## 优质Java 面试题附答案解析

**前言**

个人觉得面试也像是一场全新的征程，失败和胜利都是平常之事。所以，劝各位不要因为面试失败而灰心、 丧失斗志。也不要因为面试通过而沾沾自喜，等待你的将是更美好的未来，继续加油！

本篇分享的面试题内容包括：Java、MyBatis、ZooKeeper、Dubbo、Elasticsearch、Redis、MySQL、Spring、Spring Boot、Spring Cloud、RabbitMQ、Kafka、Linux 等技术栈。

**目录**

[2020 年字节跳动 Java 面试题附答案解析 前言 1](#_bookmark0)

[一、Java 基础系列面试题 1](#_bookmark1)

[1、面向对象的特征有哪些方面？ 1](#_bookmark2)

[2、访问修饰符 public,private,protected,以及不写（默认）时的区别？ 2](#_bookmark3)

[3、String 是最基本的数据类型吗？ 2](#_bookmark4)

[4、float f=3.4;是否正确？ 2](#_bookmark5)

[5、short s1 = 1; s1 = s1 + 1;有错吗?short s1 = 1; s1 += 1;有错吗？ 2](#_bookmark6)

[6、Java 有没有 goto？ 2](#_bookmark7)

[7、int 和 Integer 有什么区别？ 3](#_bookmark8)

[8、&和&&的区别？ 5](#_bookmark9)

[9、解释内存中的栈(stack)、堆(heap)和方法区(method area)的用法。 5](#_bookmark10)

[10、Math.round(11.5) 等于多少？Math.round(-11.5)等于多少？ 6](#_bookmark11)

[11、switch 是否能作用在 byte 上，是否能作用在 long 上，是否能作用在 String 上？](#_bookmark12)

..................................................................................................................................................... 6

[12、用最有效率的方法计算 2 乘以 8？ 6](#_bookmark13)

[13、数组有没有 length()方法？String 有没有 length()方法？ 8](#_bookmark14)

[14、在 Java 中，如何跳出当前的多重嵌套循环？ 8](#_bookmark15)

[15、构造器（constructor）是否可被重写（override）？ 8](#_bookmark16)

[16、两个对象值相同(x.equals(y) == true)，但却可有不同的 hash code，这句话对不对？8](#_bookmark17) [17、是否可以继承 String 类？ 9](#_bookmark18)

[18、当一个对象被当作参数传递到一个方法后，此方法可改变这个对象的属性，并可返](#_bookmark19)

[回变化后的结果，那么这里到底是值传递还是引用传递？ 9](#_bookmark19)

[19、String 和 StringBuilder、StringBuffer 的区别？ 10](#_bookmark20)

[20、重载（Overload）和重写（Override）的区别。重载的方法能否根据返回类型进行](#_bookmark21)区分？ 11

[21、描述一下 JVM 加载 class 文件的原理机制？ 11](#_bookmark22)

[22、char 型变量中能不能存贮一个中文汉字，为什么？ 12](#_bookmark23)

[23、抽象类（abstract class）和接口（interface）有什么异同？ 12](#_bookmark24)

[24、静态嵌套类(Static Nested Class)和内部类（Inner Class）的不同？ 12](#_bookmark25)

[25、Java 中会存在内存泄漏吗，请简单描述。 16](#_bookmark26)

[26、抽象的（abstract）方法是否可同时是静态的（static）,是否可同时是本地方法（native）](#_bookmark27)，是否可同时被 synchronized 修饰？ 17

[27、阐述静态变量和实例变量的区别。 17](#_bookmark28)

[28、是否可以从一个静态（static）方法内部发出对非静态（non-static）方法的调用？17](#_bookmark29) [29、如何实现对象克隆？ 17](#_bookmark30)

[30、GC 是什么？为什么要有 GC？ 21](#_bookmark31)

[31、String s = new String(“xyz”);创建了几个字符串对象？ 22](#_bookmark32)

[32、接口是否可继承（extends）接口？抽象类是否可实现（implements）接口？抽象类](#_bookmark33) 是否可继承具体类（concreteclass）？ 22

[33、一个”.java”源文件中是否可以包含多个类（不是内部类）？有什么限制？ 23](#_bookmark34)

[34、Anonymous Inner Class(匿名内部类)是否可以继承其它类？是否可以实现接口？. 23](#_bookmark35) [35、内部类可以引用它的包含类（外部类）的成员吗？有没有什么限制？ 23](#_bookmark36)

[36、Java 中的 final 关键字有哪些用法？ 23](#_bookmark37)

[37、指出下面程序的运行结果 23](#_bookmark38)

[38、数据类型之间的转换： 24](#_bookmark39)

[39、如何实现字符串的反转及替换？ 24](#_bookmark40)

[40、怎样将 GB2312 编码的字符串转换为 ISO-8859-1 编码的字符串？ 24](#_bookmark41)

[41、日期和时间： 25](#_bookmark42)

[42、打印昨天的当前时刻。 26](#_bookmark43)

[43、比较一下 Java 和 JavaSciprt。 27](#_bookmark44)

[44、什么时候用断言（assert）？ 27](#_bookmark45)

[45、Error 和 Exception 有什么区别？ 28](#_bookmark46)

[46、try{}里有一个 return 语句，那么紧跟在这个 try 后的 finally{}里的代码会不会被执](#_bookmark47)行，什么时候被执行，在 return 前还是后? 28

[47、Java 语言如何进行异常处理，关键字：throws、throw、try、catch、finally 分别如](#_bookmark48)何使用？ 28

[48、运行时异常与受检异常有何异同？ 29](#_bookmark49)

[49、列出一些你常见的运行时异常？ 29](#_bookmark50)

[50、阐述 final、finally、finalize 的区别。 29](#_bookmark51)

[51、类 ExampleA 继承 Exception，类 ExampleB 继承 ExampleA。 30](#_bookmark52)

[52、List、Set、Map 是否继承自 Collection 接口？ 31](#_bookmark53)

[53、阐述 ArrayList、Vector、LinkedList 的存储性能和特性。 31](#_bookmark54)

[54、Collection 和 Collections 的区别？ 32](#_bookmark55)

[55、List、Map、Set 三个接口存取元素时，各有什么特点？ 32](#_bookmark56)

[56、TreeMap 和 TreeSet 在排序时如何比较元素？Collections 工具类中的 sort()方法如](#_bookmark57)

[何比较元素？ 32](#_bookmark57)

[57、Thread 类的 sleep()方法和对象的 wait()方法都可以让线程暂停执行，它们有什么](#_bookmark58)区别? 33

[58、线程的 sleep()方法和 yield()方法有什么区别？ 33](#_bookmark59)

[59、当一个线程进入一个对象的 synchronized 方法 A 之后，其它线程是否可进入此对](#_bookmark60)象的 synchronized 方法 B？ 34

[60、请说出与线程同步以及线程调度相关的方法。 34](#_bookmark61)

[61、编写多线程程序有几种实现方式？ 34](#_bookmark62)

[62、synchronized 关键字的用法？ 34](#_bookmark63)

[63、举例说明同步和异步。 35](#_bookmark64)

[64、启动一个线程是调用 run()还是 start()方法？ 35](#_bookmark65)

[65、什么是线程池（thread pool）？ 35](#_bookmark66)

[66、线程的基本状态以及状态之间的关系？ 36](#_bookmark67)

[67、简述 synchronized 和 java.util.concurrent.locks.Lock 的异同？ 36](#_bookmark68)

[68、Java 中如何实现序列化，有什么意义？ 36](#_bookmark69)

[69、Java 中有几种类型的流？ 37](#_bookmark70)

[70、写一个方法，输入一个文件名和一个字符串，统计这个字符串在这个文件中出现的](#_bookmark71) 次数。 37

[71、如何用 Java 代码列出一个目录下所有的文件？ 38](#_bookmark72)

[72、用 Java 的套接字编程实现一个多线程的回显（echo）服务器。 39](#_bookmark73)

[73、XML 文档定义有几种形式？它们之间有何本质区别？解析 XML 文档有哪几种方](#_bookmark74)式？ 40

[74、你在项目中哪些地方用到了 XML？ 40](#_bookmark75)

[75、阐述 JDBC 操作数据库的步骤。 40](#_bookmark76)

[76、Statement 和 PreparedStatement 有什么区别？哪个性能更好？ 41](#_bookmark77)

[77、使用 JDBC 操作数据库时，如何提升读取数据的性能？如何提升更新数据的性能？](#_bookmark78)

................................................................................................................................................... 42

[78、在进行数据库编程时，连接池有什么作用？ 42](#_bookmark79)

[79、什么是 DAO 模式？ 42](#_bookmark80)

[80、事务的 ACID 是指什么？ 43](#_bookmark81)

[81、JDBC 中如何进行事务处理？ 44](#_bookmark82)

[82、JDBC 能否处理 Blob 和 Clob？ 44](#_bookmark83)

[83、简述正则表达式及其用途。 44](#_bookmark84)

[84、Java 中是如何支持正则表达式操作的？ 44](#_bookmark85)

[85、获得一个类的类对象有哪些方式？ 45](#_bookmark86)

[86、如何通过反射创建对象？ 45](#_bookmark87)

[87、如何通过反射获取和设置对象私有字段的值？ 45](#_bookmark88)

[88、如何通过反射调用对象的方法？ 45](#_bookmark89)

[89、简述一下面向对象的”六原则一法则”。 46](#_bookmark90)

[90、简述一下你了解的设计模式。 47](#_bookmark91)

[91、用 Java 写一个单例类。 48](#_bookmark92)

[92、什么是 UML？ 49](#_bookmark93)

[93、UML 中有哪些常用的图？ 49](#_bookmark94)

[94、用 Java 写一个冒泡排序。 49](#_bookmark95)

[95、用 Java 写一个折半查找。 51](#_bookmark96)

[Java 面试题（二） 53](#_bookmark97)

[四、Redis 面试题 53](#_bookmark98)

[1、Java 中能创建 volatile 数组吗？ 53](#_bookmark99)

[2、volatile 能使得一个非原子操作变成原子操作吗？ 54](#_bookmark100)

[3、volatile 修饰符的有过什么实践？ 54](#_bookmark101)

[4、volatile 类型变量提供什么保证？ 54](#_bookmark102)

[5、10 个线程和 2 个线程的同步代码，哪个更容易写？ 54](#_bookmark103)

[6、你是如何调用 wait（）方法的？使用 if 块还是循环？为什么？ 55](#_bookmark104)

[7、什么是多线程环境下的伪共享（false sharing）？ 55](#_bookmark105)

[8、什么是 Busy spin？我们为什么要使用它？ 55](#_bookmark106)

[9、Java 中怎么获取一份线程 dump 文件？ 55](#_bookmark107)

[10、Swing 是线程安全的？ 55](#_bookmark108)

[11、什么是线程局部变量？ 56](#_bookmark109)

[12、用 wait-notify 写一段代码来解决生产者-消费者问题？ 56](#_bookmark110)

[13、用 Java 写一个线程安全的单例模式（Singleton）？ 56](#_bookmark111)

[14、Java 中 sleep 方法和 wait 方法的区别？ 56](#_bookmark112)

[15、什么是不可变对象（immutable object）？Java 中怎么创建一个不可变对象？ 56](#_bookmark113)

[16、我们能创建一个包含可变对象的不可变对象吗？ 57](#_bookmark114)

[17、Java 中应该使用什么数据类型来代表价格？ 57](#_bookmark115)

[18、怎么将 byte 转换为 String？ 57](#_bookmark116)

[19、Java 中怎样将 bytes 转换为 long 类型？ 57](#_bookmark117)

[20、我们能将 int 强制转换为 byte 类型的变量吗？如果该值大于 byte 类型的范围，](#_bookmark118) 将会出现什么现象？ 59

[21、存在两个类，B 继承 A，C 继承 B，我们能将 B 转换为 C 么？如 C = (C) B； 59](#_bookmark119)

[22、哪个类包含 clone 方法？是 Cloneable 还是 Object？ 59](#_bookmark120)

[23、Java 中 ++ 操作符是线程安全的吗？ 60](#_bookmark121)

[24、a = a + b 与 a += b 的区别 60](#_bookmark122)

[25、我能在不进行强制转换的情况下将一个 double 值赋值给 long 类型的变量吗？ 60](#_bookmark123)

[26、3\*0.1 == 0.3 将会返回什么？true 还是 false？ 60](#_bookmark124)

[27、int 和 Integer 哪个会占用更多的内存？ 60](#_bookmark125)

[28、为什么 Java 中的 String 是不可变的（Immutable）？ 61](#_bookmark126)

[29、我们能在 Switch 中使用 String 吗？ 61](#_bookmark127)

[30、Java 中的构造器链是什么？ 61](#_bookmark128)

[31、64 位 JVM 中，int 的长度是多数？ 61](#_bookmark129)

[32、Serial 与 Parallel GC 之间的不同之处？ 61](#_bookmark130)

[33、32 位和 64 位的 JVM，int 类型变量的长度是多数？ 61](#_bookmark131)

[34、Java 中 WeakReference 与 SoftReference 的区别？ 61](#_bookmark132)

[35、WeakHashMap 是怎么工作的？ 62](#_bookmark133)

[36、JVM 选项 -XX:+UseCompressedOops 有什么作用？为什么要使用？ 62](#_bookmark134)

[37、怎样通过 Java 程序来判断 JVM 是 32 位 还是 64 位？ 62](#_bookmark135)

[38、32 位 JVM 和 64 位 JVM 的最大堆内存分别是多数？ 62](#_bookmark136)

[39、JRE、JDK、JVM 及 JIT 之间有什么不同？ 62](#_bookmark137)

[40、解释 Java 堆空间及 GC？ 63](#_bookmark138)

[41、你能保证 GC 执行吗？ 63](#_bookmark139)

[42、怎么获取 Java 程序使用的内存？堆使用的百分比？ 63](#_bookmark140)

[43、Java 中堆和栈有什么区别？ 63](#_bookmark141)

[44、“a==b”和”a.equals(b)”有什么区别？ 63](#_bookmark142)

[45、a.hashCode() 有什么用？与 a.equals(b) 有什么关系？ 64](#_bookmark143)

[46、final、finalize 和 finally 的不同之处？ 64](#_bookmark144)

[47、Java 中的编译期常量是什么？使用它又什么风险？ 64](#_bookmark145)

[48、List、Set、Map 和 Queue 之间的区别(答案) 64](#_bookmark146)

[49、poll() 方法和 remove() 方法的区别？ 64](#_bookmark147)

[50、Java 中 LinkedHashMap 和 PriorityQueue 的区别是什么？ 65](#_bookmark148)

[51、ArrayList 与 LinkedList 的不区别？ 65](#_bookmark149)

[52、用哪两种方式来实现集合的排序？ 65](#_bookmark150)

[53、Java 中怎么打印数组？ 65](#_bookmark151)

[54、Java 中的 LinkedList 是单向链表还是双向链表？ 65](#_bookmark152)

[二、Java 并发系列面试题 78](#_bookmark153)

[1、在 java 中守护线程和本地线程区别？ 78](#_bookmark154)

[2、线程与进程的区别？ 78](#_bookmark155)

[3、什么是多线程中的上下文切换？ 79](#_bookmark156)

[4、死锁与活锁的区别，死锁与饥饿的区别？ 79](#_bookmark157)

[6、什么是线程组，为什么在 Java 中不推荐使用？ 79](#_bookmark158)

[7、为什么使用 Executor 框架？ 79](#_bookmark159)

[8、在 Java 中 Executor 和 Executors 的区别？ 80](#_bookmark160)

[9、如何在 Windows 和 Linux 上查找哪个线程使用的 CPU 时间最长？ 80](#_bookmark161)

[10、什么是原子操作？在 Java Concurrency API 中有哪些原子类(atomic classes)？ 80](#_bookmark162)

[11、Java Concurrency API 中的 Lock 接口(Lock interface)是什么？对比同步它有什么优](#_bookmark163)势？ 81

[12、什么是 Executors 框架？ 81](#_bookmark164)

[13、什么是阻塞队列？阻塞队列的实现原理是什么？如何使用阻塞队列来实现生产者-](#_bookmark165)

消费者模型？ 81

[14、什么是 Callable 和 Future? 82](#_bookmark166)

[15、什么是 FutureTask?使用 ExecutorService 启动任务。 82](#_bookmark167)

[16、什么是并发容器的实现？ 83](#_bookmark168)

[17、多线程同步和互斥有几种实现方法，都是什么？ 83](#_bookmark169)

[18、什么是竞争条件？你怎样发现和解决竞争？ 83](#_bookmark170)

[19、你将如何使用 thread dump？你将如何分析 Thread dump？ 84](#_bookmark171)

[20、为什么我们调用 start()方法时会执行 run()方法，为什么我们不能直接调用 run()](#_bookmark172)

方法？ 84

[21、Java 中你怎样唤醒一个阻塞的线程？ 84](#_bookmark173)

[22、在 Java 中 CycliBarriar 和 CountdownLatch 有什么区别？ 85](#_bookmark174)

[23、什么是不可变对象，它对写并发应用有什么帮助？ 85](#_bookmark175)

[24、什么是多线程中的上下文切换？ 86](#_bookmark176)

[25、Java 中用到的线程调度算法是什么？ 86](#_bookmark177)

[26、什么是线程组，为什么在 Java 中不推荐使用？ 86](#_bookmark178)

[27、为什么使用 Executor 框架比使用应用创建和管理线程好？ 87](#_bookmark179)

[28、java 中有几种方法可以实现一个线程？ 87](#_bookmark180)

[29、如何停止一个正在运行的线程？ 87](#_bookmark181)

[30、notify()和 notifyAll()有什么区别？ 88](#_bookmark182)

[31、什么是 Daemon 线程？它有什么意义？ 88](#_bookmark183)

[32、java 如何实现多线程之间的通讯和协作？ 88](#_bookmark184)

[33、什么是可重入锁（ReentrantLock）？ 88](#_bookmark185)

[34、当一个线程进入某个对象的一个 synchronized 的实例方法后，其它线程是否可进](#_bookmark186)入此对象的其它方法？ 89

[35、乐观锁和悲观锁的理解及如何实现，有哪些实现方式？ 89](#_bookmark187)

[36、SynchronizedMap 和 ConcurrentHashMap 有什么区别？ 90](#_bookmark188)

[37、CopyOnWriteArrayList 可以用于什么应用场景？ 90](#_bookmark189)

[38、什么叫线程安全？servlet 是线程安全吗? 91](#_bookmark190)

[39、volatile 有什么用？能否用一句话说明下 volatile 的应用场景？ 91](#_bookmark191)

[40、为什么代码会重排序？ 91](#_bookmark192)

[41、在 java 中 wait 和 sleep 方法的不同？ 91](#_bookmark193)

[42、用 Java 实现阻塞队列 92](#_bookmark194)

[43、一个线程运行时发生异常会怎样？ 92](#_bookmark195)

[44、如何在两个线程间共享数据？ 92](#_bookmark196)

[45、Java 中 notify 和 notifyAll 有什么区别？ 92](#_bookmark197)

[46、为什么 wait, notify 和 notifyAll 这些方法不在 thread 类里面？ 92](#_bookmark198)

[47、什么是 ThreadLocal 变量？ 92](#_bookmark199)

[48、Java 中 interrupted 和 isInterrupted 方法的区别？ 93](#_bookmark200)

[49、为什么 wait 和 notify 方法要在同步块中调用？ 93](#_bookmark201)

[50、为什么你应该在循环中检查等待条件? 93](#_bookmark202)

[51、Java 中的同步集合与并发集合有什么区别？ 93](#_bookmark203)

[52、什么是线程池？ 为什么要使用它？ 94](#_bookmark204)

[53、怎么检测一个线程是否拥有锁？ 94](#_bookmark205)

[54、你如何在 Java 中获取线程堆栈？ 94](#_bookmark206)

[55、JVM 中哪个参数是用来控制线程的栈堆栈小的? 94](#_bookmark207)

[56、Thread 类中的 yield 方法有什么作用？ 94](#_bookmark208)

[57、Java 中 ConcurrentHashMap 的并发度是什么？ 94](#_bookmark209)

[58、Java 中 Semaphore 是什么？ 95](#_bookmark210)

[59、Java 线程池中 submit() 和 execute()方法有什么区别？ 95](#_bookmark211)

[60、什么是阻塞式方法？ 95](#_bookmark212)

[61、Java 中的 ReadWriteLock 是什么？ 95](#_bookmark213)

[62、volatile 变量和 atomic 变量有什么不同？ 95](#_bookmark214)

[63、可以直接调用 Thread 类的 run ()方法么？ 96](#_bookmark215)

[64、如何让正在运行的线程暂停一段时间？ 96](#_bookmark216)

[65、你对线程优先级的理解是什么？ 96](#_bookmark217)

[66、什么是线程调度器(Thread Scheduler)和时间分片(Time Slicing )？ 96](#_bookmark218)

[67、你如何确保 main()方法所在的线程是 Java 程序最后结束的线程？ 97](#_bookmark219)

[68、线程之间是如何通信的？ 97](#_bookmark220)

[69、为什么线程通信的方法 wait(), notify()和 notifyAll()被定义在 Object 类里？ 97](#_bookmark221)

[70、为什么 wait(), notify()和 notifyAll ()必须在同步方法或者同步块中被调用？ 97](#_bookmark222)

[71、为什么 Thread 类的 sleep()和 yield ()方法是静态的？ 97](#_bookmark223)

[72、如何确保线程安全？ 98](#_bookmark224)

[73、同步方法和同步块，哪个是更好的选择？ 98](#_bookmark225)

[74、如何创建守护线程？ 98](#_bookmark226)

[75、什么是 Java Timer 类？如何创建一个有特定时间间隔的任务？ 98](#_bookmark227)

[四、Redis 面试题 98](#_bookmark228)

[1、什么是 Redis? 98](#_bookmark229)

[2、Redis 的数据类型？ 99](#_bookmark230)

[3、使用 Redis 有哪些好处？ 100](#_bookmark231)

[4、Redis 相比 Memcached 有哪些优势？ 100](#_bookmark232)

[5、Memcache 与 Redis 的区别都有哪些？ 100](#_bookmark233)

[6、Redis 是单进程单线程的？ 100](#_bookmark234)

[7、一个字符串类型的值能存储最大容量是多少？ 100](#_bookmark235)

[9、Redis 常见性能问题和解决方案： 101](#_bookmark236)

[10、redis 过期键的删除策略？ 102](#_bookmark237)

[11、Redis 的回收策略（淘汰策略）? 102](#_bookmark238)

[12、为什么 edis 需要把所有数据放到内存中？ 102](#_bookmark239)

[13、Redis 的同步机制了解么？ 103](#_bookmark240)

[14、Pipeline 有什么好处，为什么要用 pipeline？ 103](#_bookmark241)

[15、是否使用过 Redis 集群，集群的原理是什么？ 103](#_bookmark242)

[16、Redis 集群方案什么情况下会导致整个集群不可用？ 103](#_bookmark243)

[17、Redis 支持的 Java 客户端都有哪些？官方推荐用哪个？ 103](#_bookmark244)

[18、Jedis 与 Redisson 对比有什么优缺点？ 104](#_bookmark245)

[19、Redis 如何设置密码及验证密码？ 104](#_bookmark246)

[20、说说 Redis 哈希槽的概念？ 104](#_bookmark247)

[21、Redis 集群的主从复制模型是怎样的？ 104](#_bookmark248)

[22、Redis 集群会有写操作丢失吗？为什么？ 104](#_bookmark249)

[23、Redis 集群之间是如何复制的？ 104](#_bookmark250)

[24、Redis 集群最大节点个数是多少？ 104](#_bookmark251)

[25、Redis 集群如何选择数据库？ 105](#_bookmark252)

[26、怎么测试 Redis 的连通性？ 105](#_bookmark253)

[27、怎么理解 Redis 事务？ 105](#_bookmark254)

[28、Redis 事务相关的命令有哪几个？ 105](#_bookmark255)

[29、Redis key 的过期时间和永久有效分别怎么设置？ 105](#_bookmark256)

[30、Redis 如何做内存优化？ 105](#_bookmark257)

[31、Redis 回收进程如何工作的？ 105](#_bookmark258)

[32、都有哪些办法可以降低 Redis 的内存使用情况呢？ 106](#_bookmark259)

[33、Redis 的内存用完了会发生什么？ 106](#_bookmark260)

[34、一个 Redis 实例最多能存放多少的 keys？List、Set、Sorted Set 他们最多能存放多](#_bookmark261)少元素？ 106

[35、MySQL 里有 2000w 数据，redis 中只存 20w 的数据，如何保证 redis 中的数据](#_bookmark262)都是热点数据？ 106

[36、Redis 最适合的场景？ 107](#_bookmark263)

[37、假如 Redis 里面有 1 亿个 key，其中有 10w 个 key 是以某个固定的已知的前缀](#_bookmark264)

[开头的，如果将它们全部找出来？ 108](#_bookmark264)

[38、如果有大量的 key 需要设置同一时间过期，一般需要注意什么？ 108](#_bookmark265)

[39、使用过 Redis 做异步队列么，你是怎么用的？ 108](#_bookmark266)

[40、使用过 Redis 分布式锁么，它是什么回事？ 109](#_bookmark267)

[五、MyBatis 面试题答案解析 109](#_bookmark268)

[六、ZooKeeper 面试题答案解析 120](#_bookmark269)

[七、Spring Boot 面试题 131](#_bookmark270)

[1、什么是 Spring Boot？ 131](#_bookmark271)

[2、Spring Boot 有哪些优点？ 131](#_bookmark272)

[3、什么是 JavaConfig？ 132](#_bookmark273)

[4、如何重新加载 Spring Boot 上的更改，而无需重新启动服务器？ 132](#_bookmark274)

[5、Spring Boot 中的监视器是什么？ 132](#_bookmark275)

[6、如何在 Spring Boot 中禁用 Actuator 端点安全性？ 133](#_bookmark276)

[7、如何在自定义端口上运行 Spring Boot 应用程序？ 133](#_bookmark277)

[8、什么是 YAML？ 133](#_bookmark278)

[9、如何实现 Spring Boot 应用程序的安全性？ 133](#_bookmark279)

[10、如何集成 Spring Boot 和 ActiveMQ？ 133](#_bookmark280)

[11、如何使用 Spring Boot 实现分页和排序？ 133](#_bookmark281)

[12、什么是 Swagger？你用 Spring Boot 实现了它吗？ 134](#_bookmark282)

[13、什么是 Spring Profiles？ 134](#_bookmark283)

[14、什么是 Spring Batch？ 134](#_bookmark284)

[15、什么是 FreeMarker 模板？ 134](#_bookmark285)

[16、如何使用 Spring Boot 实现异常处理？ 134](#_bookmark286)

[17、您使用了哪些 starter maven 依赖项？ 135](#_bookmark287)

[18、什么是 CSRF 攻击？ 135](#_bookmark288)

[19、什么是 WebSockets？ 135](#_bookmark289)

[20、什么是 AOP？ 135](#_bookmark290)

[21、什么是 Apache Kafka？ 135](#_bookmark291)

[22、我们如何监视所有 Spring Boot 微服务？ 135](#_bookmark292)

[十、 kafka 面试题 152](#_bookmark293)

[十一、RabbitMQ 面试题 158](#_bookmark294)

[1、什么是 rabbitmq 159](#_bookmark295)

[2、为什么要使用 rabbitmq 159](#_bookmark296)

[3、使用 rabbitmq 的场景 159](#_bookmark297)

[4、如何确保消息正确地发送至 RabbitMQ？ 如何确保消息接收方消费了消息？ 159](#_bookmark298)

[5.如何避免消息重复投递或重复消费？ 160](#_bookmark299)

[6、消息基于什么传输？ 160](#_bookmark300)

[7、消息如何分发？ 160](#_bookmark301)

[8、消息怎么路由？ 160](#_bookmark302)

[9、如何确保消息不丢失？ 161](#_bookmark303)

[10、使用 RabbitMQ 有什么好处？ 161](#_bookmark304)

[11、RabbitMQ 的集群 161](#_bookmark305)

[12、mq 的缺点 161](#_bookmark306)

[十三、Elasticsearch 面试题 169](#_bookmark307)

[1、elasticsearch 了解多少，说说你们公司 es 的集群架构，索引数据大小，分片有多](#_bookmark308)

[少，以及一些调优手段 。 169](#_bookmark308)

[2、elasticsearch 的倒排索引是什么 170](#_bookmark309)

[3、elasticsearch 索引数据多了怎么办，如何调优，部署 171](#_bookmark310)

[4、elasticsearch 是如何实现 master 选举的 171](#_bookmark311)

[5、详细描述一下 Elasticsearch 索引文档的过程 172](#_bookmark312)

[6、详细描述一下 Elasticsearch 搜索的过程？ 172](#_bookmark313)

[7、Elasticsearch 在部署时，对 Linux 的设置有哪些优化方法 173](#_bookmark314)

[8、lucence 内部结构是什么？ 173](#_bookmark315)

[9、Elasticsearch 是如何实现 Master 选举的？ 173](#_bookmark316)

[10、Elasticsearch 中的节点（比如共 20 个），其中的 10 个 174](#_bookmark317)

[11、客户端在和集群连接时，如何选择特定的节点执行请求的？ 174](#_bookmark318)

[12、详细描述一下 Elasticsearch 索引文档的过程。 174](#_bookmark319)

[vi 文件名 #编辑方式查看，可修改 181](#_bookmark320)

[X 死掉的进程 183](#_bookmark321)

[2 /bin/su 复制代码 189](#_bookmark322)

# 一、Java 基础系列面试题

## 1、面向对象的特征有哪些方面？

#### 面向对象的特征主要有以下几个方面：

**抽象：**抽象是将一类对象的共同特征总结出来构造类的过程，包括数据抽象和行为抽象两方面。抽象只关注对象有哪些属性和行为，并不关注这些行为的细节是什么。

**继承：**继承是从已有类得到继承信息创建新类的过程。提供继承信息的类被称为父类（超类、基类）；得到继承信息的类被称为子类（派生类）。继承让变化中的软件系统有了一定的延续性，同时继承也是封装程序中可变因素的重要手段

（如果不能理解请阅读阎宏博士的《Java 与模式》或《设计模式精解》中关于桥梁模式的部分）。

**封装：**通常认为封装是把数据和操作数据的方法绑定起来，对数据的访问只能通过已定义的接口。面向对象的本质就是将现实世界描绘成一系列完全自治、封闭的对象。我们在类中编写的方法就是对实现细节的一种封装；我们编写一个类就是对数据和数据操作的封装。可以说，封装就是隐藏一切可隐藏的东西，只向外界提供最简单的编程接口（可以想想普通洗衣机和全自动洗衣机的差别，明显全自动洗衣机封装更好因此操作起来更简单；我们现在使用的智能手机也是封装得足够好的，因为几个按键就搞定了所有的事情）。

**多态性：**多态性是指允许不同子类型的对象对同一消息作出不同的响应。简单的说就是用同样的对象引用调用同样的方法但是做了不同的事情。多态性分为编译时的多态性和运行时的多态性。如果将对象的方法视为对象向外界提供的服务， 那么运行时的多态性可以解释为：当 A 系统访问 B 系统提供的服务时，B 系统有多种提供服务的方式，但一切对 A 系统来说都是透明的（就像电动剃须刀是A 系统，它的供电系统是 B 系统，B 系统可以使用电池供电或者用交流电，甚至还有可能是太阳能，A 系统只会通过 B 类对象调用供电的方法，但并不知道供电系统的底层实现是什么，究竟通过何种方式获得了动力）。方法重载

（overload）实现的是编译时的多态性（也称为前绑定），而方法重写（override） 实现的是运行时的多态性（也称为后绑定）。运行时的多态是面向对象最精髓的东西，要实现多态需要做两件事：

1. 方法重写（子类继承父类并重写父类中已有的或抽象的方法）；
2. 对象造型（用父类型引用引用子类型对象，这样同样的引用调用同样的方法就会根据子类对象的不同而表现出不同的行为）。

## 2、访问修饰符 public,private,protected,以及不写（默认）时的区别？

修饰符 当前类 同 包 子 类 其他包类的成员不写访问修饰时默认为 default。默认对于同一个包中的其他类相当于公开（public），对于不是同一个包中的其他类相当于私有（private）。受保护（protected）对子类相当于公开，对不是同一包中的没有父子关系的类相当于私有。Java 中，外部类的修饰符只能是public 或默认，类的成员（包括内部类）的修饰符可以是以上四种。

## 3、String 是最基本的数据类型吗？

不是。Java 中的基本数据类型只有 8 个 ：byte、short、int、long、float、double、char、boolean；除了基本类型（primitive type），剩下的都是引用类型（referencetype），Java 5 以后引入的枚举类型也算是一种比较特殊的引用类型。

## 4、float f=3.4;是否正确？

不正确。3.4 是双精度数，将双精度型（double）赋值给浮点型（float）属于下转型（down-casting，也称为窄化）会造成精度损失，因此需要强制类型转换float f =(float)3.4; 或者写成 float f =3.4F;。

## 5、short s1 = 1; s1 = s1 + 1;有错吗?short s1 = 1; s1 += 1;有错吗？

对于 short s1 = 1; s1 = s1 + 1;由于 1 是 int 类型，因此 s1+1 运算结果也是 int 型，需要强制转换类型才能赋值给 short 型。而 short s1 = 1; s1 += 1;可以正确编译，因为 s1+= 1;相当于 s1 = (short(s1 + 1);其中有隐含的强制类型转换。

## 6、Java 有没有 goto？

goto 是 Java 中的保留字，在目前版本的 Java 中没有使用。（根据 James Gosling（Java 之父）编写的《The Java Programming Language》一书的附录中给出了一个 Java 关键字列表，其中有 goto 和 const，但是这两个是目前无法使用的关键字，因此有些地方将其称之为保留字，其实保留字这个词应该有更广泛的意义，因为熟悉 C 语言的程序员都知道，在系统类库中使用过的有特殊意义的但词或单词的组合都被视为保留字）

## 7、int 和 Integer 有什么区别？

Java 是一个近乎纯洁的面向对象编程语言，但是为了编程的方便还是引入了基本数据类型，但是为了能够将这些基本数据类型当成对象操作，Java 为每一个基本数据类型都引入了对应的包装类型（wrapper class），int 的包装类就是Integer，从 Java 5 开始引入了自动装箱/拆箱机制，使得二者可以相互转换。Java 为每个原始类型提供了包装类型：

原始类型: boolean，char，byte，short，int，long，float，double

包装类型：Boolean，Character，Byte，Short，Integer，Long，Float，Double class AutoUnboxingTest {

public static void main(String[] args) { Integer a = new Integer(3); Integer b = 3;

// 将 3 自动装箱成 Integer 类型int c = 3;

System.out.println(a == b);

// false 两个引用没有引用同一对

象

System.out.println(a == c);

// true a 自动拆箱成 int 类型再和 c

比较

}

}复制代码

最近还遇到一个面试题，也是和自动装箱和拆箱有点关系的，代码如下所示：

public class Test03 {

public static void main(String[] args) {

Integer f1 = 100, f2 = 100, f3 = 150, f4 = 150;

System.out.println(f1 == f2); System.out.println(f3 == f4);

}

}复制代码

如果不明就里很容易认为两个输出要么都是 true 要么都是 false。首先需要注意的是 f1、f2、f3、f4 四个变量都是 Integer 对象引用，所以下面的==运算比较的不是值而是引用。装箱的本质是什么呢？当我们给一个 Integer 对象赋一个 int 值的时候，会调用 Integer 类的静态方法 valueOf，如果看 valueOf 的源代码就知道发生了什么。

public static Integer valueOf(int i) {

if (i >= IntegerCache.low && i <= IntegerCache.high) return IntegerCache.cache[i + (-IntegerCache.low)]; return new Integer(i);

}复制代码

IntegerCache 是 Integer 的内部类，其代码如下所示：

/\*\*

* Cache to support the object identity semantics of autoboxing for

values between

* -128 and 127 (inclusive) as required by JLS.

\*

* The cache is initialized on first usage. The size of the cache
* may be controlled by the {@code -XX:AutoBoxCacheMax=<size>} option.
* During VM initialization, java.lang.Integer.IntegerCache.high property
* may be set and saved in the private system properties in the
* sun.misc.VM class.

\*/

private static class IntegerCache { static final int low = -128; static final int high;

static final Integer cache[]; static {

// high value may be configured by property int h = 127;

String integerCacheHighPropValue =

high");

sun.misc.VM.getSavedProperty("java.lang.Integer.IntegerCache.

if (integerCacheHighPropValue != null) { try {

int i = parseint(integerCacheHighPropValue);

i = Math.max(i, 127);

// Maximum array size is

Integer.MAX\_VALUE

-1);

}

h = Math.min(i, Integer.MAX\_VALUE - (-low)

an int,

catch( NumberFormatException nfe) {

// If the property cannot be parsed into

ignore it.

