错误，80%的可能是你服务器报空指针异常了^_^
      - 503：服务器当前不能处理客户端请求，一段时间后可能恢复正常，有可能是服务正在重启