}

}

high = h;

cache = new Integer[(high - low) + 1]; int j = low;

for (int k = 0; k < cache.length; k++) cache[k] = new Integer(j++);

// range [-128, 127] must be interned (JLS7 5.1.7) assert IntegerCache.high >= 127;

}

private IntegerCache() {

}

}复制代码

简单的说，如果整型字面量的值在-128 到 127 之间，那么不会 new 新的Integer 对象，而是直接引用常量池中的 Integer 对象，所以上面的面试题中f1f4 的结果是 false。

提醒：越是貌似简单的面试题其中的玄机就越多，需要面试者有相当深厚的功力。

## 8、&和&&的区别？

&运算符有两种用法：(1)按位与；(2)逻辑与。&&运算符是短路与运算。逻辑与跟短路与的差别是非常巨大的，虽然二者都要求运算符左右两端的布尔值都是true 整个表达式的值才是 true。&&之所以称为短路运算是因为，如果&&左边的表达式的值是 false，右边的表达式会被直接短路掉，不会进行运算。很多时候我们可能都需要用&&而不是&，例如在验证用户登录时判定用户名不是 null 而且不是空字符串，应当写为：username != null &&!username.equals(“”)， 二者的顺序不能交换，更不能用&运算符，因为第一个条件如果不成立，根本不能进行字符串的 equals 比较，否则会生 NullPointerException 异常。注意： 逻辑或运算符（|）和短路或运算符（||）的差别也是如此。

## 9、解释内存中的栈(stack)、堆(heap)和方法区(method area)的用法。

通常我们定义一个基本数据类型的变量，一个对象的引用，还有就是函数调用的现场保存都使用 JVM 中的栈空间；而通过 new 关键字和构造器创建的对象则放在堆空间，堆是垃圾收集器管理的主要区域，由于现在的垃圾收集器都采用分代收集算法，所以堆空间还可以细分为新生代和老生代，再具体一点可以分为Eden、Survivor（又可分为 From Survivor 和 To Survivor）、Tenured；方法区和堆都是各个线程共享的内存区域，用于存储已经被 JVM 加载的类信息、常量、静态变量、JIT 编译器编译后的代码等数据；程序中的字面量（literal） 如直接书写的 100、”hello”和常量都是放在常量池中，常量池是方法区的一部分，。栈空间操作起来最快但是栈很小，通常大量的对象都是放在堆空间，栈和堆的大小都可以通过 JVM 的启动参数来进行调整，栈空间用光了会引发StackOverflowError，而堆和常量池空间不足则会引发 OutOfMemoryError。

String str = new String("hello");复制代码

上面的语句中变量 str 放在栈上，用 new 创建出来的字符串对象放在堆上， 而”hello”这个字面量是放在方法区的。

补充 1：较新版本的 Java（从 Java 6 的某个更新开始）中，由于 JIT 编译器的发展和”逃逸分析”技术的逐渐成熟，栈上分配、标量替换等优化技术使得对象一定分配在堆上这件事情已经变得不那么绝对了。

补充 2：运行时常量池相当于 Class 文件常量池具有动态性，Java 语言并不要求常量一定只有编译期间才能产生，运行期间也可以将新的常量放入池中， String 类的 intern()方法就是这样的。

看看下面代码的执行结果是什么并且比较一下 Java 7 以前和以后的运行结果是否一致。

String s1 = new StringBuilder("go")

.append("od").toString(); System.out.println(s1.intern() == s1); String s2 = new StringBuilder("ja")

.append("va").toString();

System.out.println(s2.intern() == s2);复制代码

## 10、Math.round(11.5) 等于多少？ Math.round(-11.5)等于多少？

Math.round(11.5)的返回值是 12，Math.round(-11.5)的返回值是-11。四舍五入的原理是在参数上加 0.5 然后进行下取整。

## 11、switch 是否能作用在 byte 上，是否能作用在long 上，是否能作用在 String 上？

在 Java 5 以前，switch(expr)中，expr 只能是 byte、short、char、int。从Java5 开始，Java 中引入了枚举类型，expr 也可以是 enum 类型，从 Java 7 开始，expr 还可以是字符串（String），但是长整型（long）在目前所有的版本中都是不可以的。

## 12、用最有效率的方法计算 2 乘以 8？

2 << 3（左移 3 位相当于乘以 2 的 3 次方，右移 3 位相当于除以 2 的 3 次方）。

**补充：**我们为编写的类重写 hashCode 方法时，可能会看到如下所示的代码，其实我们不太理解为什么要使用这样的乘法运算来产生哈希码（散列码），而且为什么这个数是个素数，为什么通常选择 31 这个数？前两个问题的答案你可以自

己百度一下，选择 31 是因为可以用移位和减法运算来代替乘法，从而得到更好

的性能。说到这里你可能已经想到了：31 \* num 等价于(num << 5) - num，左移 5 位相当于乘以 2 的 5 次方再减去自身就相当于乘以 31，现在的 VM 都能自动完成这个优化。

public class PhoneNumber { private int areaCode; private String prefix; private String lineNumber; @Override

public int hashCode() {

final int prime = 31; int result = 1;

result = prime \* result + areaCode; result = prime \* result

+ ((lineNumber == null) ? 0 : lineNumber.hashCode()); result = prime \* result + ((prefix == null) ? 0 :

prefix.hashCode());

return result;

}

@Override

public Boolean equals(Object obj) {

if (this == obj)

return true;

if (obj == null)

return false;

if (getClass() != obj.getClass())

return false;

PhoneNumber other = (PhoneNumber) obj;

if (areaCode != other.areaCode)

return false;

if (lineNumber == null) {

if (other.lineNumber != null)

return false;

} else if (!lineNumber.equals(other.lineNumber))

return false;

if (prefix == null) {

if (other.prefix != null)

return false;

} else if (!prefix.equals(other.prefix))

return false; return true;

}

}复制代码

## 13、数组有没有 length()方法？String 有没有length()方法？

数组没有 length()方法 ，有 length 的属性。String 有 length()方法。JavaScript 中，获得字符串的长度是通过 length 属性得到的，这一点容易和Java 混淆。

## 14、在 Java 中，如何跳出当前的多重嵌套循环？

在最外层循环前加一个标记如 A，然后用 break A;可以跳出多重循环。（Java 中支持带标签的 break 和 continue 语句，作用有点类似于 C 和 C++中的 goto 语句，但是就像要避免使用 goto 一样，应该避免使用带标签的 break 和continue，因为它不会让你的程序变得更优雅，很多时候甚至有相反的作用，所以这种语法其实不知道更好）

## 15、构造器（constructor）是否可被重写

**（override）？**

构造器不能被继承，因此不能被重写，但可以被重载。

## 16、两个对象值相同(x.equals(y) == true)，但却可有不同的 hash code，这句话对不对？

不对，如果两个对象 x 和 y 满足 x.equals(y) == true，它们的哈希码（hash code）应当相同。Java 对于 eqauls 方法和 hashCode 方法是这样规定的： (1)如果两个对象相同（equals 方法返回 true），那么它们的 hashCode 值一定要相同；

(2)如果两个对象的 hashCode 相同，它们并不一定相同。当然，你未必要按照要求去做，但是如果你违背了上述原则就会发现在使用容器时，相同的对象可以出现在 Set 集合中，同时增加新元素的效率会大大下降（对于使用哈希存储的系统，如果哈希码频繁的冲突将会造成存取性能急剧下降）。

**补充：**关于 equals 和 hashCode 方法，很多 Java 程序都知道，但很多人也就是仅仅知道而已，在 Joshua Bloch 的大作《Effective Java》（很多软件公司，

《Effective Java》、《Java 编程思想》以及《重构：改善既有代码质量》是 Java 程序员必看书籍，如果你还没看过，那就赶紧去亚马逊买一本吧）中是这样介绍

equals 方法的：首先 equals 方法必须满足自反性（x.equals(x)必须返回true）、

对称性（x.equals(y)返回 true 时，y.equals(x)也必须返回 true）、传递性

（x.equals(y)和 y.equals(z)都返回 true 时，x.equals(z)也必须返回 true） 和一致性（当 x 和 y 引用的对象信息没有被修改时，多次调用 x.equals(y) 应该得到同样的返回值），而且对于任何非 null 值的引用 x，x.equals(null) 必须返回 false。

实现高质量的 equals 方法的诀窍包括：

1. 使用==操作符检查”参数是否为这个对象的引用”；
2. 使用 instanceof 操作符检查”参数是否为正确的类型”；
3. 对于类中的关键属性，检查参数传入对象的属性是否与之相匹配；
4. 编写完 equals 方法后，问自己它是否满足对称性、传递性、一致性；
5. 重写 equals 时总是要重写 hashCode；
6. 不要将 equals 方法参数中的 Object 对象替换为其他的类型，在重写时不要忘掉@Override 注解。

## 17、是否可以继承 String 类？

String 类是 final 类，不可以被继承。

补充：继承 String 本身就是一个错误的行为，对 String 类型最好的重用方式是关联关系（Has-A）和依赖关系（Use-A）而不是继承关系（Is-A）。

## 18、当一个对象被当作参数传递到一个方法后，此方法可改变这个对象的属性，并可返回变化后的结果， 那么这里到底是值传递还是引用传递？

是值传递。Java 语言的方法调用只支持参数的值传递。当一个对象实例作为一个参数被传递到方法中时，参数的值就是对该对象的引用。对象的属性可以在被调用过程中被改变，但对对象引用的改变是不会影响到调用者的。C++和 C#中可以通过传引用或传输出参数来改变传入的参数的值。在 C#中可以编写如下所示的代码，但是在 Java 中却做不到。

using System; namespace CS01 {

class Program {

public static void swap(ref int x, ref int y) { int temp = x;

x = y;

y = temp;

}

public static void Main (string[] args) { int a = 5, b = 10;

}

}

}复制代码

swap (ref a, ref b);

// a = 10, b = 5;

第 225 页 共 485 页

Console.WriteLine ("a = {0}, b = {1}", a, b);

**说明：**Java 中没有传引用实在是非常的不方便，这一点在 Java 8 中仍然没有得到改进，正是如此在 Java 编写的代码中才会出现大量的 Wrapper 类（将需要通过方法调用修改的引用置于一个 Wrapper 类中，再将 Wrapper 对象传入方法），这样的做法只会让代码变得臃肿，尤其是让从 C 和 C++转型为 Java 程序员的开发者无法容忍。

## 19、String 和 StringBuilder、StringBuffer 的区别？

Java 平台提供了两种类型的字符串：String 和 StringBuffer/StringBuilder， 它们可以储存和操作字符串。其中 String 是只读字符串，也就意味着 String 引用的字符串内容是不能被改变的。而 StringBuffer/StringBuilder 类表示的字符串对象可以直接进行修改。StringBuilder 是 Java 5 中引入的，它和StringBuffer 的方法完全相同，区别在于它是在单线程环境下使用的，因为它的所有方面都没有被 synchronized 修饰，因此它的效率也比 StringBuffer 要高。

面试题 1 - 什么情况下用+运算符进行字符串连接比调用StringBuffer/StringBuilder 对象的 append 方法连接字符串性能更好？ 面试题 2 - 请说出下面程序的输出。

class StringEqualTest {

public static void main(String[] args) { String s1 = "Programming";

String s2 = new String("Programming");

String s3 = "Program"; String s4 = "ming";

String s5 = "Program" + "ming"; String s6 = s3 + s4;

System.out.println(s1 == s2); System.out.println(s1 == s5); System.out.println(s1 == s6); System.out.println(s1 == s6.intern()); System.out.println(s2 == s2.intern());

}

}复制代码

**补充：**解答上面的面试题需要清除两点：

1. String 对象的 intern 方法会得到字符串对象在常量池中对应的版本的引用（如果常量池中有一个字符串与 String 对象的 equals 结果是 true），如果常量池中没有对应的字符串，则该字符串将被添加到常量池中，然后返回常量池中字符串的引用；
2. 字符串的+操作其本质是创建了 StringBuilder 对象进行 append 操作，然后将拼接后的 StringBuilder 对象用 toString 方法处理成 String 对象，这一点可以用 javap -c StringEqualTest.class 命令获得 class 文件对应的JVM 字节码指令就可以看出来。

## 20、重载（Overload）和重写（Override）的区别。重载的方法能否根据返回类型进行区分？

方法的重载和重写都是实现多态的方式，区别在于前者实现的是编译时的多态 性，而后者实现的是运行时的多态性。重载发生在一个类中，同名的方法如果有不同的参数列表（参数类型不同、参数个数不同或者二者都不同）则视为重载； 重写发生在子类与父类之间，重写要求子类被重写方法与父类被重写方法有相同的返回类型，比父类被重写方法更好访问，不能比父类被重写方法声明更多的异常（里氏代换原则）。重载对返回类型没有特殊的要求。

## 21、描述一下 JVM 加载 class 文件的原理机制？

JVM 中类的装载是由类加载器（ClassLoader）和它的子类来实现的，Java 中的类加载器是一个重要的 Java 运行时系统组件，它负责在运行时查找和装入类文件中的类。

由于 Java 的跨平台性，经过编译的 Java 源程序并不是一个可执行程序，而是一个或多个类文件。当 Java 程序需要使用某个类时，JVM 会确保这个类已经被加载、连接（验证、准备和解析）和初始化。类的加载是指把类的.class 文件中的数据读入到内存中，通常是创建一个字节数组读入.class 文件，然后产生与所加载类对应的 Class 对象。加载完成后，Class 对象还不完整，所以此时的类还不可用。当类被加载后就进入连接阶段，这一阶段包括验证、准备（为静态变量分配内存并设置默认的初始值）和解析（将符号引用替换为直接引用）三个步骤。最后 JVM 对类进行初始化，包括：1)如果类存在直接的父类并且这个类还没有被初始化，那么就先初始化父类；2)如果类中存在初始化语句，就依次执行这些初始化语句。

类的加载是由类加载器完成的，类加载器包括：根加载器（BootStrap）、扩展加载器（Extension）、系统加载器（System）和用户自定义类加载器

（java.lang.ClassLoader 的子类）。从 Java 2（JDK 1.2）开始，从 Java 2

（JDK 1.2）开始，类加载过程采取了父亲委托机制（PDM）。PDM 更好的保证了Java 平台的安全性，在该机制中，JVM 自带的 Bootstrap 是根加载器，其他的

加载器都有且仅有一个父类加载器。类的加载首先请求父类加载器加载，父类加载器无能为力时才由其子类加载器自行加载。JVM 不会向 Java 程序提供对Bootstrap 的引用。下面是关于几个类加载器的说明：

1. Bootstrap：一般用本地代码实现，负责加载 JVM 基础核心类库（rt.jar）；
2. Extension：从 java.ext.dirs 系统属性所指定的目录中加载类库，它的父加载器是 Bootstrap；
3. System：又叫应用类加载器，其父类是 Extension。它是应用最广泛的类加载器。它从环境变量 classpath 或者系统属性 java.class.path 所指定的目录中记载类，是用户自定义加载器的默认父加载器。

## 22、char 型变量中能不能存贮一个中文汉字，为什么？

char 类型可以存储一个中文汉字，因为 Java 中使用的编码是 Unicode（不选择任何特定的编码，直接使用字符在字符集中的编号，这是统一的唯一方法）， 一个 char 类型占 2 个字节（16 比特），所以放一个中文是没问题的。

补充：使用 Unicode 意味着字符在 JVM 内部和外部有不同的表现形式，在 JVM 内部都是 Unicode，当这个字符被从 JVM 内部转移到外部时（例如存入文件系统中），需要进行编码转换。所以 Java 中有字节流和字符流，以及在字符流和字节流之间进行转换的转换流，如 InputStreamReader 和OutputStreamReader，这两个类是字节流和字符流之间的适配器类，承担了编码转换的任务；对于 C 程序员来说，要完成这样的编码转换恐怕要依赖于 union

（联合体/共用体）共享内存的特征来实现了。

## 23、抽象类（abstract class）和接口（interface） 有什么异同？

抽象类和接口都不能够实例化，但可以定义抽象类和接口类型的引用。一个类如果继承了某个抽象类或者实现了某个接口都需要对其中的抽象方法全部进行实 现，否则该类仍然需要被声明为抽象类。接口比抽象类更加抽象，因为抽象类中可以定义构造器，可以有抽象方法和具体方法，而接口中不能定义构造器而且其中的方法全部都是抽象方法。抽象类中的成员可以是 private、默认、protected、public 的，而接口中的成员全都是 public 的。抽象类中可以定义成员变量， 而接口中定义的成员变量实际上都是常量。有抽象方法的类必须被声明为抽象 类，而抽象类未必要有抽象方法。

## 24、静态嵌套类(Static Nested Class)和内部类

**（Inner Class）的不同？**

Static Nested Class 是被声明为静态（static）的内部类，它可以不依赖于外部类实例被实例化。而通常的内部类需要在外部类实例化后才能实例化，其语法看起来挺诡异的，如下所示。

/\*\*

* 扑克类（一副扑克）
* @author 骆 昊

\*

\*/

public class Poker {

private static String[] suites = {"黑桃", "红桃", "草花", "方

块"};

private static int[] faces = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11,

12, 13};

private Card[] cards;

/\*\*

* 构造器

\*

\*/

public Poker() {

cards = new Card[52];

for (int i = 0; i < suites.length; i++) {

for (int j = 0; j < faces.length; j++) { cards[i \* 13 + j] = new Card(suites[i],

faces[j]);

}

}

}

/\*\*

* 洗牌 （随机乱序）

\*

\*/

public void shuffle() {

for (int i = 0, len = cards.length; i < len; i++) { int index = (int) (Math.random() \* len);

Card temp = cards[index]; cards[index] = cards[i]; cards[i] = temp;

}

}

/\*\*

* 发牌
* @param index 发牌的位置

\*

\*/

public Card deal(int index) {

return cards[index];

}

/\*\*

* 卡片类（一张扑克）
* [内部类]
* @author 骆 昊

\*

\*/

public class Card {

private String suite;

// 花 色

private int face;

// 点 数

public Card(String suite, int face) { this.suite = suite;

this.face = face;

}

@Override

public String toString() { String faceStr = ""; switch(face) {

case 1: faceStr = "A"; break;

case 11: faceStr = "J"; break;

case 12: faceStr = "Q"; break;

case 13: faceStr = "K"; break;

default: faceStr = String.valueOf(face);

}

return suite + faceStr;

}

}

}

测试代码：

class PokerTest {

public static void main(String[] args) { Poker poker = new Poker(); poker.shuffle();

// 洗 牌

Poker.Card c1 = poker.deal(0);

// 发第一张牌

// 对于非静态内部类 Card

}

}复制代码

// 只有通过其外部类 Poker 对象才能创建 Card 对象

Poker.Card c2 = poker.new Card("红心", 1);

// 自己创建一张牌

System.out.println(c1);

// 洗牌后的第一张

System.out.println(c2);

// 打印: 红心 A

面试题 - 下面的代码哪些地方会产生编译错误？

class Outer {

class Inner {

}

public static void foo() { new Inner();

}

public void bar() {

new Inner();

}

public static void main(String[] args) { new Inner();

}

}复制代码

**注意：**Java 中非静态内部类对象的创建要依赖其外部类对象，上面的面试题中foo 和 main 方法都是静态方法，静态方法中没有 this，也就是说没有所谓的外部类对象，因此无法创建内部类对象，如果要在静态方法中创建内部类对象， 可以这样做：

new Outer().new Inner();复制代码

## 25、Java 中会存在内存泄漏吗，请简单描述。

理论上 Java 因为有垃圾回收机制（GC）不会存在内存泄露问题（这也是 Java 被广泛使用于服务器端编程的一个重要原因）；然而在实际开发中，可能会存在无用但可达的对象，这些对象不能被 GC 回收，因此也会导致内存泄露的发生。例如 Hibernate 的 Session（一级缓存）中的对象属于持久态，垃圾回收器是不会回收这些对象的，然而这些对象中可能存在无用的垃圾对象，如果不及时关闭

（close）或清空（flush）一级缓存就可能导致内存泄露。下面例子中的代码也会导致内存泄露。

import java.util.Arrays;

import java.util.EmptyStackException; public class MyStack<T> {

private T[] elements; private int size = 0;

private static final int INIT\_CAPACITY = 16; public MyStack() {

elements = (T[]) new Object[INIT\_CAPACITY];

}

public void push(T elem) { ensureCapacity(); elements[size++] = elem;

}

public T pop() {

if(size == 0)

throw new EmptyStackException();

return elements[--size];

}

private void ensureCapacity() {

if(elements.length == size) {

elements = Arrays.copyOf(elements, 2 \* size + 1);

}

}

}复制代码

上面的代码实现了一个栈（先进后出（FILO））结构，乍看之下似乎没有什么明显的问题，它甚至可以通过你编写的各种单元测试。然而其中的 pop 方法却存在内存泄露的问题，当我们用 pop 方法弹出栈中的对象时，该对象不会被当作垃圾回收，即使使用栈的程序不再引用这些对象，因为栈内部维护着对这些对象

的过期引 用（obsolete reference）。在支持垃圾回收的语言中，内存泄露是很隐蔽的，这种内存泄露其实就是无意识的对象保持。如果一个对象引用被无意识的保留起来了，那么垃圾回收器不会处理这个对象，也不会处理该对象引用的其他对象，即使这样的对象只有少数几个，也可能会导致很多的对象被排除在垃圾回收之外，从而对性能造成重大影响，极端情况下会引发 Disk Paging（物理内存与硬盘的虚拟内存交换数据），甚至造成 OutOfMemoryError。

## 26、抽象的（abstract）方法是否可同时是静态的

**（static）,是否可同时是本地方法（native），是否可同时被 synchronized 修饰？**

都不能。抽象方法需要子类重写，而静态的方法是无法被重写的，因此二者是矛盾的。本地方法是由本地代码（如 C 代码）实现的方法，而抽象方法是没有实现的，也是矛盾的。synchronized 和方法的实现细节有关，抽象方法不涉及实现细节，因此也是相互矛盾的。

## 27、阐述静态变量和实例变量的区别。

静态变量是被 static 修饰符修饰的变量，也称为类变量，它属于类，不属于类的任何一个对象，一个类不管创建多少个对象，静态变量在内存中有且仅有一个拷贝；实例变量必须依存于某一实例，需要先创建对象然后通过对象才能访问到它。静态变量可以实现让多个对象共享内存。

补充：在 Java 开发中，上下文类和工具类中通常会有大量的静态成员。

## 28、是否可以从一个静态（static）方法内部发出对非静态（non-static）方法的调用？

不可以，静态方法只能访问静态成员，因为非静态方法的调用要先创建对象，在调用静态方法时可能对象并没有被初始化。

## 29、如何实现对象克隆？

有两种方式：

1. 实现 Cloneable 接口并重写 Object 类中的 clone()方法；
2. 实现 Serializable 接口，通过对象的序列化和反序列化实现克隆，可以实现真正的深度克隆，代码如下。

import java.io.ByteArrayInputStream; import java.io.ByteArrayOutputStream; import java.io.ObjectInputStream; import java.io.ObjectOutputStream; import java.io.Serializable;

public class MyUtil {

private MyUtil() {

throw new AssertionError();

}

@SuppressWarnings("unchecked")

public static <T extends Serializable> T clone(T obj) throws Exception {

ByteArrayOutputStream bout = new ByteArrayOutputStream();

ObjectOutputStream oos = new ObjectOutputStream(bout); oos.writeObject(obj);

ByteArrayInputStream bin = new ByteArrayInputStream(bout.toByteArray()); ObjectInputStream ois = new ObjectInputStream(bin);

return (T) ois.readObject();

// 说明：调用 ByteArrayInputStream 或

ByteArrayOutputStream

对象的 close 方法没有任何意义

// 这两个基于内存的流只要垃圾回收器清理对象就能够释放资源，这

一点不同于对外部资源（如文件流）的释放

}

}复制代码

下面是测试代码：

import java.io.Serializable;

/\*\*

* 人类
* @author 骆 昊

\*

\*/

class Person implements Serializable {

private static final long serialVersionUID =

-9102017020286042305L;

private String name;

// 姓 名

private int age;

// 年 龄

private Car car;

// 座 驾

public Person(String name, int age, Car car) { this.name = name;

this.age = age; this.car = car;

}

public String getName() {

return name;

}

public void setName(String name) { this.name = name;

}

public int getAge() {

return age;

}

public void setAge(int age) { this.age = age;

}

public Car getCar() {

return car;

}

public void setCar(Car car) { this.car = car;

}

@Override

public String toString() {

return "Person [name=" + name + ", age=" + age + ", car="

+

car + "]";

}

}

/\*\*

* 小汽车类
* @author 骆 昊

\*

\*/

class Car implements Serializable {

private static final long serialVersionUID =

-5713945027627603702L;

private String brand;

// 品 牌

private int maxSpeed;

// 最高时速

public Car(String brand, int maxSpeed) { this.brand = brand; this.maxSpeed = maxSpeed;

}

public String getBrand() {

return brand;

}

public void setBrand(String brand) { this.brand = brand;

}

public int getMaxSpeed() {

return maxSpeed;

}

public void setMaxSpeed(int maxSpeed) { this.maxSpeed = maxSpeed;

}

@Override

public String toString() {

return "Car [brand=" + brand + ", maxSpeed=" + maxSpeed

+

"]";

}

}

class CloneTest {

public static void main(String[] args) { try {

Car("Benz",

Person p1 = new Person("Hao LUO", 33, new

300));

Person p2 = MyUtil.clone(p1);

// 深度克隆

p2.getCar().setBrand("BYD");

// 修改克隆的 Person 对象 p2 关联的汽车对象的品牌属性

// 原来的 Person 对象 p1 关联的汽车不会受到任何影响

// 因为在克隆 Person 对象时其关联的汽车对象也被克隆了

System.out.println(p1);

}

catch (Exception e) {

e.printStackTrace();

}

}

}复制代码

**注意：**基于序列化和反序列化实现的克隆不仅仅是深度克隆，更重要的是通过泛型限定，可以检查出要克隆的对象是否支持序列化，这项检查是编译器完成的， 不是在运行时抛出异常，这种是方案明显优于使用 Object 类的 clone 方法克隆对象。让问题在编译的时候暴露出来总是好过把问题留到运行时。

## 30、GC 是什么？为什么要有 GC？

GC 是垃圾收集的意思，内存处理是编程人员容易出现问题的地方，忘记或者错误的内存回收会导致程序或系统的不稳定甚至崩溃，Java 提供的 GC 功能可以自动监测对象是否超过作用域从而达到自动回收内存的目的，Java 语言没有提供释放已分配内存的显示操作方法。Java 程序员不用担心内存管理，因为垃圾收集器会自动进行管理。要请求垃圾收集，可以调用下面的方法之一：System.gc()或 Runtime.getRuntime().gc() ，但 JVM 可以屏蔽掉显示的垃圾回收调用。垃圾回收可以有效的防止内存泄露，有效的使用可以使用的内存。垃圾回收器通常是作为一个单独的低优先级的线程运行，不可预知的情况下对内存堆中已经死亡的或者长时间没有使用的对象进行清除和回收，程序员不能实时的调用垃圾回收器对某个对象或所有对象进行垃圾回收。在 Java 诞生初期，垃圾回收是 Java 最大的亮点之一，因为服务器端的编程需要有效的防止内存泄露问题，然而时过境迁，如今 Java 的垃圾回收机制已经成为被诟病的东西。移动智能终端用户通常觉得 iOS 的系统比 Android 系统有更好的用户体验，其中一个深层次的原因就在于 Android 系统中垃圾回收的不可预知性。

补充：垃圾回收机制有很多种，包括：分代复制垃圾回收、标记垃圾回收、增量垃圾回收等方式。标准的 Java 进程既有栈又有堆。栈保存了原始型局部变量， 堆保存了要创建的对象。Java 平台对堆内存回收和再利用的基本算法被称为标记和清除，但是 Java 对其进行了改进，采用“分代式垃圾收集”。这种方法会跟 Java 对象的生命周期将堆内存划分为不同的区域，在垃圾收集过程中，可能会将对象移动到不同区域：

1. 伊甸园（Eden）：这是对象最初诞生的区域，并且对大多数对象来说，这里是它们唯一存在过的区域。
2. 幸存者乐园（Survivor）：从伊甸园幸存下来的对象会被挪到这里。
3. 终身颐养园（Tenured）：这是足够老的幸存对象的归宿。年轻代收集

（Minor-GC）过程是不会触及这个地方的。当年轻代收集不能把对象放进终身颐

养园时，就会触发一次完全收集（Major-GC），这里可能还会牵扯到压缩，以便为大对象腾出足够的空间。

与垃圾回收相关的 JVM 参数：

-Xms / -Xmx — 堆的初始大小 / 堆的最大大小

-Xmn — 堆中年轻代的大小

-XX:-DisableExplicitGC — 让 System.gc()不产生任何作用

-XX:+PrintGCDetails — 打印 GC 的细节

-XX:+PrintGCDateStamps — 打印 GC 操作的时间戳

-XX:NewSize / XX:MaxNewSize — 设置新生代大小/新生代最大大小

-XX:NewRatio — 可以设置老生代和新生代的比例

-XX:PrintTenuringDistribution — 设置每次新生代 GC 后输出幸存者乐园中对象年龄的分布

-XX:InitialTenuringThreshold / -XX:MaxTenuringThreshold：设置老年代阀值的初始值和最大值

-XX:TargetSurvivorRatio：设置幸存区的目标使用率 复制代码

## 31、String s = new String(“xyz”);创建了几个字符串对象？

两个对象，一个是静态区的”xyz”，一个是用 new 创建在堆上的对象。

## 32、接口是否可继承（extends）接口？抽象类是否可实现（implements）接口？抽象类是否可继承具体类（concreteclass）？

接口可以继承接口，而且支持多重继承。抽象类可以实现(implements)接口，抽象类可继承具体类也可以继承抽象类。

## 33、一个”.java”源文件中是否可以包含多个类（不 是内部类）？有什么限制？

可以，但一个源文件中最多只能有一个公开类（public class）而且文件名必须和公开类的类名完全保持一致。

## 34、Anonymous Inner Class(匿名内部类)是否可以继承其它类？是否可以实现接口？

可以继承其他类或实现其他接口，在 Swing 编程和 Android 开发中常用此方式来实现事件监听和回调。

## 35、内部类可以引用它的包含类（外部类）的成员吗？ 有没有什么限制？

一个内部类对象可以访问创建它的外部类对象的成员，包括私有成员。

## 36、Java 中的 final 关键字有哪些用法？

1. 修饰类：表示该类不能被继承；
2. 修饰方法：表示方法不能被重写；
3. 修饰变量：表示变量只能一次赋值以后值不能被修改（常量）。

## 37、指出下面程序的运行结果

class A {

static {

System.out.print("1");

}

public A() {

System.out.print("2");

}

}

class B extends A{

static {

System.out.print("a");

}

public B() {

System.out.print("b");

}

}

public class Hello {

public static void main(String[] args) { A ab = new B();

ab = new B();

}

}复制代码

**执行结果：**1a2b2b。创建对象时构造器的调用顺序是：先初始化静态成员，然后调用父类构造器，再初始化非静态成员，最后调用自身构造器。

**提示：**如果不能给出此题的正确答案，说明之前第 21 题 Java 类加载机制还没有完全理解，赶紧再看看吧。

## 38、数据类型之间的转换：

1. 如何将字符串转换为基本数据类型？
2. 如何将基本数据类型转换为字符串？ 答：
3. 调用基本数据类型对应的包装类中的方法 parseXXX(String)或valueOf(String)即可返回相应基本类型；
4. 一种方法是将基本数据类型与空字符串（”“）连接（+）即可获得其所对应的字符串；另一种方法是调用 String 类中的 valueOf()方法返回相应字符串

## 39、如何实现字符串的反转及替换？

方法很多，可以自己写实现也可以使用 String 或StringBuffer/StringBuilder 中的方法。有一道很常见的面试题是用递归实现字符串反转，代码如下所示：

public static String reverse(String originStr) { if(originStr == null || originStr.length() <= 1) return originStr;

return reverse(originStr.substring(1)) + originStr.charAt(0);

}复制代码

## 40、怎样将 GB2312 编码的字符串转换为ISO-8859-1 编码的字符串？

代码如下所示：

String s1 = "你好";

String s2 = new String(s1.getBytes("GB2312"), "ISO-8859-1");复制代码

## 41、日期和时间：

1. 如何取得年月日、小时分钟秒？
2. 如何取得从 1970 年 1 月 1 日 0 时 0 分 0 秒到现在的毫秒数？
3. 如何取得某月的最后一天？
4. 如何格式化日期？ 答：

**问题 1：**创建 java.util.Calendar 实例，调用其 get()方法传入不同的参数即可获得参数所对应的值。Java 8 中可以使用 java.time.LocalDateTimel 来获取，代码如下所示。

public class DateTimeTest {

public static void main(String[] args) { Calendar cal = Calendar.getInstance();

System.out.println(cal.get(Calendar.YEAR)); System.out.println(cal.get(Calendar.MONTH));

// 0 - 11

System.out.println(cal.get(Calendar.DATE)); System.out.println(cal.get(Calendar.HOUR\_OF\_DAY)); System.out.println(cal.get(Calendar.MINUTE)); System.out.println(cal.get(Calendar.SECOND));

// Java 8

LocalDateTime dt = LocalDateTime.now(); System.out.println(dt.getYear()); System.out.println(dt.getMonthValue());

// 1 - 12

System.out.println(dt.getDayOfMonth()); System.out.println(dt.getHour()); System.out.println(dt.getMinute()); System.out.println(dt.getSecond());

}

}复制代码

**问题 2：**以下方法均可获得该毫秒数。

Calendar.getInstance().getTimeInMillis(); System.currentTimeMillis(); Clock.systemDefaultZone().millis();

// Java 8 复制代码

**问题 3：**代码如下所示。

Calendar time = Calendar.getInstance(); time.getActualMaximum(Calendar.DAY\_OF\_MONTH 复制代码

**问题 4：**利用 java.text.DataFormat 的子类（如 SimpleDateFormat 类）中的format(Date)方法可将日期格式化。Java 8 中可以用java.time.format.DateTimeFormatter 来格式化时间日期，代码如下所示。import java.text.SimpleDateFormat;

import java.time.LocalDate;

import java.time.format.DateTimeFormatter; import java.util.Date;

class DateFormatTest {

public static void main(String[] args) { SimpleDateFormat oldFormatter = new SimpleDateFormat("yyyy/MM/dd");

Date date1 = new Date();

System.out.println(oldFormatter.format(date1));

// Java 8

DateTimeFormatter newFormatter = DateTimeFormatter.ofPattern("yyyy/MM/dd"); LocalDate date2 = LocalDate.now();

System.out.println(date2.format(newFormatter));

}

}复制代码

**补充：**Java 的时间日期 API 一直以来都是被诟病的东西，为了解决这一问题， Java8 中引入了新的时间日期 API，其中包括 LocalDate、LocalTime、LocalDateTime、Clock、Instant 等类，这些的类的设计都使用了不变模式，因此是线程安全的设计。

## 42、打印昨天的当前时刻。

import java.util.Calendar; class YesterdayCurrent {

public static void main(String[] args){ Calendar cal = Calendar.getInstance(); cal.add(Calendar.DATE, -1); System.out.println(cal.getTime());

}

}复制代码

在 Java 8 中，可以用下面的代码实现相同的功能。

import java.time.LocalDateTime; class YesterdayCurrent {

public static void main(String[] args) { LocalDateTime today = LocalDateTime.now(); LocalDateTime yesterday = today.minusDays(1); System.out.println(yesterday);

}

}复制代码

## 43、比较一下 Java 和 JavaSciprt。

JavaScript 与 Java 是两个公司开发的不同的两个产品。Java 是原SunMicrosystems 公司推出的面向对象的程序设计语言，特别适合于互联网应用程序开发；而 JavaScript 是 Netscape 公司的产品，为了扩展 Netscape 浏览器的功能而开发的一种可以嵌入 Web 页面中运行的基于对象和事件驱动的解释性语言。JavaScript 的前身是 LiveScript；而 Java 的前身是 Oak 语言。 下面对两种语言间的异同作如下比较：

1. 基于对象和面向对象：Java 是一种真正的面向对象的语言，即使是开发简单的程序，必须设计对象；JavaScript 是种脚本语言，它可以用来制作与网络无关的，与用户交互作用的复杂软件。它是一种基于对象（Object-Based）和事件驱动（Event-Driven）的编程语言，因而它本身提供了非常丰富的内部对象供设计人员使用。
2. 解释和编译：Java 的源代码在执行之前，必须经过编译。JavaScript 是一种解释性编程语言，其源代码不需经过编译，由浏览器解释执行。（目前的浏览器几乎都使用了 JIT（即时编译）技术来提升 JavaScript 的运行效率）
3. 强类型变量和类型弱变量：Java 采用强类型变量检查，即所有变量在编译之前必须作声明；JavaScript 中变量是弱类型的，甚至在使用变量前可以不作声明，JavaScript 的解释器在运行时检查推断其数据类型。
4. 代码格式不一样。

补充：上面列出的四点是网上流传的所谓的标准答案。其实 Java 和 JavaScript 最重要的区别是一个是静态语言，一个是动态语言。目前的编程语言的发展趋势是函数式语言和动态语言。在 Java 中类（class）是一等公民，而 JavaScript 中函数（function）是一等公民，因此 JavaScript 支持函数式编程，可以使用Lambda 函数和闭包（closure），当然 Java 8 也开始支持函数式编程，提供了对 Lambda 表达式以及函数式接口的支持。对于这类问题，在面试的时候最好还是用自己的语言回答会更加靠谱，不要背网上所谓的标准答案。

## 44、什么时候用断言（assert）？

断言在软件开发中是一种常用的调试方式，很多开发语言中都支持这种机制。一般来说，断言用于保证程序最基本、关键的正确性。断言检查通常在开发和测试时开启。为了保证程序的执行效率，在软件发布后断言检查通常是关闭的。断言是一个包含布尔表达式的语句，在执行这个语句时假定该表达式为 true；如果表达式的值为 false，那么系统会报告一个 AssertionError。断言的使用如下面的代码所示：

assert(a > 0); // throws an AssertionError if a <= 0 复制代码断言可以有两种形式：

assert Expression1;

assert Expression1 : Expression2 ;

Expression1 应该总是产生一个布尔值。

Expression2 可以是得出一个值的任意表达式；这个值用于生成显示更多调试信息的字符串消息。

要在运行时启用断言，可以在启动 JVM 时使用-enableassertions 或者-ea 标记。要在运行时选择禁用断言，可以在启动 JVM 时使用-da 或者

-disableassertions 标记。要在系统类中启用或禁用断言，可使用-esa 或-dsa 标记。还可以在包的基础上启用或者禁用断言。

注意：断言不应该以任何方式改变程序的状态。简单的说，如果希望在不满足某些条件时阻止代码的执行，就可以考虑用断言来阻止它。

## 45、Error 和 Exception 有什么区别？

Error 表示系统级的错误和程序不必处理的异常，是恢复不是不可能但很困难的情况下的一种严重问题；比如内存溢出，不可能指望程序能处理这样的情况； Exception 表示需要捕捉或者需要程序进行处理的异常，是一种设计或实现问题；也就是说，它表示如果程序运行正常，从不会发生的情况。

## 46、try{}里有一个 return 语句，那么紧跟在这个try 后的finally{}里的代码会不会被执行，什么时候被执行，在 return 前还是后?

会执行，在方法返回调用者前执行。

**注意：**在 finally 中改变返回值的做法是不好的，因为如果存在 finally 代码块，try 中的 return 语句不会立马返回调用者，而是记录下返回值待 finally 代码块执行完毕之后再向调用者返回其值，然后如果在 finally 中修改了返回值，就会返回修改后的值。显然，在 finally 中返回或者修改返回值会对程序造成很大的困扰，C#中直接用编译错误的方式来阻止程序员干这种龌龊的事情， Java 中也可以通过提升编译器的语法检查级别来产生警告或错误，Eclipse 中可以在如图所示的地方进行设置，强烈建议将此项设置为编译错误。

## 47、Java 语言如何进行异常处理，关键字：throws、throw、try、catch、finally 分别如何使用？

Java 通过面向对象的方法进行异常处理，把各种不同的异常进行分类，并提供了良好的接口。在 Java 中，每个异常都是一个对象，它是 Throwable 类或其子类的实例。当一个方法出现异常后便抛出一个异常对象，该对象中包含有异常信息，调用这个对象的方法可以捕获到这个异常并可以对其进行处理。Java 的

异常处理是通过 5 个关键词来实现的：try、catch、throw、throws 和 finally。一般情况下是用 try 来执行一段程序，如果系统会抛出（throw）一个异常对象， 可以通过它的类型来捕获（catch）它，或通过总是执行代码块（finally）来处理；try 用来指定一块预防所有异常的程序；catch 子句紧跟在 try 块后面， 用来指定你想要捕获的异常的类型；throw 语句用来明确地抛出一个异常； throws 用来声明一个方法可能抛出的各种异常（当然声明异常时允许无病呻

吟）；finally 为确保一段代码不管发生什么异常状况都要被执行；try 语句可以嵌套，每当遇到一个 try 语句，异常的结构就会被放入异常栈中，直到所有的 try 语句都完成。如果下一级的 try 语句没有对某种异常进行处理，异常栈就会执行出栈操作，直到遇到有处理这种异常的 try 语句或者最终将异常抛给JVM。

## 48、运行时异常与受检异常有何异同？

异常表示程序运行过程中可能出现的非正常状态，运行时异常表示虚拟机的通常操作中可能遇到的异常，是一种常见运行错误，只要程序设计得没有问题通常就不会发生。受检异常跟程序运行的上下文环境有关，即使程序设计无误，仍然可能因使用的问题而引发。Java 编译器要求方法必须声明抛出可能发生的受检异常，但是并不要求必须声明抛出未被捕获的运行时异常。异常和继承一样，是面向对象程序设计中经常被滥用的东西，在 Effective Java 中对异常的使用给出了以下指导原则：

1. 不要将异常处理用于正常的控制流（设计良好的 API 不应该强迫它的调用者为了正常的控制流而使用异常）
2. 对可以恢复的情况使用受检异常，对编程错误使用运行时异常
3. 避免不必要的使用受检异常（可以通过一些状态检测手段来避免异常的发生）
4. 优先使用标准的异常
5. 每个方法抛出的异常都要有文档
6. 保持异常的原子性
7. 不要在 catch 中忽略掉捕获到的异常

## 49、列出一些你常见的运行时异常？

1. ArithmeticException（算术异常）
2. ClassCastException （类转换异常）
3. IllegalArgumentException （非法参数异常）
4. IndexOutOfBoundsException （下标越界异常）
5. NullPointerException （空指针异常）
6. SecurityException （安全异常）

## 50、阐述 final、finally、finalize 的区别。

1. final：修饰符（关键字）有三种用法：如果一个类被声明为 final，意味着它不能再派生出新的子类，即不能被继承，因此它和 abstract 是反义词。将变量声明为 final，可以保证它们在使用中不被改变，被声明为 final 的变量必须在声明时给定初值，而在以后的引用中只能读取不可修改。被声明为final 的方法也同样只能使用，不能在子类中被重写。
2. finally：通常放在 try…catch…的后面构造总是执行代码块，这就意味着程序无论正常执行还是发生异常，这里的代码只要 JVM 不关闭都能执行，可以将释放外部资源的代码写在 finally 块中.
3. finalize：Object 类中定义的方法，Java 中允许使用 finalize()方法在垃圾收集器将对象从内存中清除出去之前做必要的清理工作。这个方法是由垃圾收集器在销毁对象时调用的，通过重写 finalize()方法可以整理系统资源或者执行其他清理工作。

## 51、类 ExampleA 继承 Exception，类 ExampleB 继承 ExampleA。

有如下代码片断：

try {

throw new ExampleB("b")

}

catch（ExampleA e）{

System.out.println("ExampleA");

}

catch（Exception e）{

System.out.println("Exception");

}复制代码

\*\*请问执行此段代码的输出是什么？ **答：**

输出：ExampleA。（根据里氏代换原则[能使用父类型的地方一定能使用子类型]， 抓取 ExampleA 类型异常的 catch 块能够抓住 try 块中抛出的 ExampleB 类型的异常）

面试题 - 说出下面代码的运行结果。（此题的出处是《Java 编程思想》一书）

class Annoyance extends Exception {

}

class Sneeze extends Annoyance {

}

class Human {

public static void main(String[] args) throws Exception {

try {

try {

}

throw new Sneeze();

catch ( Annoyance a ) {

System.out.println("Caught Annoyance"); throw a;

}

}

catch ( Sneeze s ) {

System.out.println("Caught Sneeze"); return ;

}

finally {

System.out.println("Hello World!");

}

}

}复制代码

## 52、List、Set、Map 是否继承自 Collection 接口？

List、Set 是 ，Map 不是。Map 是键值对映射容器，与 List 和 Set 有明显的区别，而 Set 存储的零散的元素且不允许有重复元素（数学中的集合也是如此）， List 是线性结构的容器，适用于按数值索引访问元素的情形。

## 53、阐述 ArrayList、Vector、LinkedList 的存储性能和特性。

ArrayList 和 Vector 都是使用数组方式存储数据，此数组元素数大于实际存储的数据以便增加和插入元素，它们都允许直接按序号索引元素，但是插入元素要涉及数组元素移动等内存操作，所以索引数据快而插入数据慢，Vector 中的方法由于添加了 synchronized 修饰，因此 Vector 是线程安全的容器，但性能上较 ArrayList 差，因此已经是 Java 中的遗留容器。LinkedList 使用双向链表实现存储（将内存中零散的内存单元通过附加的引用关联起来，形成一个可以按序号索引的线性结构，这种链式存储方式与数组的连续存储方式相比，内存的利用率更高），按序号索引数据需要进行前向或后向遍历，但是插入数据时只需要记录本项的前后项即可，所以插入速度较快。Vector 属于遗留容器（Java 早期的版本中提供的容器，除此之外，Hashtable、Dictionary、BitSet、Stack、Properties 都是遗留容器），已经不推荐使用，但是由于 ArrayList 和

LinkedListed 都是非线程安全的，如果遇到多个线程操作同一个容器的场景， 则可以通过工具类 Collections 中的 synchronizedList 方法将其转换成线程安全的容器后再使用（这是对装潢模式的应用，将已有对象传入另一个类的构造器中创建新的对象来增强实现）。

**补充：**遗留容器中的 Properties 类和 Stack 类在设计上有严重的问题， Properties 是一个键和值都是字符串的特殊的键值对映射，在设计上应该是关联一个 Hashtable 并将其两个泛型参数设置为 String 类型，但是 Java API 中的 Properties 直接继承了 Hashtable，这很明显是对继承的滥用。这里复用代码的方式应该是 Has-A 关系而不是 Is-A 关系，另一方面容器都属于工具类， 继承工具类本身就是一个错误的做法，使用工具类最好的方式是 Has-A 关系（关联）或 Use-A 关系（依赖）。同理，Stack 类继承 Vector 也是不正确的。Sun 公司的工程师们也会犯这种低级错误，让人唏嘘不已。

## 54、Collection 和 Collections 的区别？

Collection 是一个接口，它是 Set、List 等容器的父接口；Collections 是个一个工具类，提供了一系列的静态方法来辅助容器操作，这些方法包括对容器的搜索、排序、线程安全化等等。

## 55、List、Map、Set 三个接口存取元素时，各有什么特点？

List 以特定索引来存取元素，可以有重复元素。Set 不能存放重复元素（用对象的 equals()方法来区分元素是否重复）。Map 保存键值对（key-value pair） 映射，映射关系可以是一对一或多对一。Set 和 Map 容器都有基于哈希存储和排序树的两种实现版本，基于哈希存储的版本理论存取时间复杂度为 O(1)，而基于排序树版本的实现在插入或删除元素时会按照元素或元素的键（key）构成排序树从而达到排序和去重的效果。

## 56、TreeMap 和 TreeSet 在排序时如何比较元素？ Collections 工具类中的 sort()方法如何比较元 素？

TreeSet 要求存放的对象所属的类必须实现 Comparable 接口，该接口提供了比较元素的 compareTo()方法，当插入元素时会回调该方法比较元素的大小。TreeMap 要求存放的键值对映射的键必须实现 Comparable 接口从而根据键对元素进 行排 序。Collections 工具类的 sort 方法有两种重载的形式，第一种要求传入的待排序容器中存放的对象比较实现 Comparable 接口以实现元素的比较；第二种不强制性的要求容器中的元素必须可比较，但是要求传入第二个参

数，参数是 Comparator 接口的子类型（需要重写 compare 方法实现元素的比较），相当于一个临时定义的排序规则，其实就是通过接口注入比较元素大小的算法，也是对回调模式的应用（Java 中对函数式编程的支持）。

## 57、Thread 类的 sleep()方法和对象的 wait()方法都可以让线程暂停执行，它们有什么区别?

sleep()方法（休眠）是线程类（Thread）的静态方法，调用此方法会让当前线程暂停执行指定的时间，将执行机会（CPU）让给其他线程，但是对象的锁依然保持，因此休眠时间结束后会自动恢复（线程回到就绪状态，请参考第 66 题中的线程状态转换图）。wait()是 Object 类的方法，调用对象的 wait()方法导致当前线程放弃对象的锁（线程暂停执行），进入对象的等待池（wait pool）， 只有调用对象的 notify()方法（或 notifyAll()方法）时才能唤醒等待池中的线程进入等锁池（lock pool），如果线程重新获得对象的锁就可以进入就绪状态。

补充：可能不少人对什么是进程，什么是线程还比较模糊，对于为什么需要多线程编程也不是特别理解。简单的说：进程是具有一定独立功能的程序关于某个数据集合上的一次运行活动，是操作系统进行资源分配和调度的一个独立单位；线程是进程的一个实体，是 CPU 调度和分派的基本单位，是比进程更小的能独立运行的基本单位。线程的划分尺度小于进程，这使得多线程程序的并发性高；进程在执行时通常拥有独立的内存单元，而线程之间可以共享内存。使用多线程的编程通常能够带来更好的性能和用户体验，但是多线程的程序对于其他程序是不友好的，因为它可能占用了更多的 CPU 资源。当然，也不是线程越多，程序的性能就越好，因为线程之间的调度和切换也会浪费 CPU 时间。时下很时髦的Node.js 就采用了单线程异步 I/O 的工作模式。

## 58、线程的 sleep()方法和 yield()方法有什么区别？

1. sleep()方法给其他线程运行机会时不考虑线程的优先级，因此会给低优先级的线程以运行的机会；yield()方法只会给相同优先级或更高优先级的线程以运行的机会；
2. 线程执行 sleep()方法后转入阻塞（blocked）状态，而执行 yield()方法后转入就绪（ready）状态；
3. sleep()方法声明抛出 InterruptedException，而 yield()方法没有声明任何异常；
4. sleep()方法比 yield()方法（跟操作系统 CPU 调度相关）具有更好的可移植性。

## 59、当一个线程进入一个对象的 synchronized 方法A 之后，其它线程是否可进入此对象的synchronized 方法 B？

不能。其它线程只能访问该对象的非同步方法，同步方法则不能进入。因为非静态方法上的 synchronized 修饰符要求执行方法时要获得对象的锁，如果已经进入 A 方法说明对象锁已经被取走，那么试图进入 B 方法的线程就只能在等锁池

（注意不是等待池哦）中等待对象的锁。

## 60、请说出与线程同步以及线程调度相关的方法。

1. wait()：使一个线程处于等待（阻塞）状态，并且释放所持有的对象的锁；
2. sleep()：使一个正在运行的线程处于睡眠状态，是一个静态方法，调用此方法要处理 InterruptedException 异常；
3. notify()：唤醒一个处于等待状态的线程，当然在调用此方法的时候，并不能确切的唤醒某一个等待状态的线程，而是由 JVM 确定唤醒哪个线程，而且与优先级无关；
4. notityAll()：唤醒所有处于等待状态的线程，该方法并不是将对象的锁给所有线程，而是让它们竞争，只有获得锁的线程才能进入就绪状态；

补充：Java 5 通过 Lock 接口提供了显式的锁机制（explicit lock），增强了灵活性以及对线程的协调。Lock 接口中定义了加锁（lock()）和解锁（unlock()） 的方法，同时还提供了 newCondition()方法来产生用于线程之间通信的Condition 对象；此外，Java 5 还提供了信号量机制（semaphore），信号量可以用来限制对某个共享资源进行访问的线程的数量。在对资源进行访问之前，线程必须得到信号量的许可（调用 Semaphore 对象的 acquire()方法）；在完成对资源的访问后，线程必须向信号量归还许可（调用 Semaphore 对象的release()方法）。

## 61、编写多线程程序有几种实现方式？

Java 5 以前实现多线程有两种实现方法：一种是继承 Thread 类；另一种是实现 Runnable 接口。两种方式都要通过重写 run()方法来定义线程的行为，推荐使用后者，因为 Java 中的继承是单继承，一个类有一个父类，如果继承了Thread 类就无法再继承其他类了，显然使用 Runnable 接口更为灵活。

补充：Java 5 以后创建线程还有第三种方式：实现 Callable 接口，该接口中的 call 方法可以在线程执行结束时产生一个返回值。

## 62、synchronized 关键字的用法？

synchronized 关键字可以将对象或者方法标记为同步，以实现对对象和方法的互斥访问，可以用 synchronized(对象) { … }定义同步代码块，或者在声明方法时将 synchronized 作为方法的修饰符。在第 60 题的例子中已经展示了synchronized 关键字的用法。

## 63、举例说明同步和异步。

如果系统中存在临界资源（资源数量少于竞争资源的线程数量的资源），例如正在写的数据以后可能被另一个线程读到，或者正在读的数据可能已经被另一个线程写过了，那么这些数据就必须进行同步存取（数据库操作中的排他锁就是最好的例子）。当应用程序在对象上调用了一个需要花费很长时间来执行的方法，并且不希望让程序等待方法的返回时，就应该使用异步编程，在很多情况下采用异步途径往往更有效率。事实上，所谓的同步就是指阻塞式操作，而异步就是非阻塞式操作。

## 64、启动一个线程是调用 run()还是 start()方法？

启动一个线程是调用 start()方法，使线程所代表的虚拟处理机处于可运行状态，这意味着它可以由 JVM 调度并执行，这并不意味着线程就会立即运行。run() 方法是线程启动后要进行回调（callback）的方法。

## 65、什么是线程池（thread pool）？

在面向对象编程中，创建和销毁对象是很费时间的，因为创建一个对象要获取内存资源或者其它更多资源。在 Java 中更是如此，虚拟机将试图跟踪每一个对象， 以便能够在对象销毁后进行垃圾回收。所以提高服务程序效率的一个手段就是尽可能减少创建和销毁对象的次数，特别是一些很耗资源的对象创建和销毁，这就是”池化资源”技术产生的原因。线程池顾名思义就是事先创建若干个可执行的线程放入一个池（容器）中，需要的时候从池中获取线程不用自行创建，使用完毕不需要销毁线程而是放回池中，从而减少创建和销毁线程对象的开销。Java 5+ 中的 Executor 接口定义一个执行线程的工具。它的子类型即线程池接口是ExecutorService。要配置一个线程池是比较复杂的，尤其是对于线程池的原理不是很清楚的情况下，因此在工具类 Executors 面提供了一些静态工厂方法， 生成一些常用的线程池，如下所示：

1. newSingleThreadExecutor：创建一个单线程的线程池。这个线程池只有一个线程在工作，也就是相当于单线程串行执行所有任务。如果这个唯一的线程因为异常结束，那么会有一个新的线程来替代它。此线程池保证所有任务的执行顺序按照任务的提交顺序执行。
2. newFixedThreadPool：创建固定大小的线程池。每次提交一个任务就创建一个线程，直到线程达到线程池的最大大小。线程池的大小一旦达到最大值就会保持不变，如果某个线程因为执行异常而结束，那么线程池会补充一个新线程。
3. newCachedThreadPool：创建一个可缓存的线程池。如果线程池的大小超过了处理任务所需要的线程，那么就会回收部分空闲（60 秒不执行任务）的线程，当任务数增加时，此线程池又可以智能的添加新线程来处理任务。此线程池不会对线程池大小做限制，线程池大小完全依赖于操作系统（或者说 JVM）能够创建的最大线程大小。
4. newScheduledThreadPool：创建一个大小无限的线程池。此线程池支持定时以及周期性执行任务的需求。
5. newSingleThreadExecutor：创建一个单线程的线程池。此线程池支持定时以及周期性执行任务的需求。

第 60 题的例子中演示了通过 Executors 工具类创建线程池并使用线程池执行线程的代码。如果希望在服务器上使用线程池，强烈建议使用newFixedThreadPool 方法来创建线程池，这样能获得更好的性能。

## 66、线程的基本状态以及状态之间的关系？

说明：其中 Running 表示运行状态，Runnable 表示就绪状态（万事俱备，只欠CPU），Blocked 表示阻塞状态，阻塞状态又有多种情况，可能是因为调用 wait() 方法进入等待池，也可能是执行同步方法或同步代码块进入等锁池，或者是调用了 sleep()方法或 join()方法等待休眠或其他线程结束，或是因为发生了 I/O 中断。

## 67、简述 synchronized 和java.util.concurrent.locks.Lock 的异同？

Lock 是 Java 5 以后引入的新的 API，和关键字 synchronized 相比主要相同点：Lock 能完成 synchronized 所实现的所有功能；主要不同点：Lock 有比synchronized 更精确的线程语义和更好的性能，而且不强制性的要求一定要获得锁。synchronized 会自动释放锁，而 Lock 一定要求程序员手工释放，并且最好在 finally 块中释放（这是释放外部资源的最好的地方）。

## 68、Java 中如何实现序列化，有什么意义？

序列化就是一种用来处理对象流的机制，所谓对象流也就是将对象的内容进行流化。可以对流化后的对象进行读写操作，也可将流化后的对象传输于网络之间。序列化是为了解决对象流读写操作时可能引发的问题（如果不进行序列化可能会

存在数据乱序的问题）。要实现序列化，需要让一个类实现 Serializable 接口， 该接口是一个标识性接口，标注该类对象是可被序列化的，然后使用一个输出流来构造一个对象输出流并通过 writeObject(Object)方法就可以将实现对象写出（即保存其状态）；如果需要反序列化则可以用一个输入流建立对象输入流， 然后通过 readObject 方法从流中读取对象。序列化除了能够实现对象的持久化之外，还能够用于对象的深度克隆（可以参考第 29 题）。

## 69、Java 中有几种类型的流？

字节流和字符流。字节流继承于 InputStream、OutputStream，字符流继承于Reader、Writer。在 java.io 包中还有许多其他的流，主要是为了提高性能和使用方便。关于 Java 的 I/O 需要注意的有两点：一是两种对称性（输入和输出的对称性，字节和字符的对称性）；二是两种设计模式（适配器模式和装潢模式）。另外 Java 中的流不同于 C#的是它只有一个维度一个方向。

## 70、写一个方法，输入一个文件名和一个字符串，统计这个字符串在这个文件中出现的次数。

代码如下：

import java.io.BufferedReader; import java.io.FileReader; public final class MyUtil {

// 工具类中的方法都是静态方式访问的因此将构造器私有不允许创建对象

( 绝 对 好 习 惯 ) private MyUtil() {

throw new AssertionError();

}

/\*\*

* 统计给定文件中给定字符串的出现次数

\*

* @param filename 文件名
* @param word 字符串
* @return 字符串在文件中出现的次数

\*/

public static int countWordInFile(String filename, String word)

{

int counter = 0;

try (FileReader fr = new FileReader(filename)) {

try (BufferedReader br = new BufferedReader(fr))

{

String line = null;

while ((line = br.readLine()) != null) { int index = -1;

while (line.length() >=

word.length() && (index =

line.substring(index + word.length());

line.indexOf(word)) >= 0) { counter++;

line =

}

}

}

}

catch (Exception ex) {

ex.printStackTrace();

}

}复制代码

}

return counter;

## 71、如何用 Java 代码列出一个目录下所有的文件？

如果只要求列出当前文件夹下的文件，代码如下所示：

import java.io.File; class Test12 {

public static void main(String[] args) {

File f = new File("/Users/Hao/Downloads");

for (File temp : f.listFiles()) {

if(temp.isFile()) {

System.out.println(temp.getName());

}

}

}

}复制代码

如果需要对文件夹继续展开，代码如下所示：

import java.io.File; class Test12 {

public static void main(String[] args) { showDirectory(new File("/Users/Hao/Downloads"));

}

public static void showDirectory(File f) {

\_walkDirectory(f, 0);

}

private static void \_walkDirectory(File f, int level) {

if(f.isDirectory()) {

for (File temp : f.listFiles()) {

\_walkDirectory(temp, level + 1);

}

} else {

for (int i = 0; i < level - 1; i++) { System.out.print("t");

}

System.out.println(f.getName());

}

}

}复制代码

在 Java 7 中可以使用 NIO.2 的 API 来做同样的事情，代码如下所示：

class ShowFileTest {

public static void main(String[] args) throws IOException { Path initPath = Paths.get("/Users/Hao/Downloads"); Files.walkFileTree(initPath, new

SimpleFileVisitor<Path>() {

@Override

public FileVisitResult visitFile(Path file,

BasicFileAttributes

attrs)

throws IOException {

System.out.println(file.getFileName().toString());

return FileVisitResult.CONTINUE;

}

}

);

}

}复制代码

## 72、用 Java 的套接字编程实现一个多线程的回显

**（echo）服务器。**

## 73、XML 文档定义有几种形式？它们之间有何本质区别？解析 XML 文档有哪几种方式？

XML 文档定义分为 DTD 和 Schema 两种形式，二者都是对 XML 语法的约束，其本质区别在于 Schema 本身也是一个 XML 文件，可以被 XML 解析器解析，而且可以为 XML 承载的数据定义类型，约束能力较之 DTD 更强大。对 XML 的解析主要有 DOM（文档对象模型，Document Object Model）、SAX（Simple API forXML） 和 StAX（Java 6 中引入的新的解析 XML 的方式，Streaming API for XML）， 其中 DOM 处理大型文件时其性能下降的非常厉害，这个问题是由 DOM 树结构占用的内存较多造成的，而且 DOM 解析方式必须在解析文件之前把整个文档装入内存，适合对 XML 的随机访问（典型的用空间换取时间的策略）；SAX 是事件驱动型的 XML 解析方式，它顺序读取 XML 文件，不需要一次全部装载整个文件。当遇到像文件开头，文档结束，或者标签开头与标签结束时，它会触发一个事件， 用户通过事件回调代码来处理 XML 文件，适合对 XML 的顺序访问；顾名思义， StAX 把重点放在流上，实际上 StAX 与其他解析方式的本质区别就在于应用程序能够把 XML 作为一个事件流来处理。将 XML 作为一组事件来处理的想法并不新颖（ SAX 就是这样做的），但不同之处在于 StAX 允许应用程序代码把这些事件逐个拉出来，而不用提供在解析器方便时从解析器中接收事件的处理程序。

## 74、你在项目中哪些地方用到了 XML？

XML 的主要作用有两个方面：数据交换和信息配置。在做数据交换时，XML 将数据用标签组装成起来，然后压缩打包加密后通过网络传送给接收者，接收解密与解压缩后再从 XML 文件中还原相关信息进行处理，XML 曾经是异构系统间交换数据的事实标准，但此项功能几乎已经被被 JSON（JavaScript Object Notation） 取而代之。当然，目前很多软件仍然使用 XML 来存储配置信息，我们在很多项目中通常也会将作为配置信息的硬代码写在 XML 文件中，Java 的很多框架也是这么做的，而且这些框架都选择了 dom4j 作为处理 XML 的工具，因为 Sun 公司的官方 API 实在不怎么好用。

补充：现在有很多时髦的软件（如 Sublime）已经开始将配置文件书写成 JSON 格式，我们已经强烈的感受到 XML 的另一项功能也将逐渐被业界抛弃。

## 75、阐述 JDBC 操作数据库的步骤。

下面的代码以连接本机的 Oracle 数据库为例，演示 JDBC 操作数据库的步骤。

1. 加载驱动。

Class.forName("oracle.jdbc.driver.OracleDriver");复制代码

1. 创建连接。

Connection con = DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:orcl"," scott", "tiger");复制代码

1. 创建语句。

PreparedStatement ps = con.prepareStatement("select \* from emp where sal between ? and ?");

ps.setint(1, 1000); ps.setint(2, 3000);复制代码

1. 执行语句。

ResultSet rs = ps.executeQuery();复制代码

1. 处理结果。while(rs.next()) {

System.out.println(rs.getint("empno") + " - " + rs.getString("ename"));

}复制代码

1. 关闭资源。

finally {

if(con != null) {

try {

con.close();

}

catch (SQLException e) {

e.printStackTrace();

}

}

}复制代码

**提示：**关闭外部资源的顺序应该和打开的顺序相反，也就是说先关闭 ResultSet、再关闭 Statement、在关闭 Connection。上面的代码只关闭了 Connection（连接），虽然通常情况下在关闭连接时，连接上创建的语句和打开的游标也会关闭， 但不能保证总是如此，因此应该按照刚才说的顺序分别关闭。此外，第一步加载驱动在 JDBC 4.0 中是可以省略的（自动从类路径中加载驱动），但是我们建议保留。

## 76、Statement 和 PreparedStatement 有什么区别？哪个性能更好？

与 Statement 相比，①PreparedStatement 接口代表预编译的语句，它主要的优势在于可以减少 SQL 的编译错误并增加 SQL 的安全性（减少 SQL 注射攻击的可能性）；②PreparedStatement 中的 SQL 语句是可以带参数的，避免了用字符串连接拼接 SQL 语句的麻烦和不安全；③当批量处理 SQL 或频繁执行相同的查询时，PreparedStatement 有明显的性能上的优势，由于数据库可以将编译优化后的 SQL 语句缓存起来，下次执行相同结构的语句时就会很快（不用再次编译和生成执行计划）。

补充：为了提供对存储过程的调用，JDBC API 中还提供了 CallableStatement 接口。存储过程（Stored Procedure）是数据库中一组为了完成特定功能的 SQL

语句的集合，经编译后存储在数据库中，用户通过指定存储过程的名字并给出参数（如果该存储过程带有参数）来执行它。虽然调用存储过程会在网络开销、安全性、性能上获得很多好处，但是存在如果底层数据库发生迁移时就会有很多麻烦，因为每种数据库的存储过程在书写上存在不少的差别。

## 77、使用 JDBC 操作数据库时，如何提升读取数据的性能？如何提升更新数据的性能？

要提升读取数据的性能，可以指定通过结果集（ResultSet）对象的setFetchSize()方法指定每次抓取的记录数（典型的空间换时间策略）；要提升更新数据的性能可以使用 PreparedStatement 语句构建批处理，将若干 SQL 语句置于一个批处理中执行。

## 78、在进行数据库编程时，连接池有什么作用？

由于创建连接和释放连接都有很大的开销（尤其是数据库服务器不在本地时，每次建立连接都需要进行 TCP 的三次握手，释放连接需要进行 TCP 四次握手，造成的开销是不可忽视的），为了提升系统访问数据库的性能，可以事先创建若干连接置于连接池中，需要时直接从连接池获取，使用结束时归还连接池而不必关闭连接，从而避免频繁创建和释放连接所造成的开销，这是典型的用空间换取时间的策略（浪费了空间存储连接，但节省了创建和释放连接的时间）。池化技术在 Java 开发中是很常见的，在使用线程时创建线程池的道理与此相同。基于Java 的开源数据库连接池主要有：C3P0、Proxool、DBCP、BoneCP、Druid 等。补充：在计算机系统中时间和空间是不可调和的矛盾，理解这一点对设计满足性能要求的算法是至关重要的。大型网站性能优化的一个关键就是使用缓存，而缓存跟上面讲的连接池道理非常类似，也是使用空间换时间的策略。可以将热点数据置于缓存中，当用户查询这些数据时可以直接从缓存中得到，这无论如何也快过去数据库中查询。当然，缓存的置换策略等也会对系统性能产生重要影响，对于这个问题的讨论已经超出了这里要阐述的范围。

## 79、什么是 DAO 模式？

DAO（Data Access Object）顾名思义是一个为数据库或其他持久化机制提供了抽象接口的对象，在不暴露底层持久化方案实现细节的前提下提供了各种数据访问操作。在实际的开发中，应该将所有对数据源的访问操作进行抽象化后封装在一个公共 API 中。用程序设计语言来说，就是建立一个接口，接口中定义了此应用程序中将会用到的所有事务方法。在这个应用程序中，当需要和数据源进行交互的时候则使用这个接口，并且编写一个单独的类来实现这个接口，在逻辑上该类对应一个特定的数据存储。DAO 模式实际上包含了两个模式，一是DataAccessor（数据访问器），二是 Data Object（数据对象），前者要解决如何访问数据的问题，而后者要解决的是如何用对象封装数据。

## 80、事务的 ACID 是指什么？

1. 原子性(Atomic)：事务中各项操作，要么全做要么全不做，任何一项操作的失败都会导致整个事务的失败；
2. 一致性(Consistent)：事务结束后系统状态是一致的；
3. 隔离性(Isolated)：并发执行的事务彼此无法看到对方的中间状态；
4. 持久性(Durable)：事务完成后所做的改动都会被持久化，即使发生灾难性的失败。通过日志和同步备份可以在故障发生后重建数据。

**补充：**关于事务，在面试中被问到的概率是很高的，可以问的问题也是很多的。首先需要知道的是，只有存在并发数据访问时才需要事务。当多个事务访问同一数据时，可能会存在 5 类问题，包括 3 类数据读取问题（脏读、不可重复读和

幻读）和 2 类数据更新问题（第 1 类丢失更新和第 2 类丢失更新）。

**脏读（Dirty Read）：**A 事务读取 B 事务尚未提交的数据并在此基础上操作， 而 B 事务执行回滚，那么 A 读取到的数据就是脏数据。

**不可重复读（Unrepeatable Read）：**事务 A 重新读取前面读取过的数据，发现该数据已经被另一个已提交的事务 B 修改过了。

**幻读（Phantom Read）：**事务 A 重新执行一个查询，返回一系列符合查询条件的行，发现其中插入了被事务 B 提交的行。

第 1 类丢失更新：事务 A 撤销时，把已经提交的事务 B 的更新数据覆盖了。

第 2 类丢失更新：事务 A 覆盖事务 B 已经提交的数据，造成事务 B 所做的操作丢失。

数据并发访问所产生的问题，在有些场景下可能是允许的，但是有些场景下可能就是致命的，数据库通常会通过锁机制来解决数据并发访问问题，按锁定对象不同可以分为表级锁和行级锁；按并发事务锁定关系可以分为共享锁和独占锁，具体的内容大家可以自行查阅资料进行了解。直接使用锁是非常麻烦的，为此数据库为用户提供了自动锁机制，只要用户指定会话的事务隔离级别，数据库就会通过分析 SQL 语句然后为事务访问的资源加上合适的锁，此外，数据库还会维护这些锁通过各种手段提高系统的性能，这些对用户来说都是透明的（就是说你不用理解，事实上我确实也不知道）。ANSI/ISOSQL 92 标准定义了 4 个等级的事务隔离级别，如下表所示：

需要说明的是，事务隔离级别和数据访问的并发性是对立的，事务隔离级别越高并发性就越差。所以要根据具体的应用来确定合适的事务隔离级别，这个地方没有万能的原则。

## 81、JDBC 中如何进行事务处理？

Connection 提供了事务处理的方法，通过调用 setAutoCommit(false)可以设置手动提交事务；当事务完成后用 commit()显式提交事务；如果在事务处理过程中发生异常则通过 rollback()进行事务回滚。除此之外，从 JDBC 3.0 中还引入了 Savepoint（保存点）的概念，允许通过代码设置保存点并让事务回滚到指定的保存点。

## 82、JDBC 能否处理 Blob 和 Clob？

Blob 是指二进制大对象（Binary Large Object），而 Clob 是指大字符对象

（Character Large Objec），因此其中 Blob 是为存储大的二进制数据而设计的，而 Clob 是为存储大的文本数据而设计的。JDBC 的 PreparedStatement 和ResultSet 都提供了相应的方法来支持 Blob 和 Clob 操作。

## 83、简述正则表达式及其用途。

在编写处理字符串的程序时，经常会有查找符合某些复杂规则的字符串的需要。正则表达式就是用于描述这些规则的工具。换句话说，正则表达式就是记录文本规则的代码。

说明：计算机诞生初期处理的信息几乎都是数值，但是时过境迁，今天我们使用计算机处理的信息更多的时候不是数值而是字符串，正则表达式就是在进行字符串匹配和处理的时候最为强大的工具，绝大多数语言都提供了对正则表达式的支持。

## 84、Java 中是如何支持正则表达式操作的？

Java 中的 String 类提供了支持正则表达式操作的方法，包括：matches()、replaceAll()、replaceFirst()、split()。此外，Java 中可以用 Pattern 类表示正则表达式对象，它提供了丰富的 API 进行各种正则表达式操作。

面试题： - 如果要从字符串中截取第一个英文左括号之前的字符串，例如：北京市(朝阳区)(西城区)(海淀区)，截取结果为：北京市，那么正则表达式怎么写？ import java.util.regex.Matcher;

import java.util.regex.Pattern; class RegExpTest {

public static void main(String[] args) {

String str = "北京市(朝阳区)(西城区)(海淀区)";

Pattern p = Pattern.compile(".\*?(?=\()"); Matcher m = p.matcher(str);

if(m.find()) {

System.out.println(m.group());

}

}

}复制代码

## 85、获得一个类的类对象有哪些方式？

1. 方法 1：类型.class，例如：String.class
2. 方法 2：对象.getClass()，例如：”hello”.getClass()
3. 方法 3：Class.forName()，例如：Class.forName(“java.lang.String”)

## 86、如何通过反射创建对象？

方法 1：通过类对象调用 newInstance()方法，例如： String.class.newInstance()

方法 2：通过类对象的 getConstructor()或 getDeclaredConstructor()方法获得构造器（Constructor）对象并调用其 newInstance()方法创建对象，例如： String.class.getConstructor(String.class).newInstance(“Hello”);

## 87、如何通过反射获取和设置对象私有字段的值？

可以通过类对象的 getDeclaredField()方法字段（Field）对象，然后再通过字段对象的 setAccessible(true)将其设置为可以访问，接下来就可以通过get/set 方法来获取/设置字段的值了。下面的代码实现了一个反射的工具类， 其中的两个静态方法分别用于获取和设置私有字段的值，字段可以是基本类型也可以是对象类型且支持多级对象操作。

## 88、如何通过反射调用对象的方法？

请看下面的代码：

import java.lang.reflect.Method; class MethodInvokeTest {

public static void main(String[] args) throws Exception { String str = "hello";

Method m = str.getClass().getMethod("toUpperCase");

}

}复制代码

System.out.println(m.invoke(str));

// HELLO

## 89、简述一下面向对象的”六原则一法则”。

1. 单一职责原则：一个类只做它该做的事情。（单一职责原则想表达的就是” 高内聚”，写代码最终极的原则只有六个字”高内聚、低耦合”，就如同葵花宝典或辟邪剑谱的中心思想就八个字”欲练此功必先自宫”，所谓的高内聚就是一个代码模块只完成一项功能，在面向对象中，如果只让一个类完成它该做的事， 而不涉及与它无关的领域就是践行了高内聚的原则，这个类就只有单一职责。我们都知道一句话叫”因为专注，所以专业”，一个对象如果承担太多的职责，那么注定它什么都做不好。这个世界上任何好的东西都有两个特征，一个是功能单一，好的相机绝对不是电视购物里面卖的那种一个机器有一百多种功能的，它基本上只能照相；另一个是模块化，好的自行车是组装车，从减震叉、刹车到变速器，所有的部件都是可以拆卸和重新组装的，好的乒乓球拍也不是成品拍，一定是底板和胶皮可以拆分和自行组装的，一个好的软件系统，它里面的每个功能模块也应该是可以轻易的拿到其他系统中使用的，这样才能实现软件复用的目标。）
2. 开闭原则：软件实体应当对扩展开放，对修改关闭。（在理想的状态下， 当我们需要为一个软件系统增加新功能时，只需要从原来的系统派生出一些新类就可以，不需要修改原来的任何一行代码。要做到开闭有两个要点：①抽象是关键，一个系统中如果没有抽象类或接口系统就没有扩展点；②封装可变性，将系统中的各种可变因素封装到一个继承结构中，如果多个可变因素混杂在一起，系统将变得复杂而换乱，如果不清楚如何封装可变性，可以参考《设计模式精解》一书中对桥梁模式的讲解的章节。）
3. 依赖倒转原则：面向接口编程。（该原则说得直白和具体一些就是声明方法的参数类型、方法的返回类型、变量的引用类型时，尽可能使用抽象类型而不用具体类型，因为抽象类型可以被它的任何一个子类型所替代，请参考下面的里氏替换原则。）
4. 里氏替换原则：任何时候都可以用子类型替换掉父类型。（关于里氏替换原则的描述，Barbara Liskov 女士的描述比这个要复杂得多，但简单的说就是能用父类型的地方就一定能使用子类型。里氏替换原则可以检查继承关系是否合理，如果一个继承关系违背了里氏替换原则，那么这个继承关系一定是错误的， 需要对代码进行重构。例如让猫继承狗，或者狗继承猫，又或者让正方形继承长方形都是错误的继承关系，因为你很容易找到违反里氏替换原则的场景。需要注意的是：子类一定是增加父类的能力而不是减少父类的能力，因为子类比父类的能力更多，把能力多的对象当成能力少的对象来用当然没有任何问题。）
5. 接口隔离原则：接口要小而专，绝不能大而全。（臃肿的接口是对接口的污染，既然接口表示能力，那么一个接口只应该描述一种能力，接口也应该是高度内聚的。例如，琴棋书画就应该分别设计为四个接口，而不应设计成一个接口中的四个方法，因为如果设计成一个接口中的四个方法，那么这个接口很难用， 毕竟琴棋书画四样都精通的人还是少数，而如果设计成四个接口，会几项就实现

几个接口，这样的话每个接口被复用的可能性是很高的。Java 中的接口代表能力、代表约定、代表角色，能否正确的使用接口一定是编程水平高低的重要标识。）

1. 合成聚合复用原则：优先使用聚合或合成关系复用代码。（通过继承来复用代码是面向对象程序设计中被滥用得最多的东西，因为所有的教科书都无一例外的对继承进行了鼓吹从而误导了初学者，类与类之间简单的说有三种关系， Is-A 关系、Has-A 关系、Use-A 关系，分别代表继承、关联和依赖。其中，关联关系根据其关联的强度又可以进一步划分为关联、聚合和合成，但说白了都是Has-A 关系，合成聚合复用原则想表达的是优先考虑 Has-A 关系而不是 Is-A 关系复用代码，原因嘛可以自己从百度上找到一万个理由，需要说明的是，即使在 Java 的 API 中也有不少滥用继承的例子，例如 Properties 类继承了Hashtable 类，Stack 类继承了 Vector 类，这些继承明显就是错误的，更好的做法是在 Properties 类中放置一个 Hashtable 类型的成员并且将其键和值都设置为字符串来存储数据，而 Stack 类的设计也应该是在 Stack 类中放一个Vector 对象来存储数据。记住：任何时候都不要继承工具类，工具是可以拥有并可以使用的，而不是拿来继承的。）
2. 迪米特法则：迪米特法则又叫最少知识原则，一个对象应当对其他对象有尽可能少的了解。（迪米特法则简单的说就是如何做到”低耦合”，门面模式和调停者模式就是对迪米特法则的践行。对于门面模式可以举一个简单的例子，你去一家公司洽谈业务，你不需要了解这个公司内部是如何运作的，你甚至可以对这个公司一无所知，去的时候只需要找到公司入口处的前台美女，告诉她们你要做什么，她们会找到合适的人跟你接洽，前台的美女就是公司这个系统的门面。再复杂的系统都可以为用户提供一个简单的门面，Java Web 开发中作为前端控制器的 Servlet 或 Filter 不就是一个门面吗，浏览器对服务器的运作方式一无所知，但是通过前端控制器就能够根据你的请求得到相应的服务。调停者模式也可以举一个简单的例子来说明，例如一台计算机，CPU、内存、硬盘、显卡、声卡各种设备需要相互配合才能很好的工作，但是如果这些东西都直接连接到一起，计算机的布线将异常复杂，在这种情况下，主板作为一个调停者的身份出现， 它将各个设备连接在一起而不需要每个设备之间直接交换数据，这样就减小了系统的耦合度和复杂度，如下图所示。迪米特法则用通俗的话来将就是不要和陌生人打交道，如果真的需要，找一个自己的朋友，让他替你和陌生人打交道。）

## 90、简述一下你了解的设计模式。

所谓设计模式，就是一套被反复使用的代码设计经验的总结（情境中一个问题经过证实的一个解决方案）。使用设计模式是为了可重用代码、让代码更容易被他人理解、保证代码可靠性。设计模式使人们可以更加简单方便的复用成功的设计和体系结构。将已证实的技术表述成设计模式也会使新系统开发者更加容易理解其设计思路。

在 GoF 的《Design Patterns: Elements of Reusable

Object-OrientedSoftware》中给出了三类（创建型[对类的实例化过程的抽象 化]、结构型[描述如何将类或对象结合在一起形成更大的结构]、行为型[对在不同的对象之间划分责任和算法的抽象化]）共 23 种设计模式，包括：Abstract Factory（抽象工厂模式），Builder（建造者模式），Factory Method（工厂方法模式），Prototype（原始模型模式），Singleton（单例模式）；Facade（门面模式），Adapter（适配器模式），Bridge（桥梁模式），Composite（合成模式），Decorator（装饰模式），Flyweight（享元模式），Proxy（代理模式）； Command（命令模式），Interpreter（解释器模式），Visitor（访问者模式）， Iterator（迭代子模式），Mediator（调停者模式），Memento（备忘录模式）， Observer（观察者模式），State（状态 模式 ），Strategy（策略 模式 ），

Template Method（模板方法模式），Chain Of Responsibility（责任链模式）。面试被问到关于设计模式的知识时，可以拣最常用的作答，例如：

1. 工厂模式：工厂类可以根据条件生成不同的子类实例，这些子类有一个公共的抽象父类并且实现了相同的方法，但是这些方法针对不同的数据进行了不同的操作（多态方法）。当得到子类的实例后，开发人员可以调用基类中的方法而不必考虑到底返回的是哪一个子类的实例。
2. 代理模式：给一个对象提供一个代理对象，并由代理对象控制原对象的引用。实际开发中，按照使用目的的不同，代理可以分为：远程代理、虚拟代理、保护代理、Cache 代理、防火墙代理、同步化代理、智能引用代理。
3. 适配器模式：把一个类的接口变换成客户端所期待的另一种接口，从而使原本因接口不匹配而无法在一起使用的类能够一起工作。
4. 模板方法模式：提供一个抽象类，将部分逻辑以具体方法或构造器的形式实现，然后声明一些抽象方法来迫使子类实现剩余的逻辑。不同的子类可以以不同的方式实现这些抽象方法（多态实现），从而实现不同的业务逻辑。除此之外， 还可以讲讲上面提到的门面模式、桥梁模式、单例模式、装潢模式（Collections 工具类和 I/O 系统中都使用装潢模式）等，反正基本原则就是拣自己最熟悉的、用得最多的作答，以免言多必失。

## 91、用 Java 写一个单例类。

1. 饿汉式单例

public class Singleton {

private Singleton(){

}

private static Singleton instance = new Singleton(); public static Singleton getInstance(){

return instance;

}

}复制代码

1. 懒汉式单例

public class Singleton {

private static Singleton instance = null;

private Singleton() {

}

public static synchronized Singleton getInstance(){

if (instance == null) instance ＝ new Singleton();

return instance;

}

}复制代码

注意：实现一个单例有两点注意事项，①将构造器私有，不允许外界通过构造器创建对象；②通过公开的静态方法向外界返回类的唯一实例。这里有一个问题可以思考：Spring 的 IoC 容器可以为普通的类创建单例，它是怎么做到的呢？

## 92、什么是 UML？

UML 是统一建模语言（Unified Modeling Language）的缩写，它发表于 1997 年，综合了当时已经存在的面向对象的建模语言、方法和过程，是一个支持模型化和软件系统开发的图形化语言，为软件开发的所有阶段提供模型化和可视化支持。使用 UML 可以帮助沟通与交流，辅助应用设计和文档的生成，还能够阐释系统的结构和行为。

## 93、UML 中有哪些常用的图？

UML 定义了多种图形化的符号来描述软件系统部分或全部的静态结构和动态结构，包括：用例图（use case diagram）、类图（class diagram）、时序图

（sequencediagram）、协作图（collaboration diagram）、状态图（statechart diagram）、活动图（activity diagram）、构件图（component diagram）、部署图（deploymentdiagram）等。在这些图形化符号中，有三种图最为重要，分别是：用例图（用来捕获需求，描述系统的功能，通过该图可以迅速的了解系统的功能模块及其关系）、类图（描述类以及类与类之间的关系，通过该图可以快速了解系统）、时序图（描述执行特定任务时对象之间的交互关系以及执行顺序， 通过该图可以了解对象能接收的消息也就是说对象能够向外界提供的服务）。用例图：

类图：

时序图：

## 94、用 Java 写一个冒泡排序。

冒泡排序几乎是个程序员都写得出来，但是面试的时候如何写一个逼格高的冒泡排序却不是每个人都能做到，下面提供一个参考代码：

import java.util.Comparator;

/\*\*

* 排序器接口(策略模式: 将算法封装到具有共同接口的独立的类中使得它们可以相互替换)
* @author 骆 昊

\*

\*/

public interface Sorter {

/\*\*

* 排序
* @param list 待排序的数组

\*/

* 排序

public <T extends Comparable<T>> void sort(T[] list);

/\*\*

* @param list 待排序的数组
* @param comp 比较两个对象的比较器

\*/

public <T> void sort(T[] list, Comparator<T> comp);

}

import java.util.Comparator;

/\*\*

* 冒泡排序

\*

* @author 骆 昊

\*

\*/

public class BubbleSorter implements Sorter { @Override

public <T extends Comparable<T>> void sort(T[] list) { Boolean swapped = true;

++i) {

{

for (int i = 1, len = list.length; i < len && swapped;

swapped = false;

for (int j = 0; j < len - i; ++j) {

if (list[j].compareTo(list[j + 1]) > 0)

T temp = list[j]; list[j] = list[j + 1]; list[j + 1] = temp;

swapped = true;

}

}

}

}

++i) {

0) {

@Override

public <T> void sort(T[] list, Comparator<T> comp) { Boolean swapped = true;

for (int i = 1, len = list.length; i < len && swapped;

swapped = false;

for (int j = 0; j < len - i; ++j) {

if (comp.compare(list[j], list[j + 1]) >

T temp = list[j]; list[j] = list[j + 1]; list[j + 1] = temp;

swapped = true;

}

}

}

}

}复制代码

## 95、用 Java 写一个折半查找。

折半查找，也称二分查找、二分搜索，是一种在有序数组中查找某一特定元素的搜索算法。搜素过程从数组的中间元素开始，如果中间元素正好是要查找的元素， 则搜素过程结束；如果某一特定元素大于或者小于中间元素，则在数组大于或小于中间元素的那一半中查找，而且跟开始一样从中间元素开始比较。如果在某一步骤数组已经为空，则表示找不到指定的元素。这种搜索算法每一次比较都使搜索范围缩小一半，其时间复杂度是 O(logN)。

import java.util.Comparator; public class MyUtil {

T

comp)

public static <T extends Comparable<T>> int binarySearch(T[] x,

key) {

return binarySearch(x, 0, x.length- 1, key);

}

// 使用循环实现的二分查找

public static <T> int binarySearch(T[] x, T key, Comparator<T>

{

int low = 0;

int high = x.length - 1;

while (low <= high) {

int mid = (low + high) >>> 1;

int cmp = comp.compare(x[mid], key);

if (cmp < 0) {

low= mid + 1;

} else if (cmp > 0) {

high= mid - 1;

} else {

return mid;

}

}

return -1;

}

int

key);

// 使用递归实现的二分查找

private static<T extends Comparable<T>> int binarySearch(T[] x,

low, int high, T key) {

if(low <= high) {

int mid = low + ((high -low) >> 1);

if(key.compareTo(x[mid])== 0) {

return mid;

} else if(key.compareTo(x[mid])< 0) {

return binarySearch(x,low, mid - 1, key);

} else {

return binarySearch(x,mid + 1, high,

}

}

return -1;

}

}复制代码

说明：上面的代码中给出了折半查找的两个版本，一个用递归实现，一个用循环实现。需要注意的是计算中间位置时不应该使用(high+ low) / 2 的方式，因为加法运算可能导致整数越界，这里应该使用以下三种方式之一：low + (high - low)/ 2 或 low + (high – low) >> 1 或(low + high) >>> 1（>>>是逻辑右移，是不带符号位的右移）

## Java 面试题（二）

1、Java 中能创建 volatile 数组吗？

2、volatile 能使得一个非原子操作变成原子操作吗？ 3、volatile 修饰符的有过什么实践？

4、volatile 类型变量提供什么保证？

5、10 个线程和 2 个线程的同步代码，哪个更容易写？

6、你是如何调用 wait（）方法的？使用 if 块还是循环？为什么？

8、什么是 Busy spin？我们为什么要使用它？ 9、Java 中怎么获取一份线程 dump 文件？

10、Swing 是线程安全的？

11、什么是线程局部变量？

12、用 wait-notify 写一段代码来解决生产者-消费者问题？ 13、用 Java 写一个线程安全的单例模式（Singleton）？

14、Java 中 sleep 方法和 wait 方法的区别？

15、什么是不可变对象（immutable object）？Java 中怎么创建一个不可变对象？

16、我们能创建一个包含可变对象的不可变对象吗？

17、Java 中应该使用什么数据类型来代表价格？ 18、怎么将 byte 转换为 String？

19、Java 中怎样将 bytes 转换为 long 类型？

20、我们能将 int 强制转换为 byte 类型的变量吗？如果该值大于 byte 类型的范围，将会出现什么现象？

## 四、Redis 面试题

## 1、Java 中能创建 volatile 数组吗？

能，Java 中可以创建 volatile 类型数组，不过只是一个指向数组的引用，而不是整个数组。我的意思是，如果改变引用指向的数组，将会受到 volatile 的保护，但是如果多个线程同时改变数组的元素，volatile 标示符就不能起到之前的保护作用了。

## 2、volatile 能使得一个非原子操作变成原子操作吗？

一个典型的例子是在类中有一个 long 类型的成员变量。如果你知道该成员变量会被多个线程访问，如计数器、价格等，你最好是将其设置为 volatile。为什么？因为 Java 中读取 long 类型变量不是原子的，需要分成两步，如果一个线程正在修改该 long 变量的值，另一个线程可能只能看到该值的一半（前 32 位）。但是对一个 volatile 型的 long 或 double 变量的读写是原子。

## 3、volatile 修饰符的有过什么实践？

一种实践是用 volatile 修饰 long 和 double 变量，使其能按原子类型来读写。double 和 long 都是 64 位宽，因此对这两种类型的读是分为两部分的， 第一次读取第一个 32 位，然后再读剩下的 32 位，这个过程不是原子的，但Java 中 volatile 型的 long 或 double 变量的读写是原子的。volatile 修复符的另一个作用是提供内存屏障（memory barrier），例如在分布式框架中的应用。简单的说，就是当你写一个 volatile 变量之前，Java 内存模型会插入一个写屏障（writebarrier），读一个 volatile 变量之前，会插入一个读屏障（read barrier）。意思就是说，在你写一个 volatile 域时，能保证任何线程都能看到你写的值，同时，在写之前，也能保证任何数值的更新对所有线程是可见的， 因为内存屏障会将其他所有写的值更新到缓存。

## 4、volatile 类型变量提供什么保证？

volatile 变量提供顺序和可见性保证，例如，JVM 或者 JIT 为了获得更好的性能会对语句重排序，但是 volatile 类型变量即使在没有同步块的情况下赋值也不会与其他语句重排序。 volatile 提供 happens-before 的保证，确保一个线程的修改能对其他线程是可见的。某些情况下，volatile 还能提供原子性，如读 64 位数据类型，像 long 和 double 都不是原子的，但 volatile 类型的double 和 long 就是原子的。

## 5、10 个线程和 2 个线程的同步代码，哪个更容易写？

从写代码的角度来说，两者的复杂度是相同的，因为同步代码与线程数量是相互独立的。但是同步策略的选择依赖于线程的数量，因为越多的线程意味着更大的竞争，所以你需要利用同步技术，如锁分离，这要求更复杂的代码和专业知识。

## 6、你是如何调用 wait（）方法的？使用 if 块还是循环？为什么？

wait() 方法应该在循环调用，因为当线程获取到 CPU 开始执行的时候，其他条件可能还没有满足，所以在处理前，循环检测条件是否满足会更好。下面是一段标准的使用 wait 和 notify 方法的代码：

// The standard idiom for using the wait method synchronized (obj) {

while (condition does not hold) obj.wait();

// (Releases lock, and reacquires on wakeup)

... // Perform action appropriate to condition

}复制代码

## 7、什么是多线程环境下的伪共享（false sharing）？

伪共享是多线程系统（每个处理器有自己的局部缓存）中一个众所周知的性能问题。伪共享发生在不同处理器的上的线程对变量的修改依赖于相同的缓存行。

## 8、什么是 Busy spin？我们为什么要使用它？

Busy spin 是一种在不释放 CPU 的基础上等待事件的技术。它经常用于避免丢失 CPU 缓存中的数据（如果线程先暂停，之后在其他 CPU 上运行就会丢失）。所以，如果你的工作要求低延迟，并且你的线程目前没有任何顺序，这样你就可以通过循环检测队列中的新消息来代替调用 sleep() 或 wait() 方法。它唯一的好处就是你只需等待很短的时间，如几微秒或几纳秒。LMAX 分布式框架是一个高性能线程间通信的库，该库有一个 BusySpinWaitStrategy 类就是基于这个概念实现的，使用 busy spin 循环 EventProcessors 等待屏障。

## 9、Java 中怎么获取一份线程 dump 文件？

在 Linux 下，你可以通过命令 kill -3 PID（Java 进程的进程 ID）来获取 Java应用的 dump 文件。在 Windows 下，你可以按下 Ctrl + Break 来获取。这样 JVM 就会将线程的 dump 文件打印到标准输出或错误文件中，它可能打印在控制台或者日志文件中，具体位置依赖应用的配置。如果你使用 Tomcat。

## 10、Swing 是线程安全的？

不是，Swing 不是线程安全的。你不能通过任何线程来更新 Swing 组件，如JTable、JList 或 JPanel，事实上，它们只能通过 GUI 或 AWT 线程来更新。这就是为什么 Swing 供 invokeAndWait() 和 invokeLater() 方法来获取其他线程的 GUI 更新请求。这些方法将更新请求放入 AWT 的线程队列中，可以一直等待，也可以通过异步更新直接返回结果。你也可以在参考答案中查看和学习到更详细的内容。

## 11、什么是线程局部变量？

线程局部变量是局限于线程内部的变量，属于线程自身所有，不在多个线程间共享。Java 提供 ThreadLocal 类来支持线程局部变量，是一种实现线程安全的方式。但是在管理环境下（如 web 服务器）使用线程局部变量的时候要特别小心， 在这种情况下，工作线程的生命周期比任何应用变量的生命周期都要长。任何线程局部变量一旦在工作完成后没有释放，Java 应用就存在内存泄露的风险。

## 12、用 wait-notify 写一段代码来解决生产者-消费者问题？

只要记住在同步块中调用 wait() 和 notify()方 法 ，如果阻塞，通过循环来测试等待条件。

## 13、用 Java 写一个线程安全的单例模式

**（Singleton）？**

一步一步创建一个线程安全的 Java 单例类。当我们说线程安全时，意思是即使初始化是在多线程环境中，仍然能保证单个实例。Java 中，使用枚举作为单例类是最简单的方式来创建线程安全单例模式的方式。

## 14、Java 中 sleep 方法和 wait 方法的区别？

虽然两者都是用来暂停当前运行的线程，但是 sleep() 实际上只是短暂停顿， 因为它不会释放锁，而 wait() 意味着条件等待，这就是为什么该方法要释放锁， 因为只有这样，其他等待的线程才能在满足条件时获取到该锁。

## 15、什么是不可变对象（immutable object）？Java 中怎么创建一个不可变对象？

不可变对象指对象一旦被创建，状态就不能再改变。任何修改都会创建一个新的对象，如 String、Integer 及其它包装类。详情参见答案，一步一步指导你在Java 中创建一个不可变的类。

## 16、我们能创建一个包含可变对象的不可变对象吗？

是的，我们是可以创建一个包含可变对象的不可变对象的，你只需要谨慎一点， 不要共享可变对象的引用就可以了，如果需要变化时，就返回原对象的一个拷贝。最常见的例子就是对象中包含一个日期对象的引用。数据类型和 Java 基础面试问题

## 17、Java 中应该使用什么数据类型来代表价格？

如果不是特别关心内存和性能的话，使用 BigDecimal，否则使用预定义精度的double 类型。

## 18、怎么将 byte 转换为 String？

可以使用 String 接收 byte[] 参数的构造器来进行转换，需要注意的点是要使用的正确的编码，否则会使用平台默认编码，这个编码可能跟原来的编码相同， 也可能不同。

## 19、Java 中怎样将 bytes 转换为 long 类型？

bytes[] 到数字类型的转换是个经常用到的代码,解决方式也不止一种。

### java 代码实现

如果不想借助任何已经有的类，完全可以自己实现这段代码，如下：

/\*\*

* + 将字节数组转为 long<br>
  + 如果 input 为 null,或 offset 指定的剩余数组长度不足 8 字节则抛出异常
  + @param input
  + @param offset 起始偏移量
  + @param littleEndian 输入数组是否小端模式
  + @return

\*/

public static long longFrom8Bytes(byte[] input, int offset, Boolean littleEndian){

long value=0;

// 循环读取每个字节通过移位运算完成 long 的 8 个字节拼装

for (int count=0;count<8;++count){

int shift=(littleEndian?count:(7-count))<<3;

value |=((long)0xff<< shift) & ((long)input[offset+count]

<< shift);

}

return value;

}复制代码

### 借助 java.nio.ByteBuffer 实现

java.nio.ByteBuffer 本身就有 getLong,getInt,getFloat….方法,只要将byte[]转换为 ByteBuffer 就可以实现所有 primitive 类型的数据读取,参见javadoc。

/\*\*

* + 利用 {@link java.nio.ByteBuffer}实现 byte[]转 long
  + @param input
  + @param offset
  + @param littleEndian 输入数组是否小端模式
  + @return

\*/

public static long bytesTolong(byte[] input, int offset, Boolean littleEndian) {

// 将 byte[] 封装为 ByteBuffer

ByteBuffer buffer = ByteBuffer.wrap(input,offset,8);

if(littleEndian){

// ByteBuffer.order(ByteOrder) 方法指定字节序,即大小端模式

(BIG\_ENDIAN/LITTLE\_ENDIAN)

// ByteBuffer 默认为大端(BIG\_ENDIAN)模式

buffer.order(ByteOrder.LITTLE\_ENDIAN);

}

return buffer.getlong();

}复制代码

### 借助 java.io.DataInputStream 实现

java.io.DataInputStream 同样提供了 readLong,readLong,readLong….方法, 只要将 byte[]转换为 DataInputStream 就可以实现所有 primitive 类型的数据读取,参见 javadoc。

## 20、我们能将 int 强制转换为 byte 类型的变量吗？如果该值大于 byte 类型的范围，将会出现什么现象？

是的，我们可以做强制转换，但是 Java 中 int 是 32 位的，而 byte 是 8 位的，所以，如果强制转化是，int 类型的高 24 位将会被丢弃，byte 类型的范围是从 -128 到 127。

## 21、存在两个类，B 继承 A，C 继承 B，我们能将 B 转换为 C 么？如 C = (C) B；

这属于强制类型转换，如果被转换的 B 实例不是 C 类型，会有异常比如你的 ABC 分别对应动物，猫，黑猫。

向上转型就是比如C c = new C();

B b = c;

你把 c 转型为 B，黑猫是猫吗？是啊，所以这是 ok 的。但是反过来

B b = new B(); C c = (C)b;

这就不 ok 了，只知道这个 b 是一只猫，他不一定是黑猫。但如果这个 b 已经确定是一只黑猫了，那就可以转型了

B b = new C(); C c = (C)b;

这里的 b 本来就是黑猫啊。

## 22、哪个类包含 clone 方法？是 Cloneable 还是Object？

java.lang.Cloneable 是一个标示性接口，不包含任何方法，clone 方法在object 类中定义。并且需要知道 clone() 方法是一个本地方法，这意味着它是由 c 或 c++ 或 其他本地语言实现的。

## 23、Java 中 ++ 操作符是线程安全的吗？

不是线程安全的操作。它涉及到多个指令，如读取变量值，增加，然后存储回内存，这个过程可能会出现多个线程交差。

## 24、a = a + b 与 a += b 的区别

+= 隐式的将加操作的结果类型强制转换为持有结果的类型。如果两这个整型相加，如 byte、short 或者 int，首先会将它们提升到 int 类型，然后在执行加法操作。如果加法操作的结果比 a 的最大值要大，则 a+b 会出现编译错误，但是

byte a = 127;byte b = 127;

b = a + b;// error : cannot convert from int to byte

b += a;// ok 复制代码

（译者注：这个地方应该表述的有误，其实无论 a+b 的值为多少，编译器都会报错，因为 a+b 操作会将 a、b 提升为 int 类型，所以将 int 类型赋值给 byte 就会编译出错）

## 25、我能在不进行强制转换的情况下将一个 double 值赋值给 long 类型的变量吗？

不行，你不能在没有强制类型转换的前提下将一个 double 值赋值给 long 类型的变量，因为 double 类型的范围比 long 类型更广，所以必须要进行强制转换。

## 26、3\*0.1 == 0.3 将会返回什么？true 还是false？

false，因为有些浮点数不能完全精确的表示出来。

## 27、int 和 Integer 哪个会占用更多的内存？

Integer 对象会占用更多的内存。Integer 是一个对象，需要存储对象的元数据。但是 int 是一个原始类型的数据，所以占用的空间更少。

## 28、为什么 Java 中的 String 是不可变的

**（Immutable）？**

Java 中的 String 不可变是因为 Java 的设计者认为字符串使用非常频繁，将字符串设置为不可变可以允许多个客户端之间共享相同的字符串。

## 29、我们能在 Switch 中使用 String 吗？

从 Java 7 开始，我们可以在 switch case 中使用字符串，但这仅仅是一个语法糖。内部实现在 switch 中使用字符串的 hash code。

## 30、Java 中的构造器链是什么？

当你从一个构造器中调用另一个构造器，就是 Java 中的构造器链。这种情况只在重载了类的构造器的时候才会出现。

## 31、64 位 JVM 中，int 的长度是多数？

Java 中，int 类型变量的长度是一个固定值，与平台无关，都是 32 位。意思就是说，在 32 位和 64 位的 java 虚拟机中，int 类型的长度是相同的。

## 32、Serial 与 Parallel GC 之间的不同之处？

Serial 与 Parallel 在 GC 执行的时候都会引起 stop-the-world。它们之间主要不同 serial 收集器是默认的复制收集器，执行 GC 的时候只有一个线程，而parallel 收集器使用多个 GC 线程来执行。

## 33、32 位和 64 位的 JVM，int 类型变量的长度是多数？

32 位和 64 位的 JVM 中，int 类型变量的长度是相同的，都是 32 位或者 4 个字节。

## 34、Java 中 WeakReference 与 SoftReference 的区别？

虽然 WeakReference 与 SoftReference 都有利于提高 GC 和 内存的效率，但是 WeakReference ，一旦失去最后一个强引用，就会被 GC 回收，而软引用虽然不能阻止被回收，但是可以延迟到 JVM 内存不足的时候。

## 35、WeakHashMap 是怎么工作的？

WeakHashMap 的工作与正常的 HashMap 类似，但是使用弱引用作为 key，意思就是当 key 对象没有任何引用时，key/value 将会被回收。

## 36、JVM 选项 -XX:+UseCompressedOops 有什么作用？为什么要使用？

当你将你的应用从 32 位的 JVM 迁移到 64 位的 JVM 时，由于对象的指针从32 位增加到了 64 位，因此堆内存会突然增加，差不多要翻倍。这也会对 CPU 缓存（容量比内存小很多）的数据产生不利的影响。因为，迁移到 64 位的 JVM 主要动机在于可以指定最大堆大小，通过压缩 OOP 可以节省一定的内存。通过

-XX:+UseCompressedOops 选项，JVM 会使用 32 位的 OOP，而不是 64 位的OOP。

## 37、怎样通过 Java 程序来判断 JVM 是 32 位 还是

**64 位？**

你可以检查某些系统属性如 sun.arch.data.model 或 os.arch 来获取该信息。

## 38、32 位 JVM 和 64 位 JVM 的最大堆内存分别是多数？

理论上说上 32 位的 JVM 堆内存可以到达 2^32，即 4GB，但实际上会比这个小很多。不同操作系统之间不同，如 Windows 系统大约 1.5 GB，Solaris 大约3GB。64 位 JVM 允许指定最大的堆内存，理论上可以达到 2^64，这是一个非常大的数字，实际上你可以指定堆内存大小到 100GB。甚至有的 JVM，如 Azul， 堆内存到 1000G 都是可能的。

## 39、JRE、JDK、JVM 及 JIT 之间有什么不同？

JRE 代表 Java 运行 时（Java run-time），是 运 行 Java 引用所必须的。JDK 代表 Java 开发工具（Java development kit），是 Java 程序的开发工具，如

Java 编译器，它也包含 JRE。JVM 代表 Java 虚拟机（Java virtual machine），它的责任是运行 Java 应用。JIT 代表即时编译（Just In Time compilation）， 当代码执行的次数超过一定的阈值时，会将 Java 字节码转换为本地代码，如， 主要的热点代码会被准换为本地代码，这样有利大幅度提高 Java 应用的性能。

## 40、解释 Java 堆空间及 GC？

当通过 Java 命令启动 Java 进程的时候，会为它分配内存。内存的一部分用于创建堆空间，当程序中创建对象的时候，就从对空间中分配内存。GC 是 JVM 内部的一个进程，回收无效对象的内存用于将来的分配。

## 41、你能保证 GC 执行吗？

不能，虽然你可以调用 System.gc() 或者 Runtime.gc()，但是没有办法保证 GC的执行。

## 42、怎么获取 Java 程序使用的内存？堆使用的百分比？

可以通过 java.lang.Runtime 类中与内存相关方法来获取剩余的内存，总内存及最大堆内存。通过这些方法你也可以获取到堆使用的百分比及堆内存的剩余空间。Runtime.freeMemory() 方法返回剩余空间的字节数，Runtime.totalMemory()方法总内存的字节数，Runtime.maxMemory() 返回最大内存的字节数。

## 43、Java 中堆和栈有什么区别？

JVM 中堆和栈属于不同的内存区域，使用目的也不同。栈常用于保存方法帧和局部变量，而对象总是在堆上分配。栈通常都比堆小，也不会在多个线程之间共享， 而堆被整个 JVM 的所有线程共享。

## 44、“a==b”和”a.equals(b)”有什么区别？

如果 a 和 b 都是对象，则 a==b 是比较两个对象的引用，只有当 a 和 b 指向的是堆中的同一个对象才会返回 true，而 a.equals(b) 是进行逻辑比较，所以通常需要重写该方法来提供逻辑一致性的比较。例如，String 类重写 equals() 方法，所以可以用于两个不同对象，但是包含的字母相同的比较。

## 45、a.hashCode() 有什么用？与 a.equals(b) 有什么关系？

hashCode() 方法是相应对象整型的 hash 值。它常用于基于 hash 的集合类， 如 Hashtable、HashMap、LinkedHashMap 等等。它与 equals() 方法关系特别紧密。根据 Java 规范，两个使用 equal() 方法来判断相等的对象，必须具有相同的 hash code。

## 46、final、finalize 和 finally 的不同之处？

final 是一个修饰符，可以修饰变量、方法和类。如果 final 修饰变量，意味着该变量的值在初始化后不能被改变。finalize 方法是在对象被回收之前调用的方法，给对象自己最后一个复活的机会，但是什么时候调用 finalize 没有保证。finally 是一个关键字，与 try 和 catch 一起用于异常的处理。finally 块一定会被执行，无论在 try 块中是否有发生异常。

## 47、Java 中的编译期常量是什么？使用它又什么风险？

公共静态不可变（public static final ）变量也就是我们所说的编译期常量， 这里的 public 可选的。实际上这些变量在编译时会被替换掉，因为编译器知道这些变量的值，并且知道这些变量在运行时不能改变。这种方式存在的一个问题是你使用了一个内部的或第三方库中的公有编译时常量，但是这个值后面被其他人改变了，但是你的客户端仍然在使用老的值，甚至你已经部署了一个新的 jar。为了避免这种情况，当你在更新依赖 JAR 文件时，确保重新编译你的程序。

## 48、List、Set、Map 和 Queue 之间的区别(答案)

List 是一个有序集合，允许元素重复。它的某些实现可以提供基于下标值的常量访问时间，但是这不是 List 接口保证的。Set 是一个无序集合。

## 49、poll() 方法和 remove() 方法的区别？

poll() 和 remove() 都是从队列中取出一个元素，但是 poll() 在获取元素失败的时候会返回空，但是 remove() 失败的时候会抛出异常。

## 50、Java 中 LinkedHashMap 和 PriorityQueue 的区别是什么？

PriorityQueue 保证最高或者最低优先级的的元素总是在队列头部，但是LinkedHashMap 维持的顺序是元素插入的顺序。当遍历一个 PriorityQueue 时， 没有任何顺序保证，但是 LinkedHashMap 课保证遍历顺序是元素插入的顺序。

## 51、ArrayList 与 LinkedList 的不区别？

最明显的区别是 ArrrayList 底层的数据结构是数组，支持随机访问，而LinkedList 的底层数据结构书链表，不支持随机访问。使用下标访问一个元素， ArrayList 的时间复杂度是 O(1)，而 LinkedList 是 O(n)。

## 52、用哪两种方式来实现集合的排序？

你可以使用有序集合，如 TreeSet 或 TreeMap，你也可以使用有顺序的的集合， 如 list，然后通过 Collections.sort() 来排序。

## 53、Java 中怎么打印数组？

你可以使用 Arrays.toString() 和 Arrays.deepToString() 方法来打印数组。由于数组没有实现 toString() 方法，所以如果将数组传递给System.out.println()方法，将无法打印出数组的内容，但是 Arrays.toString() 可以打印每个元素。

**54、Java 中的 LinkedList 是单向链表还是双向链表？**

是双向链表，你可以检查 JDK 的源码。在 Eclipse，你可以使用快捷键 Ctrl + T，直接在编辑器中打开该类。

## 55、Java 中的 TreeMap 是采用什么树实现的？

Java 中的 TreeMap 是使用红黑树实现的。

## 56、Hashtable 与 HashMap 有什么不同之处？

这两个类有许多不同的地方，下面列出了一部分：

a) Hashtable 是 JDK 1 遗留下来的类，而 HashMap 是后来增加的。b）Hashtable 是同步的，比较慢，但 HashMap 没有同步策略，所以会更快。c）Hashtable 不允许有个空的 key，但是 HashMap 允许出现一个 null key。

## 57、Java 中的 HashSet，内部是如何工作的？

HashSet 的内部采用 HashMap 来实现。由于 Map 需要 key 和 value，所以所有 key 的都有一个默认 value。类似于 HashMap，HashSet 不允许重复的 key， 只允许有一个 null key，意思就是 HashSet 中只允许存储一个 null 对象。

## 58、写一段代码在遍历 ArrayList 时移除一个元素？

该问题的关键在于面试者使用的是 ArrayList 的 remove() 还是 Iterator 的remove()方法。这有一段示例代码，是使用正确的方式来实现在遍历的过程中移除元素，而不会出现 ConcurrentModificationException 异常的示例代码。

## 59、我们能自己写一个容器类，然后使用 for-each 循环码？

可以，你可以写一个自己的容器类。如果你想使用 Java 中增强的循环来遍历， 你只需要实现 Iterable 接口。如果你实现 Collection 接口，默认就具有该属性。

## 60、ArrayList 和 HashMap 的默认大小是多数？

在 Java 7 中，ArrayList 的默认大小是 10 个元素，HashMap 的默认大小是16 个元素（必须是 2 的幂）。这就是 Java 7 中 ArrayList 和 HashMap 类的代码片段：

// from ArrayList.java JDK 1.7

private static final int DEFAULT\_CAPACITY = 10;

//from HashMap.java JDK 7

static final int DEFAULT\_INITIAL\_CAPACITY = 1 << 4;

// aka 16 复制代码

## 61、有没有可能两个不相等的对象有有相同的hashcode？

有可能，两个不相等的对象可能会有相同的 hashcode 值，这就是为什么在hashmap 中会有冲突。相等 hashcode 值的规定只是说如果两个对象相等，必须有相同的 hashcode 值，但是没有关于不相等对象的任何规定。

## 62、两个相同的对象会有不同的的 hash code 吗？

不能，根据 hash code 的规定，这是不可能的。

## 63、我们可以在 hashcode() 中使用随机数字吗？

不行，因为对象的 hashcode 值必须是相同的。参见答案获取更多关于 Java 中重写 hashCode() 方法的知识。

## 64、Java 中，Comparator 与 Comparable 有什么不同？

Comparable 接口用于定义对象的自然顺序，而 comparator 通常用于定义用户定制的顺序。Comparable 总是只有一个，但是可以有多个 comparator 来定义对象的顺序。

## 65、为什么在重写 equals 方法的时候需要重写hashCode 方法？

因为有强制的规范指定需要同时重写 hashcode 与 equal 是方法，许多容器类， 如 HashMap、HashSet 都依赖于 hashcode 与 equals 的规定。

## 66、在我 Java 程序中，我有三个 socket，我需要多少个线程来处理？

这个需要看你是并行处理还是串行处理了。

## 67、Java 中怎么创建 ByteBuffer？

byte[] bytes = new byte[10];

ByteBuffer buf = ByteBuffer.wrap(bytes);复制代码

## 68、Java 中，怎么读写 ByteBuffer ？

### 69、Java 采用的是大端还是小端？ 70、ByteBuffer 中的字节序是什么？

**71、Java 中，直接缓冲区与非直接缓冲器有什么区别？**

**72、Java 中的内存映射缓存区是什么？**

**73、socket 选项 TCP NO DELAY 是指什么？**

**74、TCP 协议与 UDP 协议有什么区别？**

**75、Java 中，ByteBuffer 与 StringBuffer 有什么区别？** **76、Java 中，编写多线程程序的时候你会遵循哪些最佳实践？**

1. 给线程命名，这样可以帮助调试。
2. 最小化同步的范围，而不是将整个方法同步，只对关键部分做同步。c）如果可以，更偏向于使用 volatile 而不是 synchronized。
3. 使用更高层次的并发工具，而不是使用 wait() 和 notify() 来实现线程间通信，如 BlockingQueue，CountDownLatch 及 Semeaphore。
4. 优先使用并发集合，而不是对集合进行同步。并发集合提供更好的可扩展性。

### 77、说出几点 Java 中使用 Collections 的最佳实践

1. 使用正确的集合类，例如，如果不需要同步列表，使用 ArrayList 而不是Vector。
2. 优先使用并发集合，而不是对集合进行同步。并发集合提供更好的可扩展性。 c）使用接口代表和访问集合，如使用 List 存储 ArrayList，使用 Map 存储 HashMap 等等。

d）使用迭代器来循环集合。e）使用集合的时候使用泛型。

### 78、说出至少 5 点在 Java 中使用线程的最佳实践。

这个问题与之前的问题类似，你可以使用上面的答案。对线程来说，你应该： a）对线程命名

b）将线程和任务分离，使用线程池执行器来执行 Runnable 或 Callable。c）使用线程池

### 79、说出 5 条 IO 的最佳实践

IO 对 Java 应用的性能非常重要。理想情况下，你不应该在你应用的关键路径上避免 IO 操作。下面是一些你应该遵循的 Java IO 最佳实践：

a）使用有缓冲区的 IO 类，而不要单独读取字节或字符。b）使用 NIO 和 NIO2

c）在 finally 块中关闭流，或者使用 try-with-resource 语句。d）使用内存映射文件获取更快的 IO。

### 80、列出 5 个应该遵循的 JDBC 最佳实践

有很多的最佳实践，你可以根据你的喜好来例举。下面是一些更通用的原则： a）使用批量的操作来插入和更新数据

b）使用 PreparedStatement 来避免 SQL 异常，并提高性能。c）使用数据库连接池

d）通过列名来获取结果集，不要使用列的下标来获取。

### 81、说出几条 Java 中方法重载的最佳实践？

下面有几条可以遵循的方法重载的最佳实践来避免造成自动装箱的混乱。

1. 不要重载这样的方法：一个方法接收 int 参数，而另个方法接收 Integer 参数。
2. 不要重载参数数量一致，而只是参数顺序不同的方法。c）如果重载的方法参数个数多于 5 个，采用可变参数。

### 82、在多线程环境下，SimpleDateFormat 是线程安全的吗？

不是，非常不幸，DateFormat 的所有实现，包括 SimpleDateFormat 都不是线程安全的，因此你不应该在多线程序中使用，除非是在对外线程安全的环境中使用，如 将 SimpleDateFormat 限制在 ThreadLocal 中。如果你不这么做，在解析或者格式化日期的时候，可能会获取到一个不正确的结果。因此，从日期、时间处理的所有实践来说，我强力推荐 joda-time 库。

### 83、Java 中如何格式化一个日期？如格式化为 ddMMyyyy 的形式？

Java 中，可以使用 SimpleDateFormat 类或者 joda-time 库来格式日期。DateFormat 类允许你使用多种流行的格式来格式化日期。参见答案中的示例代码，代码中演示了将日期格式化成不同的格式，如 dd-MM-yyyy 或 ddMMyyyy。

### 84、Java 中，怎么在格式化的日期中显示时区？

使用 SimpleDateFormat 来实现格式化日期，但是 DateFormat 的所有实现，包括 SimpleDateFormat 都不是线程安全的，因此你不应该在多线程序中使用，除非是在对外线程安全的环境中使用，如将 SimpleDateFormat 限制在ThreadLocal 中。如果你不这么做，在解析或者格式化日期的时候，可能会获取到一个不正确的结果。因此，从日期、时间处理的所有实践来说，强力推荐joda-time 库。

### 85、Java 中 java.util.Date 与 java.sql.Date 有什么区别？ 86、Java 中，如何计算两个日期之间的差距？

**87、Java 中，如何将字符串 YYYYMMDD 转换为日期？**

**89、如何测试静态方法？**

可以使用 PowerMock 库来测试静态方法。

### 90、怎么利用 JUnit 来测试一个方法的异常？

对需要测试异常的代码使用 try，catch 语句块。比如：public void testException(){try{Long.parseLong(null);} catch(NumberFormatException expected){}}然后使用 Junit 的 fail 函数，对于没有抛出预期异常的代码

### 91、你使用过哪个单元测试库来测试你的 Java 程序？

**92、@Before 和 @BeforeClass 有什么区别？**

@Before ：在每个测试方法之前都执行一次, 方法需要声明为 public @beforeclass ：只在类中执行一次, 必须声明为 public static

**93、怎么检查一个字符串只包含数字？解决方案**用 Java 自带的函数、用正则表达式、用 ascii 码判断**94、Java 中如何利用泛型写一个 LRU 缓存？**

这是一种混合的数据结构，我们需要在哈希表的基础上建立一个链表。但是 Java

已经为我们提供了这种形式的数据结构-LinkedHashMap！它甚至提供可覆盖回收策略的方法。唯一需要我们注意的事情是，改链表的顺序是插入的顺序，而不是访问的顺序。但是，有一个构造函数提供了一个选项，可以使用访问的顺序。

### 95、写一段 Java 程序将 byte 转换为 long？

**96、在不使用 StringBuffer 的前提下，怎么反转一个字符串？**

**97、Java 中，怎么获取一个文件中单词出现的最高频率？**

这是一道算法面试题，Java 中文的比较多。1、将文件内容存入 String 字符串中。

2、利用 split()函数分割字符串，因为直接替换英文空格或者,逗号分隔就可以了，中文类似，分隔得到一个数组。

3、遍历数组中所有的单词，统计结果 Map 中,key=单词，value=单词出现的次数。

4、使用 TreeSet 类型，对 Map 中的结果进行排序，依据统计次数。

5、输出最高的排序的前 N 名结果

### 98、如何检查出两个给定的字符串是反序的？

思路主要是，从开始字符，和另外一个从末尾字符比较，先判断长度是否相同， 不同直接不可能反文。然后再比较。

### 99、Java 中，怎么打印出一个字符串的所有排列？

**100、Java 中，怎样才能打印出数组中的重复元素？**

**101、Java 中如何将字符串转换为整数？**

String s="123"; int i;

第一种方法：i=Integer.parseInt(s);

第二种方法：i=Integer.valueOf(s).intValue();

### 102、在没有使用临时变量的情况如何交换两个整数变量的值？

加减法、乘除法、异或法

### 103、接口是什么？为什么要使用接口而不是直接使用具体类？

接口用于定义 API。它定义了类必须得遵循的规则。同时，它提供了一种抽象， 因为客户端只使用接口，这样可以有多重实现，如 List 接口，你可以使用可随机访问的 ArrayList，也可以使用方便插入和删除的 LinkedList。接口中不允许写代码，以此来保证抽象，但是 Java 8 中你可以在接口声明静态的默认方法， 这种方法是具体的。

### 104、Java 中，抽象类与接口之间有什么不同？

Java 中，抽象类和接口有很多不同之处，但是最重要的一个是 Java 中限制一个类只能继承一个类，但是可以实现多个接口。抽象类可以很好的定义一个家族类的默认行为，而接口能更好的定义类型，有助于后面实现多态机制。

### 105、除了单例模式，你在生产环境中还用过什么设计模式？

这需要根据你的经验来回答。一般情况下，你可以说依赖注入，工厂模式，装饰模式或者观察者模式，随意选择你使用过的一种即可。不过你要准备回答接下的基于你选择的模式的问题。

### 106、你能解释一下里氏替换原则吗?

首先，这是编译器的要求，如果不这么做，无法通过编译。其次，面向对象的编程，其中继承有个大原则，任何子类的对象都可以当成父类的对象使用。

### 107、什么情况下会违反迪米特法则？为什么会有这个问题？

迪米特法则建议“只和朋友说话，不要陌生人说话”，以此来减少类之间的耦合。

### 108、适配器模式是什么？什么时候使用？

适配器模式提供对接口的转换。如果你的客户端使用某些接口，但是你有另外一些接口，你就可以写一个适配去来连接这些接口。

### 109、什么是“依赖注入”和“控制反转”？为什么有人使用？

控制反转（IOC）是 Spring 框架的核心思想，用我自己的话说，就是你要做一件事，别自己可劲 new 了，你就说你要干啥，然后外包出去就好~依赖注入（DI） 在我浅薄的想法中，就是通过接口的引用和构造方法的表达，将一些事情整好了反过来传给需要用到的地方~

### 110、抽象类是什么？它与接口有什么区别？你为什么要使用过抽象 类？

1. 接口用于规范，抽象类用于共性.
2. 声明方法的存在而不去实现它的类被叫做抽象类
3. 接口（interface）是抽象类的变体。在接口中，所有方法都是抽象的。

### 111、构造器注入和 setter 依赖注入，那种方式更好？

每种方式都有它的缺点和优点。构造器注入保证所有的注入都被初始化，但是setter 注入提供更好的灵活性来设置可选依赖。如果使用 XML 来描述依赖， Setter 注入的可读写会更强。经验法则是强制依赖使用构造器注入，可选依赖使用 setter 注入。

### 112、依赖注入和工程模式之间有什么不同？

虽然两种模式都是将对象的创建从应用的逻辑中分离，但是依赖注入比工程模式更清晰。通过依赖注入，你的类就是 POJO，它只知道依赖而不关心它们怎么获取。使用工厂模式，你的类需要通过工厂来获取依赖。因此，使用 DI 会比使用工厂模式更容易测试。

### 113、适配器模式和装饰器模式有什么区别？

虽然适配器模式和装饰器模式的结构类似，但是每种模式的出现意图不同。适配器模式被用于桥接两个接口，而装饰模式的目的是在不修改类的情况下给类增加新的功能。

### 114、适配器模式和代理模式之前有什么不同？

这个问题与前面的类似，适配器模式和代理模式的区别在于他们的意图不同。由于适配器模式和代理模式都是封装真正执行动作的类，因此结构是一致的，但是适配器模式用于接口之间的转换，而代理模式则是增加一个额外的中间层，以便支持分配、控制或智能访问。

### 115、什么是模板方法模式？

模板方法提供算法的框架，你可以自己去配置或定义步骤。例如，你可以将排序算法看做是一个模板。它定义了排序的步骤，但是具体的比较，可以使用Comparable 或者其语言中类似东西，具体策略由你去配置。列出算法概要的方法就是众所周知的模板方法。

### 116、什么时候使用访问者模式？

访问者模式用于解决在类的继承层次上增加操作，但是不直接与之关联。这种模式采用双派发的形式来增加中间层。

### 117、什么时候使用组合模式？

组合模式使用树结构来展示部分与整体继承关系。它允许客户端采用统一的形式来对待单个对象和对象容器。当你想要展示对象这种部分与整体的继承关系时采用组合模式。

### 118、继承和组合之间有什么不同？

虽然两种都可以实现代码复用，但是组合比继承共灵活，因为组合允许你在运行时选择不同的实现。用组合实现的代码也比继承测试起来更加简单。

### 119、描述 Java 中的重载和重写？

重载和重写都允许你用相同的名称来实现不同的功能，但是重载是编译时活动， 而重写是运行时活动。你可以在同一个类中重载方法，但是只能在子类中重写方法。重写必须要有继承。

### 120、Java 中，嵌套公共静态类与顶级类有什么不同？

类的内部可以有多个嵌套公共静态类，但是一个 Java 源文件只能有一个顶级公共类，并且顶级公共类的名称与源文件名称必须一致。

### 121、 OOP 中的 组合、聚合和关联有什么区别？

如果两个对象彼此有关系，就说他们是彼此相关联的。组合和聚合是面向对象中的两种形式的关联。组合是一种比聚合更强力的关联。组合中，一个对象是另一个的拥有者，而聚合则是指一个对象使用另一个对象。如果对象 A 是由对象 B 组合的，则 A 不存在的话，B 一定不存在，但是如果 A 对象聚合了一个对象 B， 则即使 A 不存在了，B 也可以单独存在。

### 122、给我一个符合开闭原则的设计模式的例子？

开闭原则要求你的代码对扩展开放，对修改关闭。这个意思就是说，如果你想增加一个新的功能，你可以很容易的在不改变已测试过的代码的前提下增加新的代码。有好几个设计模式是基于开闭原则的，如策略模式，如果你需要一个新的策略，只需要实现接口，增加配置，不需要改变核心逻辑。一个正在工作的例子是Collections.sort() 方法，这就是基于策略模式，遵循开闭原则的，你不需为新的对象修改 sort() 方法，你需要做的仅仅是实现你自己的 Comparator 接口。

### 123、抽象工厂模式和原型模式之间的区别？

抽象工厂模式：通常由工厂方法模式来实现。但一个工厂中往往含有多个工厂方法生成一系列的产品。这个模式强调的是客户代码一次保证只使用一个系列的产品。当要切换为另一个系列的产品，换一个工厂类即可。原型模式：工厂方法的最大缺点就是，对应一个继承体系的产品类，要有一个同样复杂的工厂类的继承体系。我们可以把工厂类中的工厂方法放到产品类自身之中吗？如果这样的话， 就可以将两个继承体系为一个。这也就是原型模式的思想，原型模式中的工厂方

法为 clone，它会返回一个拷贝（可以是浅拷贝，也可以是深拷贝，由设计者决定）。为了保证用户代码中到时可以通过指针调用 clone 来动态绑定地生成所需的具体的类。这些原型对象必须事先构造好。原型模式想对工厂方法模式的另一个好处是，拷贝的效率一般对构造的效率要高。

### 124、什么时候使用享元模式？

享元模式通过共享对象来避免创建太多的对象。为了使用享元模式，你需要确保你的对象是不可变的，这样你才能安全的共享。JDK 中 String 池、Integer 池以及 Long 池都是很好的使用了享元模式的例子。

### 125、嵌套静态类与顶级类有什么区别？

一个公共的顶级类的源文件名称与类名相同，而嵌套静态类没有这个要求。一个嵌套类位于顶级类内部，需要使用顶级类的名称来引用嵌套静态类，如HashMap.Entry 是一个嵌套静态类，HashMap 是一个顶级类，Entry 是一个嵌套静态类。

### 126、你能写出一个正则表达式来判断一个字符串是否是一个数字吗？

一个数字字符串，只能包含数字，如 0 到 9 以及 +、- 开头，通过这个信息， 你可以下一个如下的正则表达式来判断给定的字符串是不是数字。 首先要import java.util.regex.Pattern 和 java.util.regex.Matcher

public Boolean isNumeric(String str){

Pattern pattern = Pattern.compile("[0-9]\*"); Matcher isNum = pattern.matcher(str);

if( !isNum.matches() ){

return false;

}

return true;

}复制代码

### 127、Java 中，受检查异常 和 不受检查异常的区别？

受检查异常编译器在编译期间检查。对于这种异常，方法强制处理或者通过throws 子句声明。其中一种情况是 Exception 的子类但不是RuntimeException 的子类。非受检查是 RuntimeException 的子类，在编译阶段不受编译器的检查。

### 128、Java 中，throw 和 throws 有什么区别

throw 用于抛出 java.lang.Throwable 类的一个实例化对象，意思是说你可以通过关键字 throw 抛出一个 Error 或者 一个 Exception，如：throw new IllegalArgumentException(“size must be multiple of 2″)而 throws 的作用是作为方法声明和签名的一部分，方法被抛出相应的异常以便调用者能处理。Java 中，任何未处理的受检查异常强制在 throws 子句中声明。

### 129、Java 中，Serializable 与 Externalizable 的区别？

Serializable 接口是一个序列化 Java 类的接口，以便于它们可以在网络上传输或者可以将它们的状态保存在磁盘上，是 JVM 内嵌的默认序列化方式，成本高、脆弱而且不安全。Externalizable 允许你控制整个序列化过程，指定特定的二进制格式，增加安全机制。

### 130、Java 中，DOM 和 SAX 解析器有什么不同？

DOM 解析器将整个 XML 文档加载到内存来创建一棵 DOM 模型树，这样可以更快的查找节点和修改 XML 结构，而 SAX 解析器是一个基于事件的解析器，不会将整个 XML 文档加载到内存。由于这个原因，DOM 比 SAX 更快，也要求更多的内存，不适合于解析大 XML 文件。

### 131、说出 JDK 1.7 中的三个新特性？

虽然 JDK 1.7 不像 JDK 5 和 8 一样的大版本，但是，还是有很多新的特性， 如 try-with-resource 语句，这样你在使用流或者资源的时候，就不需要手动关闭，Java 会自动关闭。Fork-Join 池某种程度上实现 Java 版的 Map-reduce。允许 Switch 中有 String 变量和文本。菱形操作符(<>)用于类型推断，不再需要在变量声明的右边申明泛型，因此可以写出可读写更强、更简洁的代码。另一个值得一提的特性是改善异常处理，如允许在同一个 catch 块中捕获多个异常。

### 132、说出 5 个 JDK 1.8 引入的新特性？

Java 8 在 Java 历史上是一个开创新的版本，下面 JDK 8 中 5 个主要的特性： (1)Lambda 表达式,允许像对象一样传递匿名函数

1. Stream API,充分利用现代多核 CPU，可以写出很简洁的代码
2. Date 与 Time API,最终，有一个稳定、简单的日期和时间库可供你使用(4)扩展方法，现在，接口中可以有静态、默认方法。

(5)重复注解，现在你可以将相同的注解在同一类型上使用多次。

### 133、Java 中，Maven 和 ANT 有什么区别？

虽然两者功能上都是构建工具，都用于创建 Java 应用，但是 Maven 做的事情更多，在基于“约定优于配置”的概念下，提供标准的 Java 项目结构，同时能为应用自动管理依赖（应用中所依赖的 JAR 文件）。Ant 仅仅是软件构建工具， 而 Maven 的定位是软件项目管理和理解工具。Maven 除了具备 Ant 的功能外，有以下主要的功能：

1. 使用 Project Object Model 来对软件项目管理；
2. 内置了更多的隐式规则，使得构建文件更加简单；
3. 内置依赖管理和 Repository 来实现依赖的管理和统一存储；
4. 内置了软件构建的生命周期；

# 二、Java 并发系列面试题

## 1、在 java 中守护线程和本地线程区别？

java 中的线程分为两种：守护线程（Daemon）和用户线程（User）。任何线程都可以设置为守护线程和用户线程，通过方法

Thread.setDaemon(boolon)；true 则把该线程设置为守护线程，反之则为用户线程。Thread.setDaemon()必须在 Thread.start()之前调用，否则运行时会抛出异常。

两者的区别：

唯一的区别是判断虚拟机(JVM)何时离开，Daemon 是为其他线程提供服务，如果全部的 User Thread 已经撤离，Daemon 没有可服务的线程，JVM 撤离。也可以理解为守护线程是 JVM 自动创建的线程（但不一定），用户线程是程序创建的线程；比如 JVM 的垃圾回收线程是一个守护线程，当所有线程已经撤离，不再产生垃圾，守护线程自然就没事可干了，当垃圾回收线程是 Java 虚拟机上仅剩的线程时，Java 虚拟机会自动离开。

扩展：Thread Dump 打印出来的线程信息，含有 daemon 字样的线程即为守护进程，可能会有：服务守护进程、编译守护进程、windows 下的监听 Ctrl+break 的守护进程、Finalizer 守护进程、引用处理守护进程、GC 守护进程。

## 2、线程与进程的区别？

进程是操作系统分配资源的最小单元，线程是操作系统调度的最小单元。一个程序至少有一个进程,一个进程至少有一个线程。

## 3、什么是多线程中的上下文切换？

多线程会共同使用一组计算机上的 CPU，而线程数大于给程序分配的 CPU 数量时，为了让各个线程都有执行的机会，就需要轮转使用 CPU。不同的线程切换使用 CPU 发生的切换数据等就是上下文切换。

## 4、死锁与活锁的区别，死锁与饥饿的区别？

死锁：是指两个或两个以上的进程（或线程）在执行过程中，因争夺资源而造成的一种互相等待的现象，若无外力作用，它们都将无法推进下去。

产生死锁的必要条件：

1、互斥条件：所谓互斥就是进程在某一时间内独占资源。

2、请求与保持条件：一个进程因请求资源而阻塞时，对已获得的资源保持不放。

3、不剥夺条件:进程已获得资源，在末使用完之前，不能强行剥夺。

4、循环等待条件:若干进程之间形成一种头尾相接的循环等待资源关系。

活锁：任务或者执行者没有被阻塞，由于某些条件没有满足，导致一直重复尝试， 失败，尝试，失败。

活锁和死锁的区别在于，处于活锁的实体是在不断的改变状态，所谓的“活”， 而处于死锁的实体表现为等待；活锁有可能自行解开，死锁则不能。

饥饿：一个或者多个线程因为种种原因无法获得所需要的资源，导致一直无法执行的状态。

Java 中导致饥饿的原因：

1、高优先级线程吞噬所有的低优先级线程的 CPU 时间。

2、线程被永久堵塞在一个等待进入同步块的状态，因为其他线程总是能在它之前持续地对该同步块进行访问。

3、线程在等待一个本身也处于永久等待完成的对象(比如调用这个对象的 wait 方法)，因为其他线程总是被持续地获得唤醒。

5、Java 中用到的线程调度算法是什么？

采用时间片轮转的方式。可以设置线程的优先级，会映射到下层的系统上面的优先级上，如非特别需要，尽量不要用，防止线程饥饿。

## 6、什么是线程组，为什么在 Java 中不推荐使用？

ThreadGroup 类，可以把线程归属到某一个线程组中，线程组中可以有线程对象， 也可以有线程组，组中还可以有线程，这样的组织结构有点类似于树的形式。 为什么不推荐使用？因为使用有很多的安全隐患吧，没有具体追究，如果需要使用，推荐使用线程池。

## 7、为什么使用 Executor 框架？

每次执行任务创建线程 new Thread()比较消耗性能，创建一个线程是比较耗时、耗资源的。

调用 new Thread()创建的线程缺乏管理，被称为野线程，而且可以无限制的创建，线程之间的相互竞争会导致过多占用系统资源而导致系统瘫痪，还有线程之间的频繁交替也会消耗很多系统资源。

接使用 new Thread() 启动的线程不利于扩展，比如定时执行、定期执行、定时定期执行、线程中断等都不便实现。

## 8、在 Java 中 Executor 和 Executors 的区别？

Executors 工具类的不同方法按照我们的需求创建了不同的线程池，来满足业务的需求。

Executor 接口对象能执行我们的线程任务。

ExecutorService 接口继承了 Executor 接口并进行了扩展，提供了更多的方法我们能获得任务执行的状态并且可以获取任务的返回值。

使用 ThreadPoolExecutor 可以创建自定义线程池。

Future 表示异步计算的结果，他提供了检查计算是否完成的方法，以等待计算的完成，并可以使用 get()方法获取计算的结果。

## 9、如何在 Windows 和 Linux 上查找哪个线程使用的 CPU 时间最长？

## 10、什么是原子操作？在 Java Concurrency API 中有哪些原子类(atomic classes)？

原子操作（atomic operation）意为”不可被中断的一个或一系列操作” 。

处理器使用基于对缓存加锁或总线加锁的方式来实现多处理器之间的原子操作。在 Java 中可以通过锁和循环 CAS 的方式来实现原子操作。 CAS 操作

——Compare & Set，或是 Compare & Swap，现在几乎所有的 CPU 指令都支持 CAS的原子操作。

原子操作是指一个不受其他操作影响的操作任务单元。原子操作是在多线程环境下避免数据不一致必须的手段。

int++并不是一个原子操作，所以当一个线程读取它的值并加 1 时，另外一个线程有可能会读到之前的值，这就会引发错误。

为了解决这个问题，必须保证增加操作是原子的，在 JDK1.5 之前我们可以使用同步技术来做到这一点。到 JDK1.5，java.util.concurrent.atomic 包提供了int 和 long 类型的原子包装类，它们可以自动的保证对于他们的操作是原子的并且不需要使用同步。

java.util.concurrent 这个包里面提供了一组原子类。其基本的特性就是在多线程环境下，当有多个线程同时执行这些类的实例包含的方法时，具有排他性，

即当某个线程进入方法，执行其中的指令时，不会被其他线程打断，而别的线程就像自旋锁一样，一直等到该方法执行完成，才由 JVM 从等待队列中选择一个另一个线程进入，这只是一种逻辑上的理解。

原子类：AtomicBoolean，AtomicInteger，AtomicLong，AtomicReference 原子数组：AtomicIntegerArray，AtomicLongArray，AtomicReferenceArray 原子属性更新器：AtomicLongFieldUpdater，AtomicIntegerFieldUpdater， AtomicReferenceFieldUpdater

解决 ABA 问题的原子类：AtomicMarkableReference（通过引入一个 boolean 来反映中间有没有变过），AtomicStampedReference（通过引入一个 int 来累加来反映中间有没有变过）

## 11、Java Concurrency API 中的 Lock 接口(Lock interface)是什么？对比同步它有什么优势？

Lock 接口比同步方法和同步块提供了更具扩展性的锁操作。

他们允许更灵活的结构，可以具有完全不同的性质，并且可以支持多个相关类的条件对象。

它的优势有： 可以使锁更公平

可以使线程在等待锁的时候响应中断

可以让线程尝试获取锁，并在无法获取锁的时候立即返回或者等待一段时间可以在不同的范围，以不同的顺序获取和释放锁

整体上来说 Lock 是 synchronized 的扩展版，Lock 提供了无条件的、可轮询的(tryLock 方法)、定时的(tryLock 带参方法)、可中断的(lockInterruptibly)、可多条件队列的(newCondition 方法)锁操作。另外 Lock 的实现类基本都支持非公平锁(默认)和公平锁，synchronized 只支持非公平锁， 当然，在大部分情况下，非公平锁是高效的选择。

## 12、什么是 Executors 框架？

Executor 框架是一个根据一组执行策略调用，调度，执行和控制的异步任务的框架。

无限制的创建线程会引起应用程序内存溢出。所以创建一个线程池是个更好的的解决方案，因为可以限制线程的数量并且可以回收再利用这些线程。利用Executors 框架可以非常方便的创建一个线程池。

## 13、什么是阻塞队列？阻塞队列的实现原理是什么？ 如何使用阻塞队列来实现生产者-消费者模型？

阻塞队列（BlockingQueue）是一个支持两个附加操作的队列。

这两个附加的操作是：在队列为空时，获取元素的线程会等待队列变为非空。当队列满时，存储元素的线程会等待队列可用。

阻塞队列常用于生产者和消费者的场景，生产者是往队列里添加元素的线程，消费者是从队列里拿元素的线程。阻塞队列就是生产者存放元素的容器，而消费者也只从容器里拿元素。

JDK7 提供了 7 个阻塞队列。分别是：

ArrayBlockingQueue ：一个由数组结构组成的有界阻塞队列。LinkedBlockingQueue ：一个由链表结构组成的有界阻塞队列。PriorityBlockingQueue ：一个支持优先级排序的无界阻塞队列。DelayQueue：一个使用优先级队列实现的无界阻塞队列。SynchronousQueue：一个不存储元素的阻塞队列。LinkedTransferQueue：一个由链表结构组成的无界阻塞队列。LinkedBlockingDeque：一个由链表结构组成的双向阻塞队列。

Java 5 之前实现同步存取时，可以使用普通的一个集合，然后在使用线程的协作和线程同步可以实现生产者，消费者模式，主要的技术就是用好，

wait ,notify,notifyAll,sychronized 这些关键字。而在 java 5 之后，可以使用阻塞队列来实现，此方式大大简少了代码量，使得多线程编程更加容易，安全方面也有保障。

BlockingQueue 接口是 Queue 的子接口，它的主要用途并不是作为容器，而是作为线程同步的的工具，因此他具有一个很明显的特性，当生产者线程试图向BlockingQueue 放入元素时，如果队列已满，则线程被阻塞，当消费者线程试图从中取出一个元素时，如果队列为空，则该线程会被阻塞，正是因为它所具有这个特性，所以在程序中多个线程交替向 BlockingQueue 中放入元素，取出元素， 它可以很好的控制线程之间的通信。

阻塞队列使用最经典的场景就是 socket 客户端数据的读取和解析，读取数据的线程不断将数据放入队列，然后解析线程不断从队列取数据解析。

## 14、什么是 Callable 和 Future?

Callable 接口类似于 Runnable，从名字就可以看出来了，但是 Runnable 不会返回结果，并且无法抛出返回结果的异常，而 Callable 功能更强大一些，被线程执行后，可以返回值，这个返回值可以被 Future 拿到，也就是说，Future 可以拿到异步执行任务的返回值。

可以认为是带有回调的 Runnable。

Future 接口表示异步任务，是还没有完成的任务给出的未来结果。所以说Callable 用于产生结果，Future 用于获取结果。

## 15、什么是 FutureTask?使用 ExecutorService 启动任务。

在 Java 并发程序中 FutureTask 表示一个可以取消的异步运算。它有启动和取消运算、查询运算是否完成和取回运算结果等方法。只有当运算完成的时候结果才能取回，如果运算尚未完成 get 方法将会阻塞。一个 FutureTask 对象可以对调用了 Callable 和 Runnable 的对象进行包装，由于 FutureTask 也是调用了 Runnable 接口所以它可以提交给 Executor 来执行。

## 16、什么是并发容器的实现？

何为同步容器：可以简单地理解为通过 synchronized 来实现同步的容器，如果有多个线程调用同步容器的方法，它们将会串行执行。比如 Vector，Hashtable， 以及 Collections.synchronizedSet，synchronizedList 等方法返回的容器。可以通过查看 Vector，Hashtable 等这些同步容器的实现代码，可以看到这些容器实现线程安全的方式就是将它们的状态封装起来，并在需要同步的方法上加上关键字 synchronized。

并发容器使用了与同步容器完全不同的加锁策略来提供更高的并发性和伸缩性， 例如在 ConcurrentHashMap 中采用了一种粒度更细的加锁机制，可以称为分段锁，在这种锁机制下，允许任意数量的读线程并发地访问 map，并且执行读操作的线程和写操作的线程也可以并发的访问 map，同时允许一定数量的写操作线程并发地修改 map，所以它可以在并发环境下实现更高的吞吐量。

## 17、多线程同步和互斥有几种实现方法，都是什么？

线程同步是指线程之间所具有的一种制约关系，一个线程的执行依赖另一个线程的消息，当它没有得到另一个线程的消息时应等待，直到消息到达时才被唤醒。线程互斥是指对于共享的进程系统资源，在各单个线程访问时的排它性。当有若干个线程都要使用某一共享资源时，任何时刻最多只允许一个线程去使用，其它要使用该资源的线程必须等待，直到占用资源者释放该资源。线程互斥可以看成是一种特殊的线程同步。

线程间的同步方法大体可分为两类：用户模式和内核模式。顾名思义，内核模式就是指利用系统内核对象的单一性来进行同步，使用时需要切换内核态与用户态，而用户模式就是不需要切换到内核态，只在用户态完成操作。

用户模式下的方法有：原子操作（例如一个单一的全局变量），临界区。内核模式下的方法有：事件，信号量，互斥量。

## 18、什么是竞争条件？你怎样发现和解决竞争？

当多个进程都企图对共享数据进行某种处理，而最后的结果又取决于进程运行的顺序时，则我们认为这发生了竞争条件（race condition）。

## 19、你将如何使用 thread dump？你将如何分析Thread dump？

新建状态（New）

用 new 语句创建的线程处于新建状态，此时它和其他 Java 对象一样，仅仅在堆区中被分配了内存。

就绪状态（Runnable）

当一个线程对象创建后，其他线程调用它的 start()方法，该线程就进入就绪状态，Java 虚拟机会为它创建方法调用栈和程序计数器。处于这个状态的线程位于可运行池中，等待获得 CPU 的使用权。

运行状态（Running）

处于这个状态的线程占用 CPU，执行程序代码。只有处于就绪状态的线程才有机会转到运行状态。

阻塞状态（Blocked）

阻塞状态是指线程因为某些原因放弃 CPU，暂时停止运行。当线程处于阻塞状态时，Java 虚拟机不会给线程分配 CPU。直到线程重新进入就绪状态，它才有机会转到运行状态。

阻塞状态可分为以下 3 种：

位于对象等待池中的阻塞状态（Blocked in object’s wait pool）：

当线程处于运行状态时，如果执行了某个对象的 wait()方法，Java 虚拟机就会把线程放到这个对象的等待池中，这涉及到“线程通信”的内容。

位于对象锁池中的阻塞状态（Blocked in object’s lock pool）：

当线程处于运行状态时，试图获得某个对象的同步锁时，如果该对象的同步锁已经被其他线程占用，Java 虚拟机就会把这个线程放到这个对象的锁池中，这涉及到“线程同步”的内容。

其他阻塞状态（Otherwise Blocked）：

当前线程执行了 sleep()方法，或者调用了其他线程的 join()方法，或者发出了 I/O 请求时，就会进入这个状态。

死亡状态（Dead）

当线程退出 run()方法时，就进入死亡状态，该线程结束生命周期。

## 20、为什么我们调用 start()方法时会执行 run()方法，为什么我们不能直接调用 run()方法？

当你调用 start()方法时你将创建新的线程，并且执行在 run()方法里的代码。但是如果你直接调用 run()方法，它不会创建新的线程也不会执行调用线程的代码，只会把 run 方法当作普通方法去执行。

## 21、Java 中你怎样唤醒一个阻塞的线程？

在 Java 发展史上曾经使用 suspend()、resume()方法对于线程进行阻塞唤醒， 但随之出现很多问题，比较典型的还是死锁问题。

解决方案可以使用以对象为目标的阻塞，即利用 Object 类的 wait()和notify()方法实现线程阻塞。

首 先 ，wait、notify 方法是针对对象的，调用任意对象的 wait()方法都将导致线程阻塞，阻塞的同时也将释放该对象的锁，相应地，调用任意对象的 notify() 方法则将随机解除该对象阻塞的线程，但它需要重新获取改对象的锁，直到获取成功才能往下执行；其次，wait、notify 方法必须在 synchronized 块或方法中被调用，并且要保证同步块或方法的锁对象与调用 wait、notify 方法的对象是同一个，如此一来在调用 wait 之前当前线程就已经成功获取某对象的锁，执行 wait 阻塞后当前线程就将之前获取的对象锁释放。

## 22、在 Java 中 CycliBarriar 和 CountdownLatch 有什么区别？

CyclicBarrier 可以重复使用，而 CountdownLatch 不能重复使用。

Java 的 concurrent 包里面的 CountDownLatch 其实可以把它看作一个计数器，只不过这个计数器的操作是原子操作，同时只能有一个线程去操作这个计数器，也就是同时只能有一个线程去减这个计数器里面的值。你可以向CountDownLatch 对象设置一个初始的数字作为计数值，任何调用这个对象上的await()方法都会阻塞，直到这个计数器的计数值被其他的线程减为 0 为止。所以在当前计数到达零之前，await 方法会一直受阻塞。之后，会释放所有等待的线程，await 的所有后续调用都将立即返回。这种现象只出现一次——计数无法被重置。如果需要重置计数，请考虑使用 CyclicBarrier。CountDownLatch 的一个非常典型的应用场景是：有一个任务想要往下执行，但必须要等到其他的任务执行完毕后才可以继续往下执行。假如我们这个想要继续往下执行的任务调用一个 CountDownLatch 对象的 await()方法，其他的任务执行完自己的任务后调用同一个 CountDownLatch 对象上的 countDown()方法，这个调用 await()方法

的任务将一直阻塞等待，直到这个 CountDownLatch 对象的计数值减到 0 为止。CyclicBarrier 一个同步辅助类，它允许一组线程互相等待，直到到达某个公共屏障点 (common barrier point)。在涉及一组固定大小的线程的程序中，这些线程必须不时地互相等待，此时 CyclicBarrier 很有用。因为该 barrier 在释放等待线程后可以重用，所以称它为循环 的 barrier。

## 23、什么是不可变对象，它对写并发应用有什么帮助？

不可变对象(Immutable Objects)即对象一旦被创建它的状态（对象的数据，也即对象属性值）就不能改变，反之即为可变对象(Mutable Objects)。

不可变对象的类即为不可变类(Immutable Class)。Java 平台类库中包含许多不可变类，如 String、基本类型的包装类、BigInteger 和 BigDecimal 等。

不可变对象天生是线程安全的。它们的常量（域）是在构造函数中创建的。既然它们的状态无法修改，这些常量永远不会变。

不可变对象永远是线程安全的。

只有满足如下状态，一个对象才是不可变的； 它的状态不能在创建后再被修改；

所有域都是 final 类型；并且，它被正确创建（创建期间没有发生 this 引用的逸出）。

## 24、什么是多线程中的上下文切换？

在上下文切换过程中，CPU 会停止处理当前运行的程序，并保存当前程序运行的具体位置以便之后继续运行。从这个角度来看，上下文切换有点像我们同时阅读几本书，在来回切换书本的同时我们需要记住每本书当前读到的页码。在程序中， 上下文切换过程中的“页码”信息是保存在进程控制块（PCB）中的。PCB 还经常被称作“切换桢”（switchframe）。“页码”信息会一直保存到 CPU 的内存中，直到他们被再次使用。

上下文切换是存储和恢复 CPU 状态的过程，它使得线程执行能够从中断点恢复执行。上下文切换是多任务操作系统和多线程环境的基本特征。

## 25、Java 中用到的线程调度算法是什么？

计算机通常只有一个 CPU,在任意时刻只能执行一条机器指令,每个线程只有获得 CPU 的使用权才能执行指令.所谓多线程的并发运行,其实是指从宏观上看, 各个线程轮流获得 CPU 的使用权,分别执行各自的任务.在运行池中,会有多个处于就绪状态的线程在等待 CPU,JAVA 虚拟机的一项任务就是负责线程的调度, 线程调度是指按照特定机制为多个线程分配 CPU 的使用权.

有两种调度模型：分时调度模型和抢占式调度模型。

分时调度模型是指让所有的线程轮流获得 cpu 的使用权,并且平均分配每个线程占用的 CPU 的时间片这个也比较好理解。

Java 虚拟机采用抢占式调度模型，是指优先让可运行池中优先级高的线程占用CPU，如果可运行池中的线程优先级相同，那么就随机选择一个线程，使其占用CPU。处于运行状态的线程会一直运行，直至它不得不放弃 CPU。

## 26、什么是线程组，为什么在 Java 中不推荐使用？

线程组和线程池是两个不同的概念，他们的作用完全不同，前者是为了方便线程的管理，后者是为了管理线程的生命周期，复用线程，减少创建销毁线程的开销。

## 27、为什么使用 Executor 框架比使用应用创建和管理线程好？

为什么要使用 Executor 线程池框架

1、每次执行任务创建线程 new Thread()比较消耗性能，创建一个线程是比较耗时、耗资源的。

2、调用 new Thread()创建的线程缺乏管理，被称为野线程，而且可以无限制的创建，线程之间的相互竞争会导致过多占用系统资源而导致系统瘫痪，还有线程之间的频繁交替也会消耗很多系统资源。

3、直接使用 new Thread() 启动的线程不利于扩展，比如定时执行、定期执行、定时定期执行、线程中断等都不便实现。

使用 Executor 线程池框架的优点

1、能复用已存在并空闲的线程从而减少线程对象的创建从而减少了消亡线程的开销。

2、可有效控制最大并发线程数，提高系统资源使用率，同时避免过多资源竞争。

3、框架中已经有定时、定期、单线程、并发数控制等功能。

综上所述使用线程池框架 Executor 能更好的管理线程、提供系统资源使用率。

## 28、java 中有几种方法可以实现一个线程？

继承 Thread 类

实现 Runnable 接口

实现 Callable 接口，需要实现的是 call() 方法

## 29、如何停止一个正在运行的线程？

使用共享变量的方式

在这种方式中，之所以引入共享变量，是因为该变量可以被多个执行相同任务的线程用来作为是否中断的信号，通知中断线程的执行。

使用 interrupt 方法终止线程

如果一个线程由于等待某些事件的发生而被阻塞，又该怎样停止该线程呢？这种情况经常会发生，比如当一个线程由于需要等候键盘输入而被阻塞，或者调用Thread.join()方法，或者 Thread.sleep()方法，在网络中调用ServerSocket.accept()方法，或者调用了 DatagramSocket.receive()方法时， 都有可能导致线程阻塞，使线程处于处于不可运行状态时，即使主程序中将该线程的共享变量设置为 true，但该线程此时根本无法检查循环标志，当然也就无法立即中断。这里我们给出的建议是，不要使用 stop()方法，而是使用 Thread 提供的 interrupt()方法，因为该方法虽然不会中断一个正在运行的线程，但是

它可以使一个被阻塞的线程抛出一个中断异常，从而使线程提前结束阻塞状态， 退出堵塞代码。

## 30、notify()和 notifyAll()有什么区别？

当一个线程进入 wait 之后，就必须等其他线程 notify/notifyall,使用notifyall,可以唤醒所有处于 wait 状态的线程，使其重新进入锁的争夺队列中，而 notify 只能唤醒一个。

如果没把握，建议 notifyAll，防止 notigy 因为信号丢失而造成程序异常。

## 31、什么是 Daemon 线程？它有什么意义？

所谓后台(daemon)线程，是指在程序运行的时候在后台提供一种通用服务的线程，并且这个线程并不属于程序中不可或缺的部分。因此，当所有的非后台线程结束时，程序也就终止了，同时会杀死进程中的所有后台线程。反过来说，只要有任何非后台线程还在运行，程序就不会终止。必须在线程启动之前调用setDaemon()方法，才能把它设置为后台线程。注意：后台进程在不执行 finally 子句的情况下就会终止其 run()方法。

比如：JVM 的垃圾回收线程就是 Daemon 线程，Finalizer 也是守护线程。

## 32、java 如何实现多线程之间的通讯和协作？

中断 和 共享变量

## 33、什么是可重入锁（ReentrantLock）？

举例来说明锁的可重入性

public class UnReentrant{

Lock lock = new Lock(); public void outer(){

lock.lock();

inner(); lock.unlock();

}

public void inner(){

lock.lock();

//do something lock.unlock();

}

}复制代码

outer 中调用了 inner，outer 先锁住了 lock，这样 inner 就不能再获取lock。其实调用 outer 的线程已经获取了 lock 锁，但是不能在 inner 中重复利用已经获取的锁资源，这种锁即称之为 不可重入可重入就意味着：线程可以进入任何一个它已经拥有的锁所同步着的代码块。

synchronized、ReentrantLock 都是可重入的锁，可重入锁相对来说简化了并发编程的开发。

## 34、当一个线程进入某个对象的一个 synchronized 的实例方法后，其它线程是否可进入此对象的其它方法？

如果其他方法没有 synchronized 的话，其他线程是可以进入的。

所以要开放一个线程安全的对象时，得保证每个方法都是线程安全的。

## 35、乐观锁和悲观锁的理解及如何实现，有哪些实现方式？

悲观锁：总是假设最坏的情况，每次去拿数据的时候都认为别人会修改，所以每次在拿数据的时候都会上锁，这样别人想拿这个数据就会阻塞直到它拿到锁。传统的关系型数据库里边就用到了很多这种锁机制，比如行锁，表锁等，读锁，写锁等，都是在做操作之前先上锁。再比如 Java 里面的同步原语 synchronized 关键字的实现也是悲观锁。

乐观锁：顾名思义，就是很乐观，每次去拿数据的时候都认为别人不会修改，所以不会上锁，但是在更新的时候会判断一下在此期间别人有没有去更新这个数据，可以使用版本号等机制。乐观锁适用于多读的应用类型，这样可以提高吞吐量，像数据库提供的类似于 write\_condition 机制，其实都是提供的乐观锁。在 Java 中 java.util.concurrent.atomic 包下面的原子变量类就是使用了乐观锁的一种实现方式 CAS 实现的。

乐观锁的实现方式：

1、使用版本标识来确定读到的数据与提交时的数据是否一致。提交后修改版本标识，不一致时可以采取丢弃和再次尝试的策略。

2、java 中的 Compare and Swap 即 CAS ，当多个线程尝试使用 CAS 同时更新同一个变量时，只有其中一个线程能更新变量的值，而其它线程都失败，失败的线程并不会被挂起，而是被告知这次竞争中失败，并可以再次尝试。 CAS 操作中包含三个操作数 —— 需要读写的内存位置（V）、进行比较的预期原值（A） 和拟写入的新值(B)。如果内存位置 V 的值与预期原值 A 相匹配，那么处理器会自动将该位置值更新为新值 B。否则处理器不做任何操作。

CAS 缺点：

1、ABA 问题：

比如说一个线程 one 从内存位置 V 中取出 A，这时候另一个线程 two 也从内存中取出 A，并且 two 进行了一些操作变成了 B，然后 two 又将 V 位置的数据变成 A，这时候线程 one 进行 CAS 操作发现内存中仍然是 A，然后 one 操作成功。尽管线程 one 的 CAS 操作成功，但可能存在潜藏的问题。从 Java1.5 开始 JDK 的 atomic 包里提供了一个类 AtomicStampedReference 来解决 ABA 问题。

2、循环时间长开销大：

对于资源竞争严重（线程冲突严重）的情况，CAS 自旋的概率会比较大，从而浪费更多的 CPU 资源，效率低于 synchronized。

3、只能保证一个共享变量的原子操作：

当对一个共享变量执行操作时，我们可以使用循环 CAS 的方式来保证原子操作， 但是对多个共享变量操作时，循环 CAS 就无法保证操作的原子性，这个时候就可以用锁。

## 36、SynchronizedMap 和 ConcurrentHashMap 有什么区别？

SynchronizedMap 一次锁住整张表来保证线程安全，所以每次只能有一个线程来访为 map。

ConcurrentHashMap 使用分段锁来保证在多线程下的性能。ConcurrentHashMap 中则是一次锁住一个桶。ConcurrentHashMap 默认将 hash 表分为 16 个桶，诸如 get,put,remove 等常用操作只锁当前需要用到的桶。这样，原来只能一个线程进入，现在却能同时有 16 个写线程执行，并发性能的提升是显而易见的。

另外 ConcurrentHashMap 使用了一种不同的迭代方式。在这种迭代方式中，当iterator 被创建后集合再发生改变就不再是抛出ConcurrentModificationException，取而代之的是在改变时 new 新的数据从而不影响原有的数据 ，iterator 完成后再将头指针替换为新的数据 ，这样iterator 线程可以使用原来老的数据，而写线程也可以并发的完成改变。

## 37、CopyOnWriteArrayList 可以用于什么应用场景？

CopyOnWriteArrayList(免锁容器)的好处之一是当多个迭代器同时遍历和修改这个列表时，不会抛出 ConcurrentModificationException。在CopyOnWriteArrayList 中，写入将导致创建整个底层数组的副本，而源数组将保留在原地，使得复制的数组在被修改时，读取操作可以安全地执行。

1、由于写操作的时候，需要拷贝数组，会消耗内存，如果原数组的内容比较多的情况下，可能导致 young gc 或者 full gc；

2、不能用于实时读的场景，像拷贝数组、新增元素都需要时间，所以调用一个 set 操作后，读取到数据可能还是旧的,虽然 CopyOnWriteArrayList 能做到最终一致性,但是还是没法满足实时性要求；

CopyOnWriteArrayList 透露的思想1、读写分离，读和写分开

2、最终一致性

3、使用另外开辟空间的思路，来解决并发冲突

## 38、什么叫线程安全？servlet 是线程安全吗?

线程安全是编程中的术语，指某个函数、函数库在多线程环境中被调用时，能够正确地处理多个线程之间的共享变量，使程序功能正确完成。

Servlet 不是线程安全的，servlet 是单实例多线程的，当多个线程同时访问同一个方法，是不能保证共享变量的线程安全性的。

Struts2 的 action 是多实例多线程的，是线程安全的，每个请求过来都会 new 一个新的 action 分配给这个请求，请求完成后销毁。

SpringMVC 的 Controller 是线程安全的吗？不是的，和 Servlet 类似的处理流程。

Struts2 好处是不用考虑线程安全问题；Servlet 和 SpringMVC 需要考虑线程安全问题，但是性能可以提升不用处理太多的 gc，可以使用 ThreadLocal 来处理多线程的问题。

## 39、volatile 有什么用？能否用一句话说明下volatile 的应用场景？

volatile 保证内存可见性和禁止指令重排。

volatile 用于多线程环境下的单次操作(单次读或者单次写)。

## 40、为什么代码会重排序？

在执行程序时，为了提供性能，处理器和编译器常常会对指令进行重排序，但是不能随意重排序，不是你想怎么排序就怎么排序，它需要满足以下两个条件： 在单线程环境下不能改变程序运行的结果；

存在数据依赖关系的不允许重排序

需要注意的是：重排序不会影响单线程环境的执行结果，但是会破坏多线程的执行语义。

## 41、在 java 中 wait 和 sleep 方法的不同？

最大的不同是在等待时 wait 会释放锁，而 sleep 一直持有锁。Wait 通常被用于线程间交互，sleep 通常被用于暂停执行。

## 42、用 Java 实现阻塞队列

## 43、一个线程运行时发生异常会怎样？

如果异常没有被捕获该线程将会停止执行。Thread.UncaughtExceptionHandler 是用于处理未捕获异常造成线程突然中断情况的一个内嵌接口。当一个未捕获异常将造成线程中断的时候 JVM 会使用 Thread.getUncaughtExceptionHandler() 来查询线程的 UncaughtExceptionHandler 并将线程和异常作为参数传递给handler 的 uncaughtException()方法进行处理。

## 44、如何在两个线程间共享数据？

在两个线程间共享变量即可实现共享。

一般来说，共享变量要求变量本身是线程安全的，然后在线程内使用的时候，如果有对共享变量的复合操作，那么也得保证复合操作的线程安全性。

## 45、Java 中 notify 和 notifyAll 有什么区别？

notify() 方法不能唤醒某个具体的线程，所以只有一个线程在等待的时候它才有用武之地。而 notifyAll()唤醒所有线程并允许他们争夺锁确保了至少有一个线程能继续运行。

## 46、为什么 wait, notify 和 notifyAll 这些方法不在 thread 类里面？

一个很明显的原因是 JAVA 提供的锁是对象级的而不是线程级的，每个对象都有锁，通过线程获得。由于 wait，notify 和 notifyAll 都是锁级别的操作，所以把他们定义在 Object 类中因为锁属于对象。

## 47、什么是 ThreadLocal 变量？

ThreadLocal 是 Java 里一种特殊的变量。每个线程都有一个 ThreadLocal 就是每个线程都拥有了自己独立的一个变量，竞争条件被彻底消除了。它是为创建代价高昂的对象获取线程安全的好方法，比如你可以用 ThreadLocal 让SimpleDateFormat 变成线程安全的，因为那个类创建代价高昂且每次调用都需

要创建不同的实例所以不值得在局部范围使用它，如果为每个线程提供一个自己独有的变量拷贝，将大大提高效率。首先，通过复用减少了代价高昂的对象的创建个数。其次，你在没有使用高代价的同步或者不变性的情况下获得了线程安全。

## 48、Java 中 interrupted 和 isInterrupted 方法的区别？

interrupt

interrupt 方法用于中断线程。调用该方法的线程的状态为将被置为”中断”状态。

注意：线程中断仅仅是置线程的中断状态位，不会停止线程。需要用户自己去监视线程的状态为并做处理。支持线程中断的方法（也就是线程中断后会抛出interruptedException 的方法）就是在监视线程的中断状态，一旦线程的中断状态被置为“中断状态”，就会抛出中断异常。

interrupted

查询当前线程的中断状态，并且清除原状态。如果一个线程被中断了，第一次调用 interrupted 则返回 true，第二次和后面的就返回 false 了。isInterrupted

仅仅是查询当前线程的中断状态

## 49、为什么 wait 和 notify 方法要在同步块中调用？

Java API 强制要求这样做，如果你不这么做，你的代码会抛出IllegalMonitorStateException 异常。还有一个原因是为了避免 wait 和notify 之间产生竞态条件。

## 50、为什么你应该在循环中检查等待条件?

处于等待状态的线程可能会收到错误警报和伪唤醒，如果不在循环中检查等待条件，程序就会在没有满足结束条件的情况下退出。

## 51、Java 中的同步集合与并发集合有什么区别？

同步集合与并发集合都为多线程和并发提供了合适的线程安全的集合，不过并发集合的可扩展性更高。在 Java1.5 之前程序员们只有同步集合来用且在多线程并发的时候会导致争用，阻碍了系统的扩展性。Java5 介绍了并发集合像ConcurrentHashMap，不仅提供线程安全还用锁分离和内部分区等现代技术提高了可扩展性。

## 52、什么是线程池？ 为什么要使用它？

创建线程要花费昂贵的资源和时间，如果任务来了才创建线程那么响应时间会变长，而且一个进程能创建的线程数有限。为了避免这些问题，在程序启动的时候就创建若干线程来响应处理，它们被称为线程池，里面的线程叫工作线程。从JDK1.5 开始，Java API 提供了 Executor 框架让你可以创建不同的线程池。

## 53、怎么检测一个线程是否拥有锁？

在 java.lang.Thread 中有一个方法叫 holdsLock()，它返回 true 如果当且仅当当前线程拥有某个具体对象的锁。

## 54、你如何在 Java 中获取线程堆栈？

kill -3 [java pid]

不会在当前终端输出，它会输出到代码执行的或指定的地方去。比如，kill -3 tomcat pid, 输出堆栈到 log 目录下。

Jstack [java pid]

这个比较简单，在当前终端显示，也可以重定向到指定文件中。

-JvisualVM：Thread Dump

不做说明，打开 JvisualVM 后，都是界面操作，过程还是很简单的。

## 55、JVM 中哪个参数是用来控制线程的栈堆栈小的?

-Xss 每个线程的栈大小

## 56、Thread 类中的 yield 方法有什么作用？

使当前线程从执行状态（运行状态）变为可执行态（就绪状态）。

当前线程到了就绪状态，那么接下来哪个线程会从就绪状态变成执行状态呢？可能是当前线程，也可能是其他线程，看系统的分配了。

## 57、Java 中 ConcurrentHashMap 的并发度是什么？

ConcurrentHashMap 把实际 map 划分成若干部分来实现它的可扩展性和线程安全。这种划分是使用并发度获得的，它是 ConcurrentHashMap 类构造函数的一个可选参数，默认值为 16，这样在多线程情况下就能避免争用。

在 JDK8 后，它摒弃了 Segment（锁段）的概念，而是启用了一种全新的方式实现,利用 CAS 算法。同时加入了更多的辅助变量来提高并发度，具体内容还是查看源码吧。

## 58、Java 中 Semaphore 是什么？

Java 中的 Semaphore 是一种新的同步类，它是一个计数信号。从概念上讲，从概念上讲，信号量维护了一个许可集合。如有必要，在许可可用前会阻塞每一个acquire()，然后再获取该许可。每个 release()添加一个许可，从而可能释放一个正在阻塞的获取者。但是，不使用实际的许可对象，Semaphore 只对可用许可的号码进行计数，并采取相应的行动。信号量常常用于多线程的代码中，比如数据库连接池。

## 59、Java 线程池中 submit() 和 execute()方法有什么区别？

两个方法都可以向线程池提交任务，execute()方法的返回类型是 void，它定义在 Executor 接口中。

而 submit()方法可以返回持有计算结果的 Future 对象，它定义在ExecutorService 接口中，它扩展了 Executor 接口，其它线程池类像ThreadPoolExecutor 和 ScheduledThreadPoolExecutor 都有这些方法。

## 60、什么是阻塞式方法？

阻塞式方法是指程序会一直等待该方法完成期间不做其他事情，ServerSocket 的 accept()方法就是一直等待客户端连接。这里的阻塞是指调用结果返回之前， 当前线程会被挂起，直到得到结果之后才会返回。此外，还有异步和非阻塞式方法在任务完成前就返回。

## 61、Java 中的 ReadWriteLock 是什么？

读写锁是用来提升并发程序性能的锁分离技术的成果。

## 62、volatile 变量和 atomic 变量有什么不同？

Volatile 变量可以确保先行关系，即写操作会发生在后续的读操作之前, 但它并不能保证原子性。例如用 volatile 修饰 count 变量那么 count++ 操作就不是原子性的。

而 AtomicInteger 类提供的 atomic 方法可以让这种操作具有原子性如getAndIncrement()方法会原子性的进行增量操作把当前值加一，其它数据类型和引用变量也可以进行相似操作。

## 63、可以直接调用 Thread 类的 run ()方法么？

当然可以。但是如果我们调用了 Thread 的 run()方法，它的行为就会和普通的方法一样，会在当前线程中执行。为了在新的线程中执行我们的代码，必须使用Thread.start()方法。

## 64、如何让正在运行的线程暂停一段时间？

我们可以使用 Thread 类的 Sleep()方法让线程暂停一段时间。需要注意的是， 这并不会让线程终止，一旦从休眠中唤醒线程，线程的状态将会被改变为Runnable，并且根据线程调度，它将得到执行。

## 65、你对线程优先级的理解是什么？

每一个线程都是有优先级的，一般来说，高优先级的线程在运行时会具有优先权， 但这依赖于线程调度的实现，这个实现是和操作系统相关的(OS dependent)。我们可以定义线程的优先级，但是这并不能保证高优先级的线程会在低优先级的线程前执行。线程优先级是一个 int 变量(从 1-10)，1 代表最低优先级，10 代表最高优先级。

java 的线程优先级调度会委托给操作系统去处理，所以与具体的操作系统优先级有关，如非特别需要，一般无需设置线程优先级。

## 66、什么是线程调度器(Thread Scheduler)和时间分片(Time Slicing )？

线程调度器是一个操作系统服务，它负责为 Runnable 状态的线程分配 CPU 时间。一旦我们创建一个线程并启动它，它的执行便依赖于线程调度器的实现。同上一个问题，线程调度并不受到 Java 虚拟机控制，所以由应用程序来控制它是更好的选择（也就是说不要让你的程序依赖于线程的优先级）。

时间分片是指将可用的 CPU 时间分配给可用的 Runnable 线程的过程。分配CPU 时间可以基于线程优先级或者线程等待的时间。

## 67、你如何确保 main()方法所在的线程是 Java 程序最后结束的线程？

我们可以使用 Thread 类的 join()方法来确保所有程序创建的线程在 main() 方法退出前结束。

## 68、线程之间是如何通信的？

当线程间是可以共享资源时，线程间通信是协调它们的重要的手段。Object 类中 wait()\notify()\notifyAll()方法可以用于线程间通信关于资源的锁的状态。

## 69、为什么线程通信的方法 wait(), notify()和notifyAll()被定义在 Object 类里？

Java 的每个对象中都有一个锁(monitor，也可以成为监视器) 并且 wait()， notify()等方法用于等待对象的锁或者通知其他线程对象的监视器可用。在Java 的线程中并没有可供任何对象使用的锁和同步器。这就是为什么这些方法是 Object 类的一部分，这样 Java 的每一个类都有用于线程间通信的基本方法。

## 70、为什么 wait(), notify()和 notifyAll ()必须在同步方法或者同步块中被调用？

当一个线程需要调用对象的 wait()方法的时候，这个线程必须拥有该对象的锁， 接着它就会释放这个对象锁并进入等待状态直到其他线程调用这个对象上的notify()方法。同样的，当一个线程需要调用对象的 notify()方法时，它会释放这个对象的锁，以便其他在等待的线程就可以得到这个对象锁。由于所有的这些方法都需要线程持有对象的锁，这样就只能通过同步来实现，所以他们只能在同步方法或者同步块中被调用。

## 71、为什么 Thread 类的 sleep()和 yield ()方法是静态的？

Thread 类的 sleep()和 yield()方法将在当前正在执行的线程上运行。所以在其他处于等待状态的线程上调用这些方法是没有意义的。这就是为什么这些方法

是静态的。它们可以在当前正在执行的线程中工作，并避免程序员错误的认为可以在其他非运行线程调用这些方法。

## 72、如何确保线程安全？

在 Java 中可以有很多方法来保证线程安全——同步，使用原子类(atomic concurrent classes)，实现并发锁，使用 volatile 关键字，使用不变类和线程安全类。

## 73、同步方法和同步块，哪个是更好的选择？

同步块是更好的选择，因为它不会锁住整个对象（当然你也可以让它锁住整个对象）。同步方法会锁住整个对象，哪怕这个类中有多个不相关联的同步块，这通常会导致他们停止执行并需要等待获得这个对象上的锁。

同步块更要符合开放调用的原则，只在需要锁住的代码块锁住相应的对象，这样从侧面来说也可以避免死锁。

## 74、如何创建守护线程？

使用 Thread 类的 setDaemon(true)方法可以将线程设置为守护线程，需要注意的是，需要在调用 start()方法前调用这个方法，否则会抛出IllegalThreadStateException 异常。

## 75、什么是 Java Timer 类？如何创建一个有特定时间间隔的任务？

java.util.Timer 是一个工具类，可以用于安排一个线程在未来的某个特定时间执行。Timer 类可以用安排一次性任务或者周期任务。

java.util.TimerTask 是一个实现了 Runnable 接口的抽象类，我们需要去继承这个类来创建我们自己的定时任务并使用 Timer 去安排它的执行。

# 四、Redis 面试题

## 1、什么是 Redis?

Redis 是完全开源免费的，遵守 BSD 协议，是一个高性能的 key-value 数据库。

### Redis 与其他 key - value 缓存产品有以下三个特点：

1. Redis 支持数据的持久化，可以将内存中的数据保存在磁盘中，重启的时候可以再次加载进行使用。
2. Redis 不仅仅支持简单的 key-value 类型的数据，同时还提供list，set， zset，hash 等数据结构的存储。
3. Redis 支持数据的备份，即 master-slave 模式的数据备份。

### Redis 优势

1. 性能极高 – Redis 能读的速度是 110000 次/s,写的速度是 81000 次

/s 。

1. 丰富的数据类型 – Redis 支持二进制案例的 Strings, Lists, Hashes, Sets 及 Ordered Sets 数据类型操作。
2. 原子 – Redis 的所有操作都是原子性的，意思就是要么成功执行要么失败完全不执行。单个操作是原子性的。多个操作也支持事务，即原子性，通过MULTI 和 EXEC 指令包起来。
3. 丰富的特性 – Redis 还支持 publish/subscribe, 通知, key 过期等等特性。

### Redis 与其他 key-value 存储有什么不同？

1. Redis 有着更为复杂的数据结构并且提供对他们的原子性操作，这是一个不同于其他数据库的进化路径。Redis 的数据类型都是基于基本数据结构的同时对程序员透明，无需进行额外的抽象。
2. Redis 运行在内存中但是可以持久化到磁盘，所以在对不同数据集进行高速读写时需要权衡内存，因为数据量不能大于硬件内存。在内存数据库方面的另一个优点是，相比在磁盘上相同的复杂的数据结构，在内存中操作起来非常简单， 这样 Redis 可以做很多内部复杂性很强的事情。同时，在磁盘格式方面他们是紧凑的以追加的方式产生的，因为他们并不需要进行随机访问。

## 2、Redis 的数据类型？

答：Redis 支持五种数据类型：string（字符串），hash（哈希），list（列表）， set（集合）及 zsetsorted set：有序集合)。

我们实际项目中比较常用的是 string，hash 如果你是 Redis 中高级用户，还需要加上下面几种数据结构 HyperLogLog、Geo、Pub/Sub。

如果你说还玩过 Redis Module，像 BloomFilter，RedisSearch，Redis-ML，面试官得眼睛就开始发亮了。

## 3、使用 Redis 有哪些好处？

1. 速度快，因为数据存在内存中，类似于 HashMap，HashMap 的优势就是查找和操作的时间复杂度都是 O1)
2. 支持丰富数据类型，支持 string，list，set，Zset，hash 等
3. 支持事务，操作都是原子性，所谓的原子性就是对数据的更改要么全部执行，要么全部不执行
4. 丰富的特性：可用于缓存，消息，按 key 设置过期时间，过期后将会自动删除

## 4、Redis 相比 Memcached 有哪些优势？

1. Memcached 所有的值均是简单的字符串，redis 作为其替代者，支持更为丰富的数据类
2. Redis 的速度比 Memcached 快很
3. Redis 可以持久化其数据

## 5、Memcache 与 Redis 的区别都有哪些？

1. 存储方式 Memecache 把数据全部存在内存之中，断电后会挂掉，数据不能超过内存大小。 Redis 有部份存在硬盘上，这样能保证数据的持久性。
2. 数据支持类型 Memcache 对数据类型支持相对简单。 Redis 有复杂的数据类型。
3. 使用底层模型不同 它们之间底层实现方式 以及与客户端之间通信的应用协议不一样。 Redis 直接自己构建了 VM 机制 ，因为一般的系统调用系统函数的话，会浪费一定的时间去移动和请求。

## 6、Redis 是单进程单线程的？

答：Redis 是单进程单线程的，redis 利用队列技术将并发访问变为串行访问， 消除了传统数据库串行控制的开销。

## 7、一个字符串类型的值能存储最大容量是多少？

答：512M

### 8、Redis 的持久化机制是什么？各自的优缺点？

Redis 提供两种持久化机制 RDB 和 AOF 机制:

### 1、RDBRedis DataBase)持久化方式：

是指用数据集快照的方式半持久化模式)记录 redis 数据库的所有键值对,在某个时间点将数据写入一个临时文件，持久化结束后，用这个临时文件替换上次持久化的文件，达到数据恢复。

优点：

1. 只有一个文件 dump.rdb，方便持久化。
2. 容灾性好，一个文件可以保存到安全的磁盘。
3. 性能最大化，fork 子进程来完成写操作，让主进程继续处理命令，所以是 IO 最大化。使用单独子进程来进行持久化，主进程不会进行任何 IO 操作，保证了 redis 的高性能)
4. 相对于数据集大时，比 AOF 的启动效率更高。缺点：

数据安全性低。RDB 是间隔一段时间进行持久化，如果持久化之间 redis 发生故障，会发生数据丢失。所以这种方式更适合数据要求不严谨的时候

### 2、AOFAppend-only file)持久化方式：

是指所有的命令行记录以 redis 命令请求协议的格式完全持久化存储)保存为aof 文件。

优点：

1. 数据安全，aof 持久化可以配置 appendfsync 属性，有 always，每进行一次命令操作就记录到 aof 文件中一次。
2. 通过 append 模式写文件，即使中途服务器宕机，可以通过redis-check-aof 工具解决数据一致性问题。
3. AOF 机制的 rewrite 模式。AOF 文件没被 rewrite 之前（文件过大时会对命令进行合并重写），可以删除其中的某些命令（比如误操作的 flushall）) 缺点：
4. AOF 文件比 RDB 文件大，且恢复速度慢。
5. 数据集大的时候，比 rdb 启动效率低。

## 9、Redis 常见性能问题和解决方案：

1. Master 最好不要写内存快照，如果 Master 写内存快照，save 命令调度rdbSave 函数，会阻塞主线程的工作，当快照比较大时对性能影响是非常大的， 会间断性暂停服务
2. 如果数据比较重要，某个 Slave 开启 AOF 备份数据，策略设置为每秒同步一
3. 为了主从复制的速度和连接的稳定性，Master 和 Slave 最好在同一个局域网
4. 尽量避免在压力很大的主库上增加从
5. 主从复制不要用图状结构，用单向链表结构更为稳定，即：Master <- Slave1<- Slave2 <- Slave3…这样的结构方便解决单点故障问题，实现 Slave 对 Master 的替换。如果 Master 挂了，可以立刻启用 Slave1 做 Master，其他不变。

## 10、redis 过期键的删除策略？

1. 定时删除:在设置键的过期时间的同时，创建一个定时器 timer). 让定时器在键的过期时间来临时，立即执行对键的删除操作。
2. 惰性删除:放任键过期不管，但是每次从键空间中获取键时，都检查取得的键是否过期，如果过期的话，就删除该键;如果没有过期，就返回该键。
3. 定期删除:每隔一段时间程序就对数据库进行一次检查，删除里面的过期键。至于要删除多少过期键，以及要检查多少个数据库，则由算法决定。

## 11、Redis 的回收策略（淘汰策略）?

volatile-lru：从已设置过期时间的数据集（server.db[i].expires）中挑选最近最少使用的数据淘汰

volatile-ttl：从已设置过期时间的数据集（server.db[i].expires）中挑选将要过期的数据淘汰

volatile-random：从已设置过期时间的数据集（server.db[i].expires）中任意选择数据淘汰

allkeys-lru：从数据集（server.db[i].dict）中挑选最近最少使用的数据淘汰allkeys-random：从数据集（server.db[i].dict）中任意选择数据淘汰

no-enviction（驱逐）：禁止驱逐数据

注意这里的 6 种机制，volatile 和 allkeys 规定了是对已设置过期时间的数据集淘汰数据还是从全部数据集淘汰数据，后面的 lru、ttl 以及 random 是三种不同的淘汰策略，再加上一种 no-enviction 永不回收的策略。

使用策略规则：

1. 如果数据呈现幂律分布，也就是一部分数据访问频率高，一部分数据访问频率低，则使用 allkeys-lru
2. 如果数据呈现平等分布，也就是所有的数据访问频率都相同，则使用allkeys-random

## 12、为什么 edis 需要把所有数据放到内存中？

答 ：Redis 为了达到最快的读写速度将数据都读到内存中，并通过异步的方式将数据写入磁盘。所以 redis 具有快速和数据持久化的特征。如果不将数据放在内存中，磁盘 I/O 速度为严重影响 redis 的性能。在内存越来越便宜的今天， redis 将会越来越受欢迎。如果设置了最大使用的内存，则数据已有记录数达到内存限值后不能继续插入新值。

## 13、Redis 的同步机制了解么？

答：Redis 可以使用主从同步，从从同步。第一次同步时，主节点做一次 bgsave， 并同时将后续修改操作记录到内存 buffer，待完成后将 rdb 文件全量同步到复制节点，复制节点接受完成后将 rdb 镜像加载到内存。加载完成后，再通知主节点将期间修改的操作记录同步到复制节点进行重放就完成了同步过程。

## 14、Pipeline 有什么好处，为什么要用 pipeline？

答：可以将多次 IO 往返的时间缩减为一次，前提是 pipeline 执行的指令之间没有因果相关性。使用 redis-benchmark 进行压测的时候可以发现影响 redis 的 QPS 峰值的一个重要因素是 pipeline 批次指令的数目。

## 15、是否使用过 Redis 集群，集群的原理是什么？

1. Redis Sentinal 着眼于高可用，在 master 宕机时会自动将 slave 提升为 master，继续提供服务。
2. Redis Cluster 着眼于扩展性，在单个 redis 内存不足时，使用 Cluster 进行分片存储。

## 16、Redis 集群方案什么情况下会导致整个集群不可用？

答：有 A，B，C 三个节点的集群,在没有复制模型的情况下,如果节点 B 失败了， 那么整个集群就会以为缺少 5501-11000 这个范围的槽而不可用。

## 17、Redis 支持的 Java 客户端都有哪些？官方推荐用哪个？

答：Redisson、Jedis、lettuce 等等，官方推荐使用 Redisson。

## 18、Jedis 与 Redisson 对比有什么优缺点？

答：Jedis 是 Redis 的 Java 实现的客户端，其 API 提供了比较全面的 Redis 命令的支持；Redisson 实现了分布式和可扩展的 Java 数据结构，和 Jedis 相比，功能较为简单，不支持字符串操作，不支持排序、事务、管道、分区等 Redis 特性。

Redisson 的宗旨是促进使用者对 Redis 的关注分离，从而让使用者能够将精力更集中地放在处理业务逻辑上。

## 19、Redis 如何设置密码及验证密码？

设置密码：config set requirepass 123456 授权密码：auth 123456

## 20、说说 Redis 哈希槽的概念？

答：Redis 集群没有使用一致性 hash,而是引入了哈希槽的概念，Redis 集群有16384 个哈希槽，每个 key 通过 CRC16 校验后对 16384 取模来决定放置哪个槽，集群的每个节点负责一部分 hash 槽。

## 21、Redis 集群的主从复制模型是怎样的？

答：为了使在部分节点失败或者大部分节点无法通信的情况下集群仍然可用，所以集群使用了主从复制模型,每个节点都会有 N-1 个复制品.

## 22、Redis 集群会有写操作丢失吗？为什么？

答 ：Redis 并不能保证数据的强一致性，这意味这在实际中集群在特定的条件下可能会丢失写操作。

## 23、Redis 集群之间是如何复制的？

答：异步复制

## 24、Redis 集群最大节点个数是多少？

答：16384 个。

## 25、Redis 集群如何选择数据库？

答：Redis 集群目前无法做数据库选择，默认在 0 数据库。

## 26、怎么测试 Redis 的连通性？

答：使用 ping 命令。

## 27、怎么理解 Redis 事务？

答：

1. 事务是一个单独的隔离操作：事务中的所有命令都会序列化、按顺序地执行。事务在执行的过程中，不会被其他客户端发送来的命令请求所打断。
2. 事务是一个原子操作：事务中的命令要么全部被执行，要么全部都不执行。

## 28、Redis 事务相关的命令有哪几个？

答：MULTI、EXEC、DISCARD、WATCH

## 29、Redis key 的过期时间和永久有效分别怎么设置？

答：EXPIRE 和 PERSIST 命令。

## 30、Redis 如何做内存优化？

答：尽可能使用散列表（hashes），散列表（是说散列表里面存储的数少）使用的内存非常小，所以你应该尽可能的将你的数据模型抽象到一个散列表里面。比如你的 web 系统中有一个用户对象，不要为这个用户的名称，姓氏，邮箱，密码设置单独的 key,而是应该把这个用户的所有信息存储到一张散列表里面。

## 31、Redis 回收进程如何工作的？

答：一个客户端运行了新的命令，添加了新的数据。Redi 检查内存使用情况， 如果大于 maxmemory 的限制, 则根据设定好的策略进行回收。一个新的命令被执行，等等。所以我们不断地穿越内存限制的边界，通过不断达到边界然后不断

地回收回到边界以下。如果一个命令的结果导致大量内存被使用（例如很大的集合的交集保存到一个新的键），不用多久内存限制就会被这个内存使用量超越。

## 32、都有哪些办法可以降低 Redis 的内存使用情况呢？

答：如果你使用的是 32 位的 Redis 实例，可以好好利用 Hash,list,sorted set,set 等集合类型数据，因为通常情况下很多小的 Key-Value 可以用更紧凑的方式存放到一起。

## 33、Redis 的内存用完了会发生什么？

答：如果达到设置的上限，Redis 的写命令会返回错误信息（但是读命令还可以正常返回。）或者你可以将 Redis 当缓存来使用配置淘汰机制，当 Redis 达到内存上限时会冲刷掉旧的内容。

## 34、一个 Redis 实例最多能存放多少的 keys？ List、Set、Sorted Set 他们最多能存放多少元素？

答：理论上 Redis 可以处理多达 232 的 keys，并且在实际中进行了测试，每个实例至少存放了 2 亿 5 千万的 keys。我们正在测试一些较大的值。任何list、set、和 sorted set 都可以放 232 个元素。换句话说，Redis 的存储极限是系统中的可用内存值。

## 35、MySQL 里有 2000w 数据，redis 中只存 20w 的数据，如何保证 redis 中的数据都是热点数据？

答：Redis 内存数据集大小上升到一定大小的时候，就会施行数据淘汰策略。相关知识：Redis 提供 6 种数据淘汰策略：

volatile-lru：从已设置过期时间的数据集（server.db[i].expires）中挑选最近最少使用的数据淘汰

volatile-ttl：从已设置过期时间的数据集（server.db[i].expires）中挑选将要过期的数据淘汰

volatile-random：从已设置过期时间的数据集（server.db[i].expires）中任意选择数据淘汰

allkeys-lru：从数据集（server.db[i].dict）中挑选最近最少使用的数据淘汰allkeys-random：从数据集（server.db[i].dict）中任意选择数据淘汰

no-enviction（驱逐）：禁止驱逐数据

## 36、Redis 最适合的场景？

### 1、会话缓存（Session Cache）

最常用的一种使用 Redis 的情景是会话缓存（session cache）。用 Redis 缓存会话比其他存储（如 Memcached）的优势在于：Redis 提供持久化。当维护一个不是严格要求一致性的缓存时，如果用户的购物车信息全部丢失，大部分人都会不高兴的，现在，他们还会这样吗？ 幸运的是，随着 Redis 这些年的改进， 很容易找到怎么恰当的使用 Redis 来缓存会话的文档。甚至广为人知的商业平台 Magento 也提供 Redis 的插件。

### 2、全页缓存（FPC）

除基本的会话 token 之外，Redis 还提供很简便的 FPC 平台。回到一致性问题，即使重启了 Redis 实例，因为有磁盘的持久化，用户也不会看到页面加载速度的下降，这是一个极大改进，类似 PHP 本地 FPC。 再次以 Magento 为例， Magento 提供一个插件来使用 Redis 作为全页缓存后端。 此外，对 WordPress 的用户来说，Pantheon 有一个非常好的插件 wp-redis，这个插件能帮助你以最快速度加载你曾浏览过的页面。

### 3、队列

Reids 在内存存储引擎领域的一大优点是提供 list 和 set 操作，这使得 Redis 能作为一个很好的消息队列平台来使用。Redis 作为队列使用的操作，就类似于本地程序语言（如 Python）对 list 的 push/pop 操作。 如果你快速的在 Google 中搜索“Redis queues”，你马上就能找到大量的开源项目，这些项目的目的就是利用 Redis 创建非常好的后端工具，以满足各种队列需求。例如， Celery 有一个后台就是使用 Redis 作为 broker，你可以从这里去查看。

### 4，排行榜/计数器

Redis 在内存中对数字进行递增或递减的操作实现的非常好。集合（Set）和有序集合（Sorted Set）也使得我们在执行这些操作的时候变的非常简单，Redis 只是正好提供了这两种数据结构。所以，我们要从排序集合中获取到排名最靠前的10 个用户–我们称之为“user\_scores”，我们只需要像下面一样执行即可： 当然，这是假定你是根据你用户的分数做递增的排序。如果你想返回用户及用户的分数，你需要这样执行： ZRANGE user\_scores 0 10 WITHSCORES Agora Games 就是一个很好的例子，用 Ruby 实现的，它的排行榜就是使用 Redis 来存储数据的，你可以在这里看到。

### 5、发布/订阅

最后（但肯定不是最不重要的）是 Redis 的发布/订阅功能。发布/订阅的使用场景确实非常多。我已看见人们在社交网络连接中使用，还可作为基于发布/订阅的脚本触发器，甚至用 Redis 的发布/订阅功能来建立聊天系统！

## 37、假如 Redis 里面有 1 亿个 key，其中有 10w 个key 是以某个固定的已知的前缀开头的，如果将它们全部找出来？

答：使用 keys 指令可以扫出指定模式的 key 列表。

对方接着追问：如果这个 redis 正在给线上的业务提供服务，那使用 keys 指令会有什么问题？

这个时候你要回答 redis 关键的一个特性：redis 的单线程的。keys 指令会导致线程阻塞一段时间，线上服务会停顿，直到指令执行完毕，服务才能恢复。这个时候可以使用 scan 指令，scan 指令可以无阻塞的提取出指定模式的 key 列表，但是会有一定的重复概率，在客户端做一次去重就可以了，但是整体所花费的时间会比直接用 keys 指令长。

## 38、如果有大量的 key 需要设置同一时间过期，一般需要注意什么？

答：如果大量的 key 过期时间设置的过于集中，到过期的那个时间点，redis 可能会出现短暂的卡顿现象。一般需要在时间上加一个随机值，使得过期时间分散一些。

## 39、使用过 Redis 做异步队列么，你是怎么用的？

答：一般使用 list 结构作为队列，rpush 生产消息，lpop 消费消息。当 lpop 没有消息的时候，要适当 sleep 一会再重试。如果对方追问可不可以不用 sleep 呢？list 还有个指令叫 blpop，在没有消息的时候，它会阻塞住直到消息到来。如果对方追问能不能生产一次消费多次呢？使用 pub/sub 主题订阅者模式，可以实现 1:N 的消息队列。

如果对方追问 pub/sub 有什么缺点？

在消费者下线的情况下，生产的消息会丢失，得使用专业的消息队列如 RabbitMQ 等。

如果对方追问 redis 如何实现延时队列？

我估计现在你很想把面试官一棒打死如果你手上有一根棒球棍的话，怎么问的这么详细。但是你很克制，然后神态自若的回答道：使用 sortedset，拿时间戳作为 score，消息内容作为 key 调用 zadd 来生产消息，消费者用 zrangebyscore 指令获取 N 秒之前的数据轮询进行处理。到这里，面试官暗地里已经对你竖起了大拇指。但是他不知道的是此刻你却竖起了中指，在椅子背后。

## 40、使用过 Redis 分布式锁么，它是什么回事？

先拿 setnx 来争抢锁，抢到之后，再用 expire 给锁加一个过期时间防止锁忘记了释放。

这时候对方会告诉你说你回答得不错，然后接着问如果在 setnx 之后执行expire 之前进程意外 crash 或者要重启维护了，那会怎么样？这时候你要给予惊讶的反馈：唉，是喔，这个锁就永远得不到释放了。紧接着你需要抓一抓自己得脑袋，故作思考片刻，好像接下来的结果是你主动思考出来的，然后回答：我记得 set 指令有非常复杂的参数，这个应该是可以同时把 setnx 和 expire 合成一条指令来用的！对方这时会显露笑容，心里开始默念：摁，这小子还不错。

# 五、MyBatis 面试题答案解析

### 1、什么是 Mybatis？

1. Mybatis 是一个半 ORM（对象关系映射）框架，它内部封装了 JDBC，开发时只需要关注 SQL 语句本身，不需要花费精力去处理加载驱动、创建连接、创建 statement 等繁杂的过程。程序员直接编写原生态 sql，可以严格控制 sql 执行性能，灵活度高。
2. MyBatis 可以使用 XML 或注解来配置和映射原生信息，将 POJO 映射成数据库中的记录，避免了几乎所有的 JDBC 代码和手动设置参数以及获取结果集。
3. 通过 xml 文件或注解的方式将要执行的各种 statement 配置起来，并通过 java 对象和 statement 中 sql 的动态参数进行映射生成最终执行的 sql 语句，最后由 mybatis 框架执行 sql 并将结果映射为 java 对象并返回。（从执行 sql 到返回 result 的过程）。

### 2、Mybaits 的优点：

1. 基 于 SQL 语句编程，相当灵活，不会对应用程序或者数据库的现有设计造成任何影响，SQL 写在 XML 里，解除 sql 与程序代码的耦合，便于统一管理； 提供 XML 标签，支持编写动态 SQL 语句，并可重用。
2. 与 JDBC 相比，减少了 50%以上的代码量，消除了 JDBC 大量冗余的代码，不需要手动开关连接；
3. 很好的与各种数据库兼容（因为 MyBatis 使用 JDBC 来连接数据库，所以只要 JDBC 支持的数据库 MyBatis 都支持）。
4. 能够与 Spring 很好的集成；
5. 提供映射标签，支持对象与数据库的 ORM 字段关系映射；提供对象关系映射标签，支持对象关系组件维护。

### 3、MyBatis 框架的缺点：

1. SQL 语句的编写工作量较大，尤其当字段多、关联表多时，对开发人员编写 SQL 语句的功底有一定要求。
2. SQL 语句依赖于数据库，导致数据库移植性差，不能随意更换数据库。

### 4、MyBatis 框架适用场合：

1. MyBatis 专注于 SQL 本身，是一个足够灵活的 DAO 层解决方案。
2. 对性能的要求很高，或者需求变化较多的项目，如互联网项目，MyBatis 将是不错的选择。

### 5、MyBatis 与 Hibernate 有哪些不同？

1. Mybatis 和 hibernate 不同，它不完全是一个 ORM 框架，因为 MyBatis 需要程序员自己编写 Sql 语句。
2. Mybatis 直接编写原生态 sql，可以严格控制 sql 执行性能，灵活度高， 非常适合对关系数据模型要求不高的软件开发，因为这类软件需求变化频繁，一但需求变化要求迅速输出成果。但是灵活的前提是 mybatis 无法做到数据库无关性，如果需要实现支持多种数据库的软件，则需要自定义多套 sql 映射文件， 工作量大。
3. Hibernate 对象/关系映射能力强，数据库无关性好，对于关系模型要求高的软件，如果用 hibernate 开发可以节省很多代码，提高效率。

### 6、#{}和${}的区别是什么？

#{}是预编译处理，${}是字符串替换。

Mybatis 在处理#{}时，会将 sql 中的#{}替换为?号，调用 PreparedStatement 的 set 方法来赋值；

Mybatis 在处理${}时，就是把${}替换成变量的值。使用#{}可以有效的防止 SQL 注入，提高系统安全性。

### 7、当实体类中的属性名和表中的字段名不一样 ，怎么办 ？

第 1 种： 通过在查询的 sql 语句中定义字段名的别名，让字段名的别名和实体类的属性名一致。

<select id=”selectorder” parametertype=”int” resultetype=”

me.gacl.domain.order”>

select order\_id id, order\_no orderno ,order\_price price form orders where order\_id=#{id};

</select>复制代码

第 2 种： 通过<resultMap>来映射字段名和实体类属性名的一一对应的关系。

<select id="getOrder" parameterType="int" resultMap="orderresultmap">

select \* from orders where order\_id=#{id}

</select>

<resultMap type=”me.gacl.domain.order” id=”orderresultmap”>

<!–用 id 属性来映射主键字段–>

<id property=”id” column=”order\_id”>

<!–用 result 属性来映射非主键字段，property 为实体类属性名，column 为数据表中的属性–>

<result property = “orderno” column =”order\_no”/>

<result property=”price” column=”order\_price” />

</reslutMap>复制代码

### 8、 模糊查询 like 语句该怎么写?

第 1 种：在 Java 代码中添加 sql 通配符。

string wildcardname = “%smi%”;

list<name> names = mapper.selectlike(wildcardname);

<select id=”selectlike”>

select \* from foo where bar like #{value}

</select>复制代码

第 2 种：在 sql 语句中拼接通配符，会引起 sql 注入

string wildcardname = “smi”;

list<name> names = mapper.selectlike(wildcardname);

<select id=”selectlike”>

select \* from foo where bar like "%"#{value}"%"

</select>复制代码

### 9、通常一个 Xml 映射文件，都会写一个 Dao 接口与之对应，请问， 这个 Dao 接口的工作原理是什么？Dao 接口里的方法，参数不同时， 方法能重载吗？

Dao 接口即 Mapper 接口。接口的全限名，就是映射文件中的 namespace 的值； 接口的方法名，就是映射文件中 Mapper 的 Statement 的 id 值；接口方法内的参数，就是传递给 sql 的参数。

Mapper 接口是没有实现类的，当调用接口方法时，接口全限名+方法名拼接字符串作为 key 值，可唯一定位一个 MapperStatement。在 Mybatis 中，每一个

<select>、<insert>、<update>、<delete>标签，都会被解析为一个MapperStatement 对象。

举例：com.mybatis3.mappers.StudentDao.findStudentById，可以唯一找到namespace 为 com.mybatis3.mappers.StudentDao 下面 id 为findStudentById 的 MapperStatement。

Mapper 接口里的方法，是不能重载的，因为是使用 全限名+方法名 的保存和寻找策略。Mapper 接口的工作原理是 JDK 动态代理，Mybatis 运行时会使用 JDK 动态代理为 Mapper 接口生成代理对象 proxy，代理对象会拦截接口方法，转而执行 MapperStatement 所代表的 sql，然后将 sql 执行结果返回。

### 10、Mybatis 是如何进行分页的？分页插件的原理是什么？

Mybatis 使用 RowBounds 对象进行分页，它是针对 ResultSet 结果集执行的内存分页，而非物理分页。可以在 sql 内直接书写带有物理分页的参数来完成物理分页功能，也可以使用分页插件来完成物理分页。

分页插件的基本原理是使用 Mybatis 提供的插件接口，实现自定义插件，在插件的拦截方法内拦截待执行的 sql，然后重写 sql，根 据 dialect 方言，添加对应的物理分页语句和物理分页参数。

### 11、Mybatis 是如何将 sql 执行结果封装为目标对象并返回的？都有哪些映射形式？

第一种是使用<resultMap>标签，逐一定义数据库列名和对象属性名之间的映射关系。

第二种是使用 sql 列的别名功能，将列的别名书写为对象属性名。

有了列名与属性名的映射关系后，Mybatis 通过反射创建对象，同时使用反射给对象的属性逐一赋值并返回，那些找不到映射关系的属性，是无法完成赋值的。

### 12、如何执行批量插入?

首先,创建一个简单的 insert 语句:

<insert id=”insertname”>

insert into names (name) values (#{ value

}

)

</insert>复制代码

然后在 java 代码中像下面这样执行批处理插入:

list < string > names = new arraylist(); names.add(“fred”); names.add(“barney”); names.add(“betty”);

names.add(“wilma”);

// 注意这里 executortype.batch sqlsession sqlsession =

sqlsessionfactory.opensession(executortype.batch);

try {

namemapper mapper = sqlsession.getmapper(namemapper.class);

for (string name: names) { mapper.insertname(name);

}

sqlsession.commit();

}

catch (Exception e) {

e.printStackTrace(); sqlSession.rollback(); throw e;

}

finally {

sqlsession.close();

}复制代码

### 13、如何获取自动生成的(主)键值?

insert 方法总是返回一个 int 值 ，这个值代表的是插入的行数。

如果采用自增长策略，自动生成的键值在 insert 方法执行完后可以被设置到传入的参数对象中。

示例：

<insert id=”insertname” usegeneratedkeys=”true” keyproperty=”

id”>

insert into names (name) values (#{ name

}

)

</insert>

name name = new name();

name.setname(“fred”);

int rows = mapper.insertname(name);

// 完成后,id 已经被设置到对象中

system.out.println(“rows inserted = ” + rows); system.out.println(“generated key value = ” + name.getid());复制代码

### 14、在 mapper 中如何传递多个参数?

第一种：DAO 层的函数

public UserselectUser(String name,String area);

对应的 xml,#{0}代表接收的是 dao 层中的第一个参数，#{1}代表 dao 层中第二参数，更多参数一致往后加即可。

<select id="selectUser"resultMap="BaseResultMap"> select \* fromuser\_user\_t whereuser\_name = #{0} anduser\_area=#{1}

</select>复制代码

第二种： 使用 @param 注解:

public interface usermapper {

user selectuser(@param(“username”) string

username,@param(“hashedpassword”) string hashedpassword);

}复制代码

然后,就可以在 xml 像下面这样使用(推荐封装为一个 map,作为单个参数传递给 mapper):

<select id=”selectuser” resulttype=”user”> select id, username, hashedpassword

from some\_tablewhere username = #{username}

and hashedpassword = #{hashedpassword}

</select>复制代码

第三种：多个参数封装成 map

try {

//映射文件的命名空间.SQL 片段的 ID，就可以调用对应的映射文件中的SQL

//由于我们的参数超过了两个，而方法中只有一个 Object 参数收集，因此

我们使用 Map 集合来装载我们的参数

Map < String, Object > map = new HashMap(); map.put("start", start);

map.put("end", end);

return sqlSession.selectList("StudentID.pagination", map);

}

catch (Exception e) {

e.printStackTrace(); sqlSession.rollback(); throw e;

}

finally {

MybatisUtil.closeSqlSession();

}复制代码

### 15、Mybatis 动态 sql 有什么用？执行原理？有哪些动态 sql？

Mybatis 动态 sql 可以在 Xml 映射文件内，以标签的形式编写动态 sql，执行原理是根据表达式的值 完成逻辑判断并动态拼接 sql 的功能。

Mybatis 提供了 9 种动态 sql 标签：trim | where | set | foreach | if | choose| when | otherwise | bind。

### 16、Xml 映射文件中，除了常见的 select|insert|updae|delete 标签之外，还有哪些标签？

答：<resultMap>、<parameterMap>、<sql>、<include>、<selectKey>，加上动态 sql 的 9 个标签，其中<sql>为 sql 片段标签，通过<include>标签引入 sql 片段，<selectKey>为不支持自增的主键生成策略标签。

### 17、Mybatis 的 Xml 映射文件中，不同的 Xml 映射文件，id 是否可以重复？

不同的 Xml 映射文件，如果配置了 namespace，那么 id 可以重复；如果没有配置 namespace，那么 id 不能重复；

原因就是 namespace+id 是作为 Map<String, MapperStatement>的 key 使用的，如果没有 namespace，就剩下 id，那么，id 重复会导致数据互相覆盖。有了 namespace，自然 id 就可以重复，namespace 不同，namespace+id 自然也就不同。

### 18、为什么说 Mybatis 是半自动 ORM 映射工具？它与全自动的区别在哪里？

Hibernate 属于全自动 ORM 映射工具，使用 Hibernate 查询关联对象或者关联集合对象时，可以根据对象关系模型直接获取，所以它是全自动的。而 Mybatis 在查询关联对象或关联集合对象时，需要手动编写 sql 来完成，所以，称之为半自动 ORM 映射工具。

### 19、 一对一、一对多的关联查询 ？

<mapper namespace="com.lcb.mapping.userMapper">

<!--association 一对一关联查询 -->

<select id="getClass" parameterType="int" resultMap="ClassesResultMap">

select \* from class c,teacher t where c.teacher\_id=t.t\_id and c.c\_id=#{id}

</select>

<resultMap type="com.lcb.user.Classes" id="ClassesResultMap">

<!-- 实体类的字段名和数据表的字段名映射 -->

<id property="id" column="c\_id"/>

<result property="name" column="c\_name"/>

<association property="teacher" javaType="com.lcb.user.Teacher">

<id property="id" column="t\_id"/>

<result property="name" column="t\_name"/>

</association>

</resultMap>

<!--collection 一对多关联查询 -->

<select id="getClass2" parameterType="int" resultMap="ClassesResultMap2">

select \* from class c,teacher t,student s where c.teacher\_id=t.t\_id and c.c\_id=s.class\_id and c.c\_id=#{id}

</select>

<resultMap type="com.lcb.user.Classes" id="ClassesResultMap2">

<id property="id" column="c\_id"/>

<result property="name" column="c\_name"/>

<association property="teacher" javaType="com.lcb.user.Teacher">

<id property="id" column="t\_id"/>

<result property="name" column="t\_name"/>

</association>

<collection property="student" ofType="com.lcb.user.Student">

<id property="id" column="s\_id"/>

<result property="name" column="s\_name"/>

</collection>

</resultMap>

</mapper>复制代码

### 20、MyBatis 实现一对一有几种方式?具体怎么操作的？

有联合查询和嵌套查询,联合查询是几个表联合查询,只查询一次, 通过在resultMap 里面配置 association 节点配置一对一的类就可以完成；嵌套查询是先查一个表，根据这个表里面的结果的 外键 id，去再另外一个表里面查询数据,也是通过 association 配置，但另外一个表的查询通过 select 属性配置。

### 21、MyBatis 实现一对多有几种方式,怎么操作的？

有联合查询和嵌套查询。联合查询是几个表联合查询,只查询一次,通过在resultMap 里面的 collection 节点配置一对多的类就可以完成；嵌套查询是先

查一个表,根据这个表里面的 结果的外键 id,去再另外一个表里面查询数据,也是通过配置 collection,但另外一个表的查询通过 select 节点配置。

### 22、Mybatis 是否支持延迟加载？如果支持，它的实现原理是什么？

答：Mybatis 仅支持 association 关联对象和 collection 关联集合对象的延迟加载，association 指的就是一对一，collection 指的就是一对多查询。在Mybatis 配置文件中，可以配置是否启用延迟加载lazyLoadingEnabled=true|false。

它的原理是，使用 CGLIB 创建目标对象的代理对象，当调用目标方法时，进入拦截器方法，比如调用 a.getB().getName()，拦截器 invoke()方法发现a.getB()是 null 值，那么就会单独发送事先保存好的查询关联 B 对象的 sql， 把 B 查询上来，然后调用 a.setB(b)，于是 a 的对象 b 属性就有值了，接着完成 a.getB().getName()方法的调用。这就是延迟加载的基本原理。

当然了，不光是 Mybatis，几乎所有的包括 Hibernate，支持延迟加载的原理都是一样的。

### 23、Mybatis 的一级、二级缓存:

1. 一级缓存: 基于 PerpetualCache 的 HashMap 本地缓存，其存储作用域为Session，当 Session flush 或 close 之后，该 Session 中的所有 Cache 就将清空，默认打开一级缓存。
2. 二级缓存与一级缓存其机制相同，默认也是采用 PerpetualCache，HashMap 存储，不同在于其存储作用域为 Mapper(Namespace)，并且可自定义存储源，如Ehcache。默认不打开二级缓存，要开启二级缓存，使用二级缓存属性类需要实现 Serializable 序列化接口(可用来保存对象的状态),可在它的映射文件中配置<cache/> ；
3. 对于缓存数据更新机制，当某一个作用域(一级缓存 Session/二级缓存Namespaces)的进行了 C/U/D 操作后，默认该作用域下所有 select 中的缓存将被 clear。

### 24、什么是 MyBatis 的接口绑定？有哪些实现方式？

接口绑定，就是在 MyBatis 中任意定义接口,然后把接口里面的方法和 SQL 语句绑定, 我们直接调用接口方法就可以,这样比起原来了 SqlSession 提供的方法我们可以有更加灵活的选择和设置。

接口绑定有两种实现方式,一种是通过注解绑定，就是在接口的方法上面加上@Select、@Update 等注解，里面包含 Sql 语句来绑定；另外一种就是通过 xml 里面写 SQL 来绑定, 在这种情况下,要指定 xml 映射文件里面的 namespace 必须为接口的全路径名。当 Sql 语句比较简单时候,用注解绑定, 当 SQL 语句比较复杂时候,用 xml 绑定,一般用 xml 绑定的比较多。

### 25、使用 MyBatis 的 mapper 接口调用时有哪些要求？

1. Mapper 接口方法名和 mapper.xml 中定义的每个 sql 的 id 相同；
2. Mapper 接口方法的输入参数类型和 mapper.xml 中定义的每个 sql 的parameterType 的类型相同；
3. Mapper 接口方法的输出参数类型和 mapper.xml 中定义的每个 sql 的resultType 的类型相同；
4. Mapper.xml 文件中的 namespace 即是 mapper 接口的类路径。

### 26、Mapper 编写有哪几种方式？

第一种：接口实现类继承 SqlSessionDaoSupport：使用此种方法需要编写mapper 接口，mapper 接口实现类、mapper.xml 文件。

1. 在 sqlMapConfig.xml 中配置 mapper.xml 的位置

<mappers>

<mapper resource="mapper.xml 文件的地址" />

<mapper resource="mapper.xml 文件的地址" />

</mappers>复制代码

1. 定义 mapper 接口
2. 实现类集成 SqlSessionDaoSupport

mapper 方法中可以 this.getSqlSession()进行数据增删改查。

1. spring 配置

<bean id=" " class="mapper 接口的实现">

<property name="sqlSessionFactory" ref="sqlSessionFactory"></property>

</bean>复制代码

第二种：使用 org.mybatis.spring.mapper.MapperFactoryBean：

1. 在 sqlMapConfig.xml 中配置 mapper.xml 的位置，如果 mapper.xml 和mappre 接口的名称相同且在同一个目录，这里可以不用配置

<mappers>

<mapper resource="mapper.xml 文件的地址" />

<mapper resource="mapper.xml 文件的地址" />

</mappers>复制代码

1. 定义 mapper 接口：
2. mapper.xml 中的 namespace 为 mapper 接口的地址
3. mapper 接口中的方法名和 mapper.xml 中的定义的 statement 的 id 保持一致
4. Spring 中定义

<bean id="" class="org.mybatis.spring.mapper.MapperFactoryBean">

<property name="mapperInterface" value="mapper 接口地址" />

<property name="sqlSessionFactory" ref="sqlSessionFactory" />

</bean>复制代码

第三种：使用 mapper 扫描器：

1. mapper.xml 文件编写：

mapper.xml 中的 namespace 为 mapper 接口的地址；

mapper 接口中的方法名和 mapper.xml 中的定义的 statement 的 id 保持一致；

如果将 mapper.xml 和 mapper 接口的名称保持一致则不用在sqlMapConfig.xml 中进行配置。

1. 定义 mapper 接口：

注意 mapper.xml 的文件名和 mapper 的接口名称保持一致，且放在同一个目录

1. 配置 mapper 扫描器：

<bean class="org.mybatis.spring.mapper.MapperScannerConfigurer">

<property name="basePackage" value="mapper 接口包地址"></property>

<property name="sqlSessionFactoryBeanName" value="sqlSessionFactory"/>

</bean>复制代码

1. 使用扫描器后从 spring 容器中获取 mapper 的实现对象。

### 27、简述 Mybatis 的插件运行原理，以及如何编写一个插件。

答：Mybatis 仅可以编写针对 ParameterHandler、ResultSetHandler、StatementHandler、Executor 这 4 种接口的插件，Mybatis 使用 JDK 的动态代理，为需要拦截的接口生成代理对象以实现接口方法拦截功能，每当执行这 4 种接口对象的方法时，就会进入拦截方法，具体就是 InvocationHandler 的invoke()方法，当然，只会拦截那些你指定需要拦截的方法。

编写插件：实现 Mybatis 的 Interceptor 接口并复写 intercept()方法，然后在给插件编写注解，指定要拦截哪一个接口的哪些方法即可，记住，别忘了在配置文件中配置你编写的插件。

# 六、ZooKeeper 面试题答案解析

### ZooKeeper 是什么？

ZooKeeper 是一个开放源码的分布式协调服务，它是集群的管理者，监视着集群中各个节点的状态根据节点提交的反馈进行下一步合理操作。最终，将简单易用的接口和性能高效、功能稳定的系统提供给用户。

分布式应用程序可以基于 Zookeeper 实现诸如数据发布/订阅、负载均衡、命名服务、分布式协调/通知、集群管理、Master 选举、分布式锁和分布式队列等功能。

Zookeeper 保证了如下分布式一致性特性：

1. 顺序一致性
2. 原子性
3. 单一视图
4. 可靠性
5. 实时性（最终一致性）

客户端的读请求可以被集群中的任意一台机器处理，如果读请求在节点上注册了监听器，这个监听器也是由所连接的 zookeeper 机器来处理。对于写请求，这些请求会同时发给其他 zookeeper 机器并且达成一致后，请求才会返回成功。因此，随着 zookeeper 的集群机器增多，读请求的吞吐会提高但是写请求的吞吐会下降。

有序性是 zookeeper 中非常重要的一个特性，所有的更新都是全局有序的，每个更新都有一个唯一的时间戳，这个时间戳称为 zxid（Zookeeper Transaction Id）。而读请求只会相对于更新有序，也就是读请求的返回结果中会带有这个zookeeper 最新的 zxid。

### ZooKeeper 提供了什么？

1. 文件系统
2. 通知机制

### Zookeeper 文件系统

Zookeeper 提供一个多层级的节点命名空间（节点称为 znode）。与文件系统不同的是，这些节点都可以设置关联的数据，而文件系统中只有文件节点可以存放数据而目录节点不行。

Zookeeper 为了保证高吞吐和低延迟，在内存中维护了这个树状的目录结构，这种特性使得 Zookeeper 不能用于存放大量的数据，每个节点的存放数据上限为1M。

### ZAB 协议？

ZAB 协议是为分布式协调服务 Zookeeper 专门设计的一种支持崩溃恢复的原子广播协议。

ZAB 协议包括两种基本的模式：崩溃恢复和消息广播。

当整个 zookeeper 集群刚刚启动或者 Leader 服务器宕机、重启或者网络故障导致不存在过半的服务器与 Leader 服务器保持正常通信时，所有进程（服务器） 进入崩溃恢复模式，首先选举产生新的 Leader 服务器，然后集群中 Follower 服务器开始与新的 Leader 服务器进行数据同步，当集群中超过半数机器与该

Leader 服务器完成数据同步之后，退出恢复模式进入消息广播模式，Leader 服务器开始接收客户端的事务请求生成事物提案来进行事务请求处理。

### 四种类型的数据节点 Znode

1. PERSISTENT-持久节点

除非手动删除，否则节点一直存在于 Zookeeper 上

1. EPHEMERAL-临时节点

临时节点的生命周期与客户端会话绑定，一旦客户端会话失效（客户端与zookeeper 连接断开不一定会话失效），那么这个客户端创建的所有临时节点都会被移除。

1. PERSISTENT\_SEQUENTIAL-持久顺序节点

基本特性同持久节点，只是增加了顺序属性，节点名后边会追加一个由父节点维护的自增整型数字。

1. EPHEMERAL\_SEQUENTIAL-临时顺序节点

基本特性同临时节点，增加了顺序属性，节点名后边会追加一个由父节点维护的自增整型数字。

### Zookeeper Watcher 机制 -- 数据变更通知

Zookeeper 允许客户端向服务端的某个 Znode 注册一个 Watcher 监听，当服务端的一些指定事件触发了这个 Watcher，服务端会向指定客户端发送一个事件通知来实现分布式的通知功能，然后客户端根据 Watcher 通知状态和事件类型做出业务上的改变。

工作机制：

1. 客户端注册 watcher
2. 服务端处理 watcher
3. 客户端回调 watcher Watcher 特性总结：
4. 一次性

无论是服务端还是客户端，一旦一个 Watcher 被 触 发 ，Zookeeper 都会将其从相应的存储中移除。这样的设计有效的减轻了服务端的压力，不然对于更新非常频繁的节点，服务端会不断的向客户端发送事件通知，无论对于网络还是服务端的压力都非常大。

1. 客户端串行执行

客户端 Watcher 回调的过程是一个串行同步的过程。

1. 轻量
   1. 、Watcher 通知非常简单，只会告诉客户端发生了事件，而不会说明事件的具体内容。
   2. 、客户端向服务端注册 Watcher 的时候，并不会把客户端真实的 Watcher 对象实体传递到服务端，仅仅是在客户端请求中使用 boolean 类型属性进行了标记。
2. watcher event 异步发送 watcher 的通知事件从 server 发送到 client 是异步的，这就存在一个问题，不同的客户端和服务器之间通过 socket 进行通信，由于网络延迟或其他因素导致客户端在不通的时刻监听到事件，由于Zookeeper 本身提供了 ordering guarantee，即客户端监听事件后，才会感知它所监视 znode 发生了变化。所以我们使用 Zookeeper 不能期望能够监控到节点每次的变化。Zookeeper 只能保证最终的一致性，而无法保证强一致性。
3. 注册 watcher getData、exists、getChildren
4. 触发 watcher create、delete、setData
5. 当一个客户端连接到一个新的服务器上时，watch 将会被以任意会话事件触发。当与一个服务器失去连接的时候，是无法接收到 watch 的。而当 client 重新连接时，如果需要的话，所有先前注册过的 watch，都会被重新注册。通常这是完全透明的。只有在一个特殊情况下，watch 可能会丢失：对于一个未创建的 znode 的 exist watch，如果在客户端断开连接期间被创建了，并且随后在客户端连接上之前又删除了，这种情况下，这个 watch 事件可能会被丢失。

### 客户端注册 Watcher 实现

1. 调用 getData()/getChildren()/exist()三个 API，传入 Watcher 对象
2. 标记请求 request，封装 Watcher 到 WatchRegistration
3. 封装成 Packet 对象，发服务端发送 request
4. 收到服务端响应后，将 Watcher 注册到 ZKWatcherManager 中进行管理
5. 请求返回，完成注册。

### 服务端处理 Watcher 实现

1. 服务端接收 Watcher 并存储

接收到客户端请求，处理请求判断是否需要注册 Watcher，需要的话将数据节点的节点路径和 ServerCnxn（ServerCnxn 代表一个客户端和服务端的连接，实现了 Watcher 的 process 接口，此时可以看成一个 Watcher 对象）存储在WatcherManager 的 WatchTable 和 watch2Paths 中去。

1. Watcher 触发

以服务端接收到 setData() 事务请求触发 NodeDataChanged 事件为例：

* 1. 封装 WatchedEvent

将通知状态（SyncConnected）、事件类型（NodeDataChanged）以及节点路径封装成一个 WatchedEvent 对象

* 1. 查询 Watcher

从 WatchTable 中根据节点路径查找 Watcher

* 1. 没找到；说明没有客户端在该数据节点上注册过 Watcher
  2. 找到；提取并从 WatchTable 和 Watch2Paths 中删除对应 Watcher（从这里可以看出 Watcher 在服务端是一次性的，触发一次就失效了）

1. 调用 process 方法来触发 Watcher

这里 process 主要就是通过 ServerCnxn 对应的 TCP 连接发送 Watcher 事件通知。

### 客户端回调 Watcher

客户端 SendThread 线程接收事件通知，交由 EventThread 线程回调 Watcher。客户端的 Watcher 机制同样是一次性的，一旦被触发后，该 Watcher 就失效了。

### ACL 权限控制机制

UGO（User/Group/Others）

目前在 Linux/Unix 文件系统中使用，也是使用最广泛的权限控制方式。是一种粗粒度的文件系统权限控制模式。

ACL（Access Control List）访问控制列表包括三个方面：

权限模式（Scheme）

1. IP：从 IP 地址粒度进行权限控制
2. Digest：最常用，用类似于 username:password 的权限标识来进行权限配置，便于区分不同应用来进行权限控制
3. World：最开放的权限控制方式，是一种特殊的 digest 模式，只有一个权限标识“world:anyone”
4. Super：超级用户授权对象

授权对象指的是权限赋予的用户或一个指定实体，例如 IP 地址或是机器灯。权限 Permission

1. CREATE：数据节点创建权限，允许授权对象在该 Znode 下创建子节点
2. DELETE：子节点删除权限，允许授权对象删除该数据节点的子节点
3. READ：数据节点的读取权限，允许授权对象访问该数据节点并读取其数据内容或子节点列表等
4. WRITE：数据节点更新权限，允许授权对象对该数据节点进行更新操作
5. ADMIN：数据节点管理权限，允许授权对象对该数据节点进行 ACL 相关设置操作

### Chroot 特性

3.2.0 版本后，添加了 Chroot 特性，该特性允许每个客户端为自己设置一个命名空间。如果一个客户端设置了 Chroot，那么该客户端对服务器的任何操作， 都将会被限制在其自己的命名空间下。

通过设置 Chroot，能够将一个客户端应用于 Zookeeper 服务端的一颗子树相对应，在那些多个应用公用一个 Zookeeper 进群的场景下，对实现不同应用间的相互隔离非常有帮助。

### 会话管理

分桶策略：将类似的会话放在同一区块中进行管理，以便于 Zookeeper 对会话进行不同区块的隔离处理以及同一区块的统一处理。

分配原则：每个会话的“下次超时时间点”（ExpirationTime） 计算公式：

ExpirationTime\_ = currentTime + sessionTimeout

ExpirationTime = (ExpirationTime\_ / ExpirationInrerval + 1) \* ExpirationInterval , ExpirationInterval 是指 Zookeeper 会话超时检查时间间隔，默认 tickTime

### 服务器角色

Leader

1. 事务请求的唯一调度和处理者，保证集群事务处理的顺序性
2. 集群内部各服务的调度者Follower
3. 处理客户端的非事务请求，转发事务请求给 Leader 服务器
4. 参与事务请求 Proposal 的投票
5. 参与 Leader 选举投票Observer

（1）3.0 版本以后引入的一个服务器角色，在不影响集群事务处理能力的基础上提升集群的非事务处理能力

1. 处理客户端的非事务请求，转发事务请求给 Leader 服务器
2. 不参与任何形式的投票

### Zookeeper 下 Server 工作状态

服务器具有四种状态，分别是 LOOKING、FOLLOWING、LEADING、OBSERVING。

1. LOOKING：寻 找 Leader 状态。当服务器处于该状态时，它会认为当前集群中没有 Leader，因此需要进入 Leader 选举状态。
2. FOLLOWING：跟随者状态。表明当前服务器角色是 Follower。
3. LEADING：领导者状态。表明当前服务器角色是 Leader。
4. OBSERVING：观察者状态。表明当前服务器角色是 Observer。

### 数据同步

整个集群完成 Leader 选举之后，Learner（Follower 和 Observer 的统称）回向 Leader 服务器进行注册。当 Learner 服务器想 Leader 服务器完成注册后， 进入数据同步环节。

数据同步流程：（均以消息传递的方式进行）

Learner 向 Learder 注册数据同步

同步确认

Zookeeper 的数据同步通常分为四类：

1. 直接差异化同步（DIFF 同步）
2. 先回滚再差异化同步（TRUNC+DIFF 同步）
3. 仅回滚同步（TRUNC 同步）
4. 全量同步（SNAP 同步）

在进行数据同步前，Leader 服务器会完成数据同步初始化： peerLastZxid：

* 从 learner 服务器注册时发送的 ACKEPOCH 消息中提取 lastZxid（该Learner 服务器最后处理的 ZXID）

minCommittedLog：

* Leader 服务器 Proposal 缓存队列 committedLog 中最小ZXIDmaxCommittedLog：
* Leader 服务器 Proposal 缓存队列 committedLog 中最大 ZXID 直接差异化同步（DIFF 同步）
* 场景：peerLastZxid 介于 minCommittedLog 和 maxCommittedLog 之间先回滚再差异化同步（TRUNC+DIFF 同步）
* 场景：当新的 Leader 服务器发现某个 Learner 服务器包含了一条自己没有的事务记录，那么就需要让该 Learner 服务器进行事务回滚--回滚到 Leader 服务器上存在的，同时也是最接近于 peerLastZxid 的 ZXID 仅回滚同步（TRUNC 同步）
* 场景：peerLastZxid 大于 maxCommittedLog 全量同步（SNAP 同步）
* 场景一：peerLastZxid 小于 minCommittedLog
* 场景二：Leader 服务器上没有 Proposal 缓存队列且 peerLastZxid 不等于lastProcessZxid

### zookeeper 是如何保证事务的顺序一致性的？

zookeeper 采用了全局递增的事务 Id 来标识，所有的 proposal（提议）都在被提出的时候加上了 zxid，zxid 实际上是一个 64 位的数字，高 32 位是epoch（ 时期; 纪元; 世; 新时代）用来标识 leader 周期，如果有新的 leader 产生出来，epoch 会自增，低 32 位用来递增计数。当新产生 proposal 的时候， 会依据数据库的两阶段过程，首先会向其他的 server 发出事务执行请求，如果超过半数的机器都能执行并且能够成功，那么就会开始执行。

### 分布式集群中为什么会有 Master？

在分布式环境中，有些业务逻辑只需要集群中的某一台机器进行执行，其他的机器可以共享这个结果，这样可以大大减少重复计算，提高性能，于是就需要进行leader 选举。

### zk 节点宕机如何处理？

Zookeeper 本身也是集群，推荐配置不少于 3 个服务器。Zookeeper 自身也要保证当一个节点宕机时，其他节点会继续提供服务。

如果是一个 Follower 宕机，还有 2 台服务器提供访问，因为 Zookeeper 上的数据是有多个副本的，数据并不会丢失；

如果是一个 Leader 宕机，Zookeeper 会选举出新的 Leader。

ZK 集群的机制是只要超过半数的节点正常，集群就能正常提供服务。只有在 ZK 节点挂得太多，只剩一半或不到一半节点能工作，集群才失效。

所以

3 个节点的 cluster 可以挂掉 1 个节点(leader 可以得到 2 票>1.5)

2 个节点的 cluster 就不能挂掉任何 1 个节点了(leader 可以得到 1 票<=1)

### zookeeper 负载均衡和 nginx 负载均衡区别

zk 的负载均衡是可以调控，nginx 只是能调权重，其他需要可控的都需要自己写插件；但是 nginx 的吞吐量比 zk 大很多，应该说按业务选择用哪种方式。

### Zookeeper 有哪几种几种部署模式？

部署模式：单机模式、伪集群模式、集群模式。

### 集群最少要几台机器，集群规则是怎样的?

集群规则为 2N+1 台，N>0，即 3 台。

### 集群支持动态添加机器吗？

其实就是水平扩容了，Zookeeper 在这方面不太好。两种方式：

全部重启：关闭所有 Zookeeper 服务，修改配置之后启动。不影响之前客户端的会话。

逐个重启：在过半存活即可用的原则下，一台机器重启不影响整个集群对外提供服务。这是比较常用的方式。

3.5 版本开始支持动态扩容。

### Zookeeper 对节点的 watch 监听通知是永久的吗？为什么不是永久的?

不是。官方声明：一个 Watch 事件是一个一次性的触发器，当被设置了 Watch 的数据发生了改变的时候，则服务器将这个改变发送给设置了 Watch 的客户端， 以便通知它们。

为什么不是永久的，举个例子，如果服务端变动频繁，而监听的客户端很多情况下，每次变动都要通知到所有的客户端，给网络和服务器造成很大压力。

一般是客户端执行 getData(“/节点 A”,true)，如果节点 A 发生了变更或删除，客户端会得到它的 watch 事件，但是在之后节点 A 又发生了变更，而客户端又没有设置 watch 事件，就不再给客户端发送。

在实际应用中，很多情况下，我们的客户端不需要知道服务端的每一次变动，我只要最新的数据即可。

### Zookeeper 的 java 客户端都有哪些？

java 客户端：zk 自带的 zkclient 及 Apache 开源的 Curator。

### chubby 是什么，和 zookeeper 比你怎么看？

chubby 是 google 的，完全实现 paxos 算法，不开源。zookeeper 是 chubby 的开源实现，使用 zab 协议，paxos 算法的变种。

### 说几个 zookeeper 常用的命令。

常用命令：ls get set create delete 等。

### ZAB 和 Paxos 算法的联系与区别？

相同点：

1. 两者都存在一个类似于 Leader 进程的角色，由其负责协调多个Follower进程的运行
2. Leader 进程都会等待超过半数的 Follower 做出正确的反馈后，才会将一个提案进行提交
3. ZAB 协议中，每个 Proposal 中都包含一个 epoch 值来代表当前的Leader周期，Paxos 中名字为 Ballot

不同点：

ZAB 用来构建高可用的分布式数据主备系统（Zookeeper），Paxos 是用来构建分布式一致性状态机系统。

### Zookeeper 的典型应用场景

Zookeeper 是一个典型的发布/订阅模式的分布式数据管理与协调框架，开发人员可以使用它来进行分布式数据的发布和订阅。

通过对 Zookeeper 中丰富的数据节点进行交叉使用，配合 Watcher 事件通知机制，可以非常方便的构建一系列分布式应用中年都会涉及的核心功能，如：

1. 数据发布/订阅
2. 负载均衡
3. 命名服务
4. 分布式协调/通知
5. 集群管理
6. Master 选举
7. 分布式锁
8. 分布式队列

**数据发布*/*订阅**介绍

数据发布/订阅系统，即所谓的配置中心，顾名思义就是发布者发布数据供订阅者进行数据订阅。

目的

动态获取数据（配置信息）

实现数据（配置信息）的集中式管理和数据的动态更新设计模式

Push 模式Pull 模式

数据（配置信息）特性

1. 数据量通常比较小
2. 数据内容在运行时会发生动态更新
3. 集群中各机器共享，配置一致

如：机器列表信息、运行时开关配置、数据库配置信息等基于 Zookeeper 的实现方式

* 数据存储：将数据（配置信息）存储到 Zookeeper 上的一个数据节点
* 数据获取：应用在启动初始化节点从 Zookeeper 数据节点读取数据，并在该节点上注册一个数据变更 Watcher
* 数据变更：当变更数据时，更新 Zookeeper 对应节点数据，Zookeeper 会将数据变更通知发到各客户端，客户端接到通知后重新读取变更后的数据即可。

#### 负载均衡

zk 的命名服务

命名服务是指通过指定的名字来获取资源或者服务的地址，利用 zk 创建一个全局的路径，这个路径就可以作为一个名字，指向集群中的集群，提供的服务的地址，或者一个远程的对象等等。

#### 分布式通知和协调

对于系统调度来说：操作人员发送通知实际是通过控制台改变某个节点的状态， 然后 zk 将这些变化发送给注册了这个节点的 watcher 的所有客户端。

对于执行情况汇报：每个工作进程都在某个目录下创建一个临时节点。并携带工作的进度数据，这样汇总的进程可以监控目录子节点的变化获得工作进度的实时的全局情况。

#### zk 的命名服务（文件系统）

命名服务是指通过指定的名字来获取资源或者服务的地址，利用 zk 创建一个全局的路径，即是唯一的路径，这个路径就可以作为一个名字，指向集群中的集群， 提供的服务的地址，或者一个远程的对象等等。

#### zk 的配置管理（文件系统、通知机制）

程序分布式的部署在不同的机器上，将程序的配置信息放在 zk 的 znode 下， 当有配置发生改变时，也就是 znode 发生变化时，可以通过改变 zk 中某个目录节点的内容，利用 watcher 通知给各个客户端，从而更改配置。**Zookeeper 集群管理（文件系统、通知机制）**

所谓集群管理无在乎两点：是否有机器退出和加入、选举 master。

对于第一点，所有机器约定在父目录下创建临时目录节点，然后监听父目录节点的子节点变化消息。一旦有机器挂掉，该机器与 zookeeper 的连接断开，其所创建的临时目录节点被删除，所有其他机器都收到通知：某个兄弟目录被删除， 于是，所有人都知道：它上船了。

新机器加入也是类似，所有机器收到通知：新兄弟目录加入，highcount 又有了， 对于第二点，我们稍微改变一下，所有机器创建临时顺序编号目录节点，每次选取编号最小的机器作为 master 就好。

#### Zookeeper 分布式锁（文件系统、通知机制）

有了 zookeeper 的一致性文件系统，锁的问题变得容易。锁服务可以分为两类， 一个是保持独占，另一个是控制时序。

对于第一类，我们将 zookeeper 上的一个 znode 看作是一把锁，通过createznode 的方式来实现。所有客户端都去创建 /distribute\_lock 节点，最终成功创建的那个客户端也即拥有了这把锁。用完删除掉自己创建的distribute\_lock 节点就释放出锁。

对于第二类， /distribute\_lock 已经预先存在，所有客户端在它下面创建临时顺序编号目录节点，和选 master 一样，编号最小的获得锁，用完删除，依次方便。

Zookeeper 队列管理（文件系统、通知机制） 两种类型的队列：

1. 同步队列，当一个队列的成员都聚齐时，这个队列才可用，否则一直等待所有成员到达。
2. 队列按照 FIFO 方式进行入队和出队操作。

第一类，在约定目录下创建临时目录节点，监听节点数目是否是我们要求的数目。

第二类，和分布式锁服务中的控制时序场景基本原理一致，入列有编号，出列按编号。在特定的目录下创建 PERSISTENT\_SEQUENTIAL 节点，创建成功时 Watcher 通知等待的队列，队列删除序列号最小的节点用以消费。此场景下 Zookeeper 的znode 用于消息存储，znode 存储的数据就是消息队列中的消息内容， SEQUENTIAL 序列号就是消息的编号，按序取出即可。由于创建的节点是持久化的，所以不必担心队列消息的丢失问题。

# 七、Spring Boot 面试题

## 1、什么是 Spring Boot？

多年来，随着新功能的增加，spring 变得越来越复杂。访问 spring 官网页面， 我们就会看到可以在我们的应用程序中使用的所有 Spring 项目的不同功能。如果必须启动一个新的 Spring 项目，我们必须添加构建路径或添加 Maven 依赖关系，配置应用程序服务器，添加 spring 配置。因此，开始一个新的 spring 项目需要很多努力，因为我们现在必须从头开始做所有事情。

Spring Boot 是解决这个问题的方法。Spring Boot 已经建立在现有 spring 框架之上。使用 spring 启动，我们避免了之前我们必须做的所有样板代码和配置。因此，Spring Boot 可以帮助我们以最少的工作量，更加健壮地使用现有的Spring 功能。

## 2、Spring Boot 有哪些优点？

Spring Boot 的优点有：

1、减少开发，测试时间和努力。

2、使用 JavaConfig 有助于避免使用 XML。3、避免大量的 Maven 导入和各种版本冲突。

4、提供意见发展方法。

5、通过提供默认值快速开始开发。

6、没有单独的 Web 服务器需要。这意味着你不再需要启动 Tomcat，Glassfish 或其他任何东西。

7、需要更少的配置 因为没有 web.xml 文件。只需添加用@ Configuration 注释的类，然后添加用@Bean 注释的方法，Spring 将自动加载对象并像以前一样对其进行管理。您甚至可以将@Autowired 添加到 bean 方法中，以使 Spring 自动装入需要的依赖关系中。

8、基于环境的配置 使用这些属性，您可以将您正在使用的环境传递到应用程序：

-Dspring.profiles.active = {enviornment}。在加载主应用程序属性文件后， Spring 将在（application{environment} .properties）中加载后续的应用程序属性文件。

## 3、什么是 JavaConfig？

Spring JavaConfig 是 Spring 社区的产品，它提供了配置 Spring IoC 容器的纯 Java 方法。因此它有助于避免使用 XML 配置。使用 JavaConfig 的优点在于：

1. 面向对象的配置。由于配置被定义为 JavaConfig 中的类，因此用户可以充分利用 Java 中的面向对象功能。一个配置类可以继承另一个，重写它的@Bean 方法等。
2. 减少或消除 XML 配置。基于依赖注入原则的外化配置的好处已被证明。但是，许多开发人员不希望在 XML 和 Java 之间来回切换。JavaConfig 为开发人员提供了一种纯 Java 方法来配置与 XML 配置概念相似的 Spring 容器。从技术角度来讲，只使用 JavaConfig 配置类来配置容器是可行的，但实际上很多人认为将 JavaConfig 与 XML 混合匹配是理想的。
3. 类型安全和重构友好。JavaConfig 提供了一种类型安全的方法来配置Spring 容器。由于 Java 5.0 对泛型的支持，现在可以按类型而不是按名称检索 bean，不需要任何强制转换或基于字符串的查找。

## 4、如何重新加载 Spring Boot 上的更改，而无需重新启动服务器？

这可以使用 DEV 工具来实现。通过这种依赖关系，您可以节省任何更改，嵌入式 tomcat 将重新启动。Spring Boot 有一个开发工具（DevTools）模块，它有助于提高开发人员的生产力。Java 开发人员面临的一个主要挑战是将文件更改自动部署到服务器并自动重启服务器。开发人员可以重新加载 Spring Boot 上的更改，而无需重新启动服务器。这将消除每次手动部署更改的需要。Spring Boot 在发布它的第一个版本时没有这个功能。这是开发人员最需要的功能。DevTools 模块完全满足开发人员的需求。该模块将在生产环境中被禁用。它还提供 H2 数据库控制台以更好地测试应用程序。

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-devtools</artifactId>

<optional>true</optional>复制代码

## 5、Spring Boot 中的监视器是什么？

Spring boot actuator 是 spring 启动框架中的重要功能之一。Spring boot 监视器可帮助您访问生产环境中正在运行的应用程序的当前状态。有几个指标必须在生产环境中进行检查和监控。即使一些外部应用程序可能正在使用这些服务来向相关人员触发警报消息。监视器模块公开了一组可直接作为 HTTP URL 访问的REST 端点来检查状态。

## 6、如何在 Spring Boot 中禁用 Actuator 端点安全性？

默认情况下，所有敏感的 HTTP 端点都是安全的，只有具有 ACTUATOR 角色的用户才能访问它们。安全性是使用标准的 HttpServletRequest.isUserInRole 方法实施的。 我们可以使用来禁用安全性。只有在执行机构端点在防火墙后访问时，才建议禁用安全性。

## 7、如何在自定义端口上运行 Spring Boot 应用程序？

为了在自定义端口上运行 Spring Boot 应用程序，您可以在application.properties 中指定端口。server.port = 8090

## 8、什么是 YAML？

YAML 是一种人类可读的数据序列化语言。它通常用于配置文件。与属性文件相比，如果我们想要在配置文件中添加复杂的属性，YAML 文件就更加结构化，而且更少混淆。可以看出 YAML 具有分层配置数据。

## 9、如何实现 Spring Boot 应用程序的安全性？

为了实现 Spring Boot 的安全性，我们使用 spring-boot-starter-security 依赖项，并且必须添加安全配置。它只需要很少的代码。配置类将必须扩展WebSecurityConfigurerAdapter 并覆盖其方法。

## 10、如何集成 Spring Boot 和 ActiveMQ？

对于集成 Spring Boot 和 ActiveMQ，我们使用依赖关系。 它只需要很少的配置，并且不需要样板代码。

## 11、如何使用 Spring Boot 实现分页和排序？

使用 Spring Boot 实现分页非常简单。使用 Spring Data-JPA 可以实现将可分页的传递给存储库方法。

## 12、什么是 Swagger？你用 Spring Boot 实现了它吗？

Swagger 广泛用于可视化 API，使用 Swagger UI 为前端开发人员提供在线沙 箱。Swagger 是用于生成 RESTful Web 服务的可视化表示的工具，规范和完整框架实现。它使文档能够以与服务器相同的速度更新。当通过 Swagger 正确定义时，消费者可以使用最少量的实现逻辑来理解远程服务并与其进行交互。因此， Swagger 消除了调用服务时的猜测。

## 13、什么是 Spring Profiles？

Spring Profiles 允许用户根据配置文件（dev，test，prod 等）来注册 bean。因此，当应用程序在开发中运行时，只有某些 bean 可以加载，而在 PRODUCTION 中，某些其他 bean 可以加载。假设我们的要求是 Swagger 文档仅适用于 QA 环境，并且禁用所有其他文档。这可以使用配置文件来完成。Spring Boot 使得使用配置文件非常简单。

## 14、什么是 Spring Batch？

Spring Boot Batch 提供可重用的函数，这些函数在处理大量记录时非常重要， 包括日志/跟踪，事务管理，作业处理统计信息，作业重新启动，跳过和资源管理。它还提供了更先进的技术服务和功能，通过优化和分区技术，可以实现极高批量和高性能批处理作业。简单以及复杂的大批量批处理作业可以高度可扩展的方式利用框架处理重要大量的信息。

## 15、什么是 FreeMarker 模板？

FreeMarker 是一个基于 Java 的模板引擎，最初专注于使用 MVC 软件架构进行动态网页生成。使用 Freemarker 的主要优点是表示层和业务层的完全分离。程序员可以处理应用程序代码，而设计人员可以处理 html 页面设计。最后使用freemarker 可以将这些结合起来，给出最终的输出页面。

## 16、如何使用 Spring Boot 实现异常处理？

Spring 提供了一种使用 ControllerAdvice 处理异常的非常有用的方法。我们通过实现一个 ControlerAdvice 类，来处理控制器类抛出的所有异常。

## 17、您使用了哪些 starter maven 依赖项？

使用了下面的一些依赖项spring-boot-starter-activemq spring-boot-starter-security

这有助于增加更少的依赖关系，并减少版本的冲突。

## 18、什么是 CSRF 攻击？

CSRF 代表跨站请求伪造。这是一种攻击，迫使最终用户在当前通过身份验证的Web 应用程序上执行不需要的操作。CSRF 攻击专门针对状态改变请求，而不是数据窃取，因为攻击者无法查看对伪造请求的响应。

## 19、什么是 WebSockets？

WebSocket 是一种计算机通信协议，通过单个 TCP 连接提供全双工通信信道。1、WebSocket 是双向的 -使用 WebSocket 客户端或服务器可以发起消息发送。2、WebSocket 是全双工的 -客户端和服务器通信是相互独立的。

3、单个 TCP 连接 -初始连接使用 HTTP，然后将此连接升级到基于套接字的连接。然后这个单一连接用于所有未来的通信

4、Light -与 http 相比，WebSocket 消息数据交换要轻得多。

## 20、什么是 AOP？

在软件开发过程中，跨越应用程序多个点的功能称为交叉问题。这些交叉问题与应用程序的主要业务逻辑不同。因此，将这些横切关注与业务逻辑分开是面向方面编程（AOP）的地方。

## 21、什么是 Apache Kafka？

Apache Kafka 是一个分布式发布 - 订阅消息系统。它是一个可扩展的，容错的发布 - 订阅消息系统，它使我们能够构建分布式应用程序。这是一个 Apache 顶级项目。Kafka 适合离线和在线消息消费。

## 22、我们如何监视所有 Spring Boot 微服务？

Spring Boot 提供监视器端点以监控各个微服务的度量。这些端点对于获取有关应用程序的信息（如它们是否已启动）以及它们的组件（如数据库等）是否正常运行很有帮助。但是，使用监视器的一个主要缺点或困难是，我们必须单独打开应用程序的知识点以了解其状态或健康状况。想象一下涉及 50 个应用程序的微

服务，管理员将不得不击中所有 50 个应用程序的执行终端。为了帮助我们处理这种情况，我们将使用位于的开源项目。 它建立在 Spring Boot Actuator 之上，它提供了一个 Web UI，使我们能够可视化多个应用程序的度量。

**八、spring cloud**

### Spring Cloud 有以下特点:

约定优于配置；

适用于各种环境。开发、部署 PC Server 或各种云环境（例如阿里云、AWS 等）均可；

隐藏了组件的复杂性，并提供声明式、无 xml 的配置方式；

开箱即用，快速启动；

轻量级的组件。Spring Cloud 整合的组件大多比较轻量。例如 Eureka、Zuul 等，都是各自领域轻量级的实现；

组件丰富，功能齐全。Spring Cloud 为微服务架构提供了非常完整的支持。例如、配置管理、服务发现、断路器、微服务网关等；

选型中立、丰富。例如，Spring Cloud 支持使用 Eureka、Zookeeper 或Consul 实现服务发现；

灵活。Spring Cloud 的组成部分是解耦的，开发人员可以按需灵活挑选技术选型。

### 1、什么是 Spring Cloud？

Spring cloud 流应用程序启动器是基于 Spring Boot 的 Spring 集成应用程序，提供与外部系统的集成。Spring cloud Task，一个生命周期短暂的微服务框架，用于快速构建执行有限数据处理的应用程序。

### 2、使用 Spring Cloud 有什么优势？

使用 Spring Boot 开发分布式微服务时，我们面临以下问题

1. 与分布式系统相关的复杂性-这种开销包括网络问题，延迟开销，带宽问题，安全问题。
2. 服务发现-服务发现工具管理群集中的流程和服务如何查找和互相交谈。它涉及一个服务目录，在该目录中注册服务，然后能够查找并连接到该目录中的服务。
3. 冗余-分布式系统中的冗余问题。
4. 负载平衡 --负载平衡改善跨多个计算资源的工作负荷，诸如计算机，计算机集群，网络链路，中央处理单元，或磁盘驱动器的分布。
5. 性能-问题 由于各种运营开销导致的性能问题。
6. 部署复杂性-Devops 技能的要求。

### 3、服务注册和发现是什么意思？Spring Cloud 如何实现？

当我们开始一个项目时，我们通常在属性文件中进行所有的配置。随着越来越多的服务开发和部署，添加和修改这些属性变得更加复杂。有些服务可能会下降， 而某些位置可能会发生变化。手动更改属性可能会产生问题。 Eureka 服务注册和发现可以在这种情况下提供帮助。由于所有服务都在 Eureka 服务器上注册并通过调用 Eureka 服务器完成查找，因此无需处理服务地点的任何更改和处理。

### 4、Spring Cloud 和 dubbo 区别?

1. 服务调用方式 dubbo 是 RPC springcloud Rest Api
2. 注册中心,dubbo 是zookeeper springcloud 是eureka，也可以是 zookeeper
3. 服务网关,dubbo 本身没有实现，只能通过其他第三方技术整合， springcloud 有 Zuul 路由网关，作为路由服务器，进行消费者的请求分

发,springcloud 支持断路器，与 git 完美集成配置文件支持版本控制，事物总线实现配置文件的更新与服务自动装配等等一系列的微服务架构要素。

### 5、SpringBoot 和SpringCloud 的区别？

SpringBoot 专注于快速方便的开发单个个体微服务。

SpringCloud 是关注全局的微服务协调整理治理框架，它将 SpringBoot 开发的一个个单体微服务整合并管理起来，

为各个微服务之间提供，配置管理、服务发现、断路器、路由、微代理、事件总线、全局锁、决策竞选、分布式会话等等集成服务

SpringBoot 可以离开 SpringCloud 独立使用开发项目， 但是 SpringCloud 离不开 SpringBoot ，属于依赖的关系.

SpringBoot 专注于快速、方便的开发单个微服务个体，SpringCloud 关注全局的服务治理框架。

### 6、负载平衡的意义什么？

在计算中，负载平衡可以改善跨计算机，计算机集群，网络链接，中央处理单元或磁盘驱动器等多种计算资源的工作负载分布。负载平衡旨在优化资源使用，最大化吞吐量，最小化响应时间并避免任何单一资源的过载。使用多个组件进行负载平衡而不是单个组件可能会通过冗余来提高可靠性和可用性。负载平衡通常涉及专用软件或硬件，例如多层交换机或域名系统服务器进程。

### 7、什么是 Hystrix？它如何实现容错？

Hystrix 是一个延迟和容错库，旨在隔离远程系统，服务和第三方库的访问点， 当出现故障是不可避免的故障时，停止级联故障并在复杂的分布式系统中实现弹性。

通常对于使用微服务架构开发的系统，涉及到许多微服务。这些微服务彼此协作。思考以下微服务

假设如果上图中的微服务 9 失败了，那么使用传统方法我们将传播一个异常。但这仍然会导致整个系统崩溃。

随着微服务数量的增加，这个问题变得更加复杂。微服务的数量可以高达 1000. 这是 hystrix 出现的地方 我们将使用 Hystrix 在这种情况下的 Fallback 方法功能。我们有两个服务 employee-consumer 使用由 employee-consumer 公开的服务。

简化图如下所示

现在假设由于某种原因，employee-producer 公开的服务会抛出异常。我们在这种情况下使用 Hystrix 定义了一个回退方法。这种后备方法应该具有与公开服务相同的返回类型。如果暴露服务中出现异常，则回退方法将返回一些值。

### 8、什么是 Hystrix 断路器？我们需要它吗？

由于某些原因，employee-consumer 公开服务会引发异常。在这种情况下使用Hystrix 我们定义了一个回退方法。如果在公开服务中发生异常，则回退方法返回一些默认值。

如果 firstPage method() 中的异常继续发生，则 Hystrix 电路将中断，并且员工使用者将一起跳过 firtsPage 方法，并直接调用回退方法。 断路器的目的是给第一页方法或第一页方法可能调用的其他方法留出时间，并导致异常恢复。可能发生的情况是，在负载较小的情况下，导致异常的问题有更好的恢复机会 。

### 9、什么是 Netflix Feign？它的优点是什么？

Feign 是受到 Retrofit，JAXRS-2.0 和 WebSocket 启发的 java 客户端联编程序。

Feign 的第一个目标是将约束分母的复杂性统一到 http apis，而不考虑其稳定性。

在 employee-consumer 的例子中，我们使用了 employee-producer 使用 REST 模板公开的 REST 服务。

但是我们必须编写大量代码才能执行以下步骤

1. 使用功能区进行负载平衡。
2. 获取服务实例，然后获取基本 URL。
3. 利用 REST 模板来使用服务。 前面的代码如下@Controllerpublic class ConsumerControllerClient {

@Autowired

private LoadBalancerClient loadBalancer;

public void getEmployee() throws RestClientException, IOException {

ServiceInstance serviceInstance=loadBalancer.choose("employee-producer");

System.out.println(serviceInstance.getUri());

String baseUrl=serviceInstance.getUri().toString(); baseUrl=baseUrl+"/employee";

RestTemplate restTemplate = new RestTemplate();

ResponseEntity<String> response=null;

try{

response=restTemplate.exchange(baseUrl, HttpMethod.GET, getHeaders(),String.class);

}

catch (Exception ex)

{

System.out.println(ex);

}

System.out.println(response.getBody());

}复制代码

之前的代码，有像 NullPointer 这样的例外的机会，并不是最优的。我们将看到如何使用 Netflix Feign 使呼叫变得更加轻松和清洁。如果 Netflix Ribbon 依赖关系也在类路径中，那么 Feign 默认也会负责负载平衡。

### 10、什么是 Spring Cloud Bus？我们需要它吗？

考虑以下情况：我们有多个应用程序使用 Spring Cloud Config 读取属性，而Spring Cloud Config 从 GIT 读取这些属性。

下面的例子中多个员工生产者模块从 Employee Config Module 获取 Eureka 注册的财产。

如果假设 GIT 中的 Eureka 注册属性更改为指向另一台 Eureka 服务器，会发生什么情况。在这种情况下，我们将不得不重新启动服务以获取更新的属性。 还有另一种使用执行器端点/刷新的方式。但是我们将不得不为每个模块单独调用这个 url。例如，如果 Employee Producer1 部署在端口 8080 上，则调用http：// localhost：8080 / refresh。同样对于 Employee Producer2 http：

//localhost：8081 / refresh 等等。这又很麻烦。这就是 Spring Cloud Bus 发挥作用的地方。

Spring Cloud Bus 提供了跨多个实例刷新配置的功能。因此，在上面的示例中， 如果我们刷新 Employee Producer1，则会自动刷新所有其他必需的模块。如果我们有多个微服务启动并运行，这特别有用。这是通过将所有微服务连接到单个消息代理来实现的。无论何时刷新实例，此事件都会订阅到侦听此代理的所有微服务，并且它们也会刷新。可以通过使用端点/总线/刷新来实现对任何单个实例的刷新。

### 11.springcloud 断路器的作用

当一个服务调用另一个服务由于网络原因或自身原因出现问题，调用者就会等待被调用者的响应 当更多的服务请求到这些资源导致更多的请求等待，发生连锁效应（雪崩效应）

断路器有完全打开状态:一段时间内 达到一定的次数无法调用 并且多次监测没有恢复的迹象 断路器完全打开 那么下次请求就不会请求到该服务

半开:短时间内 有恢复迹象 断路器会将部分请求发给该服务，正常调用时 断路器关闭

关闭：当服务一直处于正常状态 能正常调用

### 12、什么是 SpringCloudConfig?

在分布式系统中，由于服务数量巨多，为了方便服务配置文件统一管理，实时更新，所以需要分布式配置中心组件。在 Spring Cloud 中，有分布式配置中心组件 spring cloud config ，它支持配置服务放在配置服务的内存中（即本地）， 也支持放在远程 Git 仓库中。在 spring cloud config 组件中，分两个角色， 一是 config server，二是 config client。

使用：

1. 添加 pom 依赖
2. 配置文件添加相关配置
3. 启动类添加注解@EnableConfigServer

### 13、Spring Cloud Gateway?

Spring Cloud Gateway 是 Spring Cloud 官方推出的第二代网关框架，取代 Zuul 网关。网关作为流量的，在微服务系统中有着非常作用，网关常见的功能有路由转发、权限校验、限流控制等作用。

使用了一个 RouteLocatorBuilder 的 bean 去创建路由，除了创建路由RouteLocatorBuilder 可以让你添加各种 predicates 和 filters，predicates 断言的意思，顾名思义就是根据具体的请求的规则，由具体的 route 去处理， filters 是各种过滤器，用来对请求做各种判断和修改。

# 九、微服务 面试题

### 1、您对微服务有何了解？

微服务，又称微服务 架 构，是一种架构风格，它将应用程序构建为以业务领域为模型的小型自治服务集合 。

通俗地说，你必须看到蜜蜂如何通过对齐六角形蜡细胞来构建它们的蜂窝状物。他们最初从使用各种材料的小部分开始，并继续从中构建一个大型蜂箱。这些细

胞形成图案，产生坚固的结构，将蜂窝的特定部分固定在一起。这里，每个细胞独立于另一个细胞，但它也与其他细胞相关。这意味着对一个细胞的损害不会损害其他细胞，因此，蜜蜂可以在不影响完整蜂箱的情况下重建这些细胞。

图 1：微服务的蜂窝表示 – 微服务访谈问题

请参考上图。这里，每个六边形形状代表单独的服务组件。与蜜蜂的工作类似， 每个敏捷团队都使用可用的框架和所选的技术堆栈构建单独的服务组件。就像在蜂箱中一样，每个服务组件形成一个强大的微服务架构，以提供更好的可扩展性。此外，敏捷团队可以单独处理每个服务组件的问题，而对整个应用程序没有影响或影响最小。

### 2、微服务架构有哪些优势？

图 2：微服务的 优点 – 微服务访谈问题

* 独立开发 – 所有微服务都可以根据各自的功能轻松开发
* 独立部署 – 基于其服务，可以在任何应用程序中单独部署它们
* 故障隔离 – 即使应用程序的一项服务不起作用，系统仍可继续运行
* 混合技术堆栈 – 可以使用不同的语言和技术来构建同一应用程序的不同服务
* 粒度缩放 – 单个组件可根据需要进行缩放，无需将所有组件缩放在一起

### 3、微服务有哪些特点？

图 3：微服务的 特点 – 微服务访谈问题

* 解耦 – 系统内的服务很大程度上是分离的。因此，整个应用程序可以轻松构建，更改和扩展
* 组件化 – 微服务被视为可以轻松更换和升级的独立组件
* 业务能力 – 微服务非常简单，专注于单一功能
* 自治 – 开发人员和团队可以彼此独立工作，从而提高速度
* 持续交付 – 通过软件创建，测试和批准的系统自动化，允许频繁发布软件
* 责任 – 微服务不关注应用程序作为项目。相反，他们将应用程序视为他们负责的产品
* 分散治理 – 重点是使用正确的工具来做正确的工作。这意味着没有标准化模式或任何技术模式。开发人员可以自由选择最有用的工具来解决他们的问题
* 敏捷 – 微服务支持敏捷开发。任何新功能都可以快速开发并再次丢弃

### 4、设计微服务的最佳实践是什么？

以下是设计微服务的最佳实践：

图 4：设计微服务的最佳实践 – 微服务访谈问题

### 5、微服务架构如何运作？

微服务架构具有以下组件：

图 5：微服务 架构 – 微服务面试问题

* 客户端 – 来自不同设备的不同用户发送请求。
* 身份提供商 – 验证用户或客户身份并颁发安全令牌。
* API 网关 – 处理客户端请求。
* 静态内容 – 容纳系统的所有内容。
* 管理 – 在节点上平衡服务并识别故障。
* 服务发现 – 查找微服务之间通信路径的指南。
* 内容交付网络 – 代理服务器及其数据中心的分布式网络。
* 远程服务 – 启用驻留在 IT 设备网络上的远程访问信息。

### 6、微服务架构的优缺点是什么？

**7、单片，SOA 和微服务架构有什么区别？**

图 6： 单片 SOA 和微服务之间的比较 – 微服务访谈问题

* 单片架构类似于大容器，其中应用程序的所有软件组件组装在一起并紧密封装。
* 一个面向服务的架构是一种相互通信服务的集合。通信可以涉及简单的数据传递，也可以涉及两个或多个协调某些活动的服务。
* 微服务架构是一种架构风格，它将应用程序构建为以业务域为模型的小型自治服务集合。

### 8、在使用微服务架构时，您面临哪些挑战？

开发一些较小的微服务听起来很容易，但开发它们时经常遇到的挑战如下。

* 自动化组件：难以自动化，因为有许多较小的组件。因此，对于每个组件， 我们必须遵循 Build，Deploy 和 Monitor 的各个阶段。
* 易感性：将大量组件维护在一起变得难以部署，维护，监控和识别问题。它需要在所有组件周围具有很好的感知能力。
* 配置管理：有时在各种环境中维护组件的配置变得困难。
* 调试：很难找到错误的每一项服务。维护集中式日志记录和仪表板以调试问题至关重要。

### 9、SOA 和微服务架构之间的主要区别是什么？

SOA 和微服务之间的主要区别如下：

### 10、微服务有什么特点？

您可以列出微服务的特征，如下所示：

图 7：微服务的特征 – 微服务访谈问题

### 11、什么是领域驱动设计？

图 8： DDD 原理 – 微服务面试问题

### 12、为什么需要域驱动设计（DDD）？

图 9：我们需要 DDD 的因素 – 微服务面试问题

### 13、什么是无所不在的语言？

如果您必须定义泛在语言（UL），那么它是特定域的开发人员和用户使用的通用语言，通过该语言可以轻松解释域。

无处不在的语言必须非常清晰，以便它将所有团队成员放在同一页面上，并以机器可以理解的方式进行翻译。

### 14、什么是凝聚力？

模块内部元素所属的程度被认为是凝聚力。

### 15、什么是耦合？

组件之间依赖关系强度的度量被认为是耦合。一个好的设计总是被认为具有高内聚力和低耦合性。

### 16、什么是 REST / RESTful 以及它的用途是什么？

Representational State Transfer（REST）/ RESTful Web 服务是一种帮助计算机系统通过 Internet 进行通信的架构风格。这使得微服务更容易理解和实现。

微服务可以使用或不使用 RESTful API 实现，但使用 RESTful API 构建松散耦合的微服务总是更容易。

### 17、你对 Spring Boot 有什么了解？

事实上，随着新功能的增加，弹簧变得越来越复杂。如果必须启动新的 spring 项目，则必须添加构建路径或添加 maven 依赖项，配置应用程序服务器，添加spring 配置。所以一切都必须从头开始。

Spring Boot 是解决这个问题的方法。使用 spring boot 可以避免所有样板代码和配置。因此，基本上认为自己就好像你正在烘烤蛋糕一样，春天就像制作蛋糕所需的成分一样，弹簧靴就是你手中的完整蛋糕。

图 10： Spring Boot 的因素 – 微服务面试问题

### 18、什么是 Spring 引导的执行器？

Spring Boot 执行程序提供了 restful Web 服务，以访问生产环境中运行应用程序的当前状态。在执行器的帮助下，您可以检查各种指标并监控您的应用程序。

### 19、什么是 Spring Cloud？

根据 Spring Cloud 的官方网站，Spring Cloud 为开发人员提供了快速构建分布式系统中一些常见模式的工具（例如配置管理，服务发现，断路器，智能路由， 领导选举，分布式会话，集群状态）。

### 20、Spring Cloud 解决了哪些问题？

在使用 Spring Boot 开发分布式微服务时，我们面临的问题很少由 Spring Cloud 解决。

* 与分布式系统相关的复杂性 – 包括网络问题，延迟开销，带宽问题，安全问题。
* 处理服务发现的能力 – 服务发现允许集群中的进程和服务找到彼此并进行通信。
* 解决冗余问题 – 冗余问题经常发生在分布式系统中。
* 负载平衡 – 改进跨多个计算资源（例如计算机集群，网络链接，中央处理单元）的工作负载分布。
* 减少性能问题 – 减少因各种操作开销导致的性能问题。

### 21、在 Spring MVC 应用程序中使用 WebMvcTest 注释有什么用处？

在测试目标只关注 Spring MVC 组件的情况下，WebMvcTest 注释用于单元测试Spring MVC 应用程序。在上面显示的快照中，我们只想启动 ToTestController。执行此单元测试时，不会启动所有其他控制器和映射。

### 22、你能否给出关于休息和微服务的要点？

虽然您可以通过多种方式实现微服务，但 REST over HTTP 是实现微服务的一种方式。REST 还可用于其他应用程序，如 Web 应用程序，API 设计和 MVC 应用程序，以提供业务数据。

微服务是一种体系结构，其中系统的所有组件都被放入单独的组件中，这些组件可以单独构建，部署和扩展。微服务的某些原则和最佳实践有助于构建弹性应用程序。

简而言之，您可以说 REST 是构建微服务的媒介。

### 23、什么是不同类型的微服务测试？

在使用微服务时，由于有多个微服务协同工作，测试变得非常复杂。因此，测试分为不同的级别。

* 在底层，我们有面向技术的测试，如单元测试和性能测试。这些是完全自动化的。
* 在中间层面，我们进行了诸如压力测试和可用性测试之类的探索性测试。
* 在顶层， 我们的 验收测试数量很少。这些验收测试有助于利益相关者理解和验证软件功能。

### 24、您对 Distributed Transaction 有何了解？

分布式事务是指单个事件导致两个或多个不能以原子方式提交的单独数据源的突变的任何情况。在微服务的世界中，它变得更加复杂，因为每个服务都是一个工作单元，并且大多数时候多个服务必须协同工作才能使业务成功。

### 25、什么是 Idempotence 以及它在哪里使用？

幂等性是能够以这样的方式做两次事情的特性，即最终结果将保持不变，即好像它只做了一次。

用法：在远程服务或数据源中使用 Idempotence，这样当它多次接收指令时，它只处理指令一次。

### 26、什么是有界上下文？

有界上下文是域驱动设计的核心模式。DDD 战略设计部门的重点是处理大型模型和团队。DDD 通过将大型模型划分为不同的有界上下文并明确其相互关系来处理大型模型。

### 27、什么是双因素身份验证？

双因素身份验证为帐户登录过程启用第二级身份验证。

图 11： 双因素认证的表示 – 微服务访谈问题

因此，假设用户必须只输入用户名和密码，那么这被认为是单因素身份验证。

### 28、双因素身份验证的凭据类型有哪些？

这三种凭证是：

图 12： 双因素认证的证书类型 – 微服务面试问题

### 29、什么是客户证书？

客户端系统用于向远程服务器发出经过身份验证的请求的一种数字证书称为客户端证书。客户端证书在许多相互认证设计中起着非常重要的作用，为请求者的身份提供了强有力的保证。

### 30、PACT 在微服务架构中的用途是什么？

PACT 是一个开源工具，允许测试服务提供者和消费者之间的交互，与合同隔离， 从而提高微服务集成的可靠性。

微服务中的用法

* 用于在微服务中实现消费者驱动的合同。
* 测试微服务的消费者和提供者之间的消费者驱动的合同。查看即将到来的批次

### 31、什么是 OAuth？

OAuth 代表开放授权协议。这允许通过在 HTTP 服务上启用客户端应用程序（例如第三方提供商 Facebook，GitHub 等）来访问资源所有者的资源。因此，您可以在不使用其凭据的情况下与另一个站点共享存储在一个站点上的资源。

### 32、康威定律是什么？

“任 何 设 计 系 统 的 组 织 （ 广 泛 定 义 ） 都 将 产 生 一 种 设计 ， 其 结 构 是 组 织 通 信 结 构的 副 本 。” – Mel Conway

图 13： Conway 定律的表示 – 微服务访谈问题

该法律基本上试图传达这样一个事实：为了使软件模块起作用，整个团队应该进行良好的沟通。因此，系统的结构反映了产生它的组织的社会边界。

### 33、合同测试你懂什么？

根据 Martin Flower 的说法，合同测试是在外部服务边界进行的测试，用于验证其是否符合消费服务预期的合同。

此外，合同测试不会深入测试服务的行为。更确切地说，它测试该服务调用的输入＆输出包含所需的属性和所述响应延迟，吞吐量是允许的限度内。

### 34、什么是端到端微服务测试？

端到端测试验证了工作流中的每个流程都正常运行。这可确保系统作为一个整体协同工作并满足所有要求。

通俗地说，你可以说端到端测试是一种测试，在特定时期后测试所有东西。

图 14：测试层次 – 微服务面试问题

### 35、Container 在微服务中的用途是什么？

容器是管理基于微服务的应用程序以便单独开发和部署它们的好方法。您可以将微服务封装在容器映像及其依赖项中，然后可以使用它来滚动按需实例的微服务，而无需任何额外的工作。

图 15： 容器的表示及其在微服务中的使用方式 – 微服务访谈问题

### 36、什么是微服务架构中的 DRY？

DRY 代表不要重复自己。它基本上促进了重用代码的概念。这导致开发和共享库， 这反过来导致紧密耦合。

### 37、什么是消费者驱动的合同（CDC）？

这基本上是用于开发微服务的模式，以便它们可以被外部系统使用。当我们处理微服务时，有一个特定的提供者构建它，并且有一个或多个使用微服务的消费者。通常，提供程序在 XML 文档中指定接口。但在消费者驱动的合同中，每个服务消费者都传达了提供商期望的接口。

### 38、Web，RESTful API 在微服务中的作用是什么？

微服务架构基于一个概念，其中所有服务应该能够彼此交互以构建业务功能。因此，要实现这一点，每个微服务必须具有接口。这使得 Web API 成为微服务的一个非常重要的推动者。RESTful API 基于 Web 的开放网络原则，为构建微服务架构的各个组件之间的接口提供了最合理的模型。

### 39、您对微服务架构中的语义监控有何了解？

语义监控，也称为 综合监控， 将自动化测试与监控应用程序相结合，以检测业务失败因素。

### 40、我们如何进行跨功能测试？

跨功能测试是对非功能性需求的验证，即那些无法像普通功能那样实现的需求。

### 41、我们如何在测试中消除非决定论？

非确定性测试（NDT）基本上是不可靠的测试。所以，有时可能会发生它们通过， 显然有时它们也可能会失败。当它们失败时，它们会重新运行通过。

从测试中删除非确定性的一些方法如下： 1、 隔离

2、 异步

3、 远程服务

4、 隔离

5、 时间

6、 资源泄漏

### 42、Mock 或 Stub 有什么区别？

存根

* 一个有助于运行测试的虚拟对象。
* 在某些可以硬编码的条件下提供固定行为。
* 永远不会测试存根的任何其他行为。

例如，对于空堆栈，您可以创建一个只为 empty（）方法 返回 true 的存根。因此，这并不关心堆栈中是否存在元素。

嘲笑

* 一个虚拟对象，其中最初设置了某些属性。
* 此对象的行为取决于 set 属性。
* 也可以测试对象的行为。

例如，对于 Customer 对象，您可以通过设置名称和年龄来模拟它。您可以将 age 设置为 12，然后测试 isAdult（）方法，该方法将在年龄大于 18 时返回 true。因此，您的 Mock Customer 对象适用于指定的条件。

### 43、您对 Mike Cohn 的测试金字塔了解多少？

Mike Cohn 提供了一个名为 Test Pyramid 的模型。这描述了软件开发所需的自动化测试类型。

图 16： Mike Cohn 的测试金字塔 – 微服务面试问题

根据金字塔，第一层的测试数量应该最高。在服务层，测试次数应小于单元测试级别，但应大于端到端级别。

### 44、Docker 的目的是什么？

Docker 提供了一个可用于托管任何应用程序的容器环境。在此，软件应用程序和支持它的依赖项紧密打包在一起。

因此，这个打包的产品被称为 Container，因为它是由 Docker 完成的，所以它被称为 Docker 容器！

### 45、什么是金丝雀释放？

Canary Releasing 是一种降低在生产中引入新软件版本的风险的技术。这是通过将变更缓慢地推广到一小部分用户，然后将其发布到整个基础架构，即将其提供给每个人来完成的。

### 46、什么是持续集成（CI）？

持续集成（CI）是每次团队成员提交版本控制更改时自动构建和测试代码的过程。这鼓励开发人员通过在每个小任务完成后将更改合并到共享版本控制存储库来 共享代码和单元测试。

### 47、什么是持续监测？

持续监控深入监控覆盖范围，从浏览器内前端性能指标，到应用程序性能，再到主机虚拟化基础架构指标。

### 48、架构师在微服务架构中的角色是什么？

微服务架构中的架构师扮演以下角色：

* 决定整个软件系统的布局。
* 帮助确定组件的分区。因此，他们确保组件相互粘合，但不紧密耦合。
* 与开发人员共同编写代码，了解日常生活中面临的挑战。
* 为开发微服务的团队提供某些工具和技术的建议。
* 提供技术治理，以便技术开发团队遵循微服务原则。

### 49、我们可以用微服务创建状态机吗？

我们知道拥有自己的数据库的每个微服务都是一个可独立部署的程序单元，这反过来又让我们可以创建一个状态机。因此，我们可以为特定的微服务指定不同的状态和事件。

例如，我们可以定义 Order 微服务。订单可以具有不同的状态。Order 状态的转换可以是 Order 微服务中的独立事件。

### 50、什么是微服务中的反应性扩展？

Reactive Extensions 也称为 Rx。这是一种设计方法，我们通过调用多个服务来收集结果，然后编译组合响应。这些调用可以是同步或异步，阻塞或非阻塞。Rx 是分布式系统中非常流行的工具，与传统流程相反。

# 十、kafka 面试题

### 1、如何获取 topic 主题的列表

bin/kafka-topics.sh --list --zookeeper localhost:2181

### 2、生产者和消费者的命令行是什么？

#### 生产者在主题上发布消息：

bin/kafka-console-producer.sh --broker-list 192.168.43.49:9092

--topicHello-Kafka

注意这里的 IP 是 server.properties 中的 listeners 的配置。接下来每个新行就是输入一条新消息。

#### 消费者接受消息：

bin/kafka-console-consumer.sh --zookeeper localhost:2181

--topicHello-Kafka --from-beginning

### 3、consumer 是推还是拉？

Kafka 最初考虑的问题是，customer 应该从 brokes 拉取消息还是 brokers 将消息推送到 consumer，也就是 pull 还 push。在这方面，Kafka 遵循了一种大部分消息系统共同的传统的设计：producer 将消息推送到 broker，consumer 从 broker 拉取消息。

一些消息系统比如 Scribe 和 Apache Flume 采用了 push 模式，将消息推送到下游的 consumer。这样做有好处也有坏处：由 broker 决定消息推送的速率， 对于不同消费速率的 consumer 就不太好处理了。消息系统都致力于让consumer 以最大的速率最快速的消费消息，但不幸的是，push 模式下，当broker 推送的速率远大于 consumer 消费的速率时，consumer 恐怕就要崩溃了。最终 Kafka 还是选取了传统的 pull 模式。

Pull 模式的另外一个好处是 consumer 可以自主决定是否批量的从 broker 拉取数据 。Push 模式必须在不知道下游 consumer 消费能力和消费策略的情况下决定是立即推送每条消息还是缓存之后批量推送。如果为了避免 consumer 崩溃而采用较低的推送速率，将可能导致一次只推送较少的消息而造成浪费。Pull 模式下，consumer 就可以根据自己的消费能力去决定这些策略。

Pull 有个缺点是，如果 broker 没有可供消费的消息，将导致 consumer 不断在循环中轮询，直到新消息到 t 达。为了避免这点，Kafka 有个参数可以让consumer 阻塞知道新消息到达(当然也可以阻塞知道消息的数量达到某个特定的量这样就可以批量发送)。

### 4、讲讲 kafka 维护消费状态跟踪的方法

大部分消息系统在 broker 端的维护消息被消费的记录：一个消息被分发到consumer 后 broker 就马上进行标记或者等待 customer 的通知后进行标记。这样也可以在消息在消费后立马就删除以减少空间占用。

但是这样会不会有什么问题呢？如果一条消息发送出去之后就立即被标记为消费过的，旦 consumer 处理消息时失败了（比如程序崩溃）消息就丢失了。为了解决这个问题，很多消息系统提供了另外一个个功能：当消息被发送出去之后仅仅被标记为已发送状态，当接到 consumer 已经消费成功的通知后才标记为已被消费的状态。这虽然解决了消息丢失的问题，但产生了新问题，首先如果consumer 处理消息成功了但是向 broker 发送响应时失败了，这条消息将被消费两次。第二个问题时，broker 必须维护每条消息的状态，并且每次都要先锁住消息然后更改状态然后释放锁。这样麻烦又来了，且不说要维护大量的状态数据，比如如果消息发送出去但没有收到消费成功的通知，这条消息将一直处于被锁定的状态，Kafka 采用了不同的策略。Topic 被分成了若干分区，每个分区在同一时间只被一个 consumer 消费。这意味着每个分区被消费的消息在日志中的位置仅仅是一个简单的整数：offset。这样就很容易标记每个分区消费状态就很容易了，仅仅需要一个整数而已。这样消费状态的跟踪就很简单了。

这带来了另外一个好处：consumer 可以把 offset 调成一个较老的值，去重新消费老的消息。这对传统的消息系统来说看起来有些不可思议，但确实是非常有用的，谁规定了一条消息只能被消费一次呢？

### 5、讲一下主从同步

Kafka 允许 topic 的分区拥有若干副本，这个数量是可以配置的，你可以为每个topci 配置副本的数量。Kafka 会自动在每个个副本上备份数据，所以当一个节点 down 掉时数据依然是可用的。

Kafka 的副本功能不是必须的，你可以配置只有一个副本，这样其实就相当于只有一份数据。

### 6、为什么需要消息系统，mysql 不能满足需求吗？

#### 解耦：

允许你独立的扩展或修改两边的处理过程，只要确保它们遵守同样的接口约束。

#### 冗余：

消息队列把数据进行持久化直到它们已经被完全处理，通过这一方式规避了数据丢失风险。许多消息队列所采用的”插入-获取-删除”范式中，在把一个消息从队列中删除之前，需要你的处理系统明确的指出该消息已经被处理完毕，从而确保你的数据被安全的保存直到你使用完毕。

#### 扩展性：

因为消息队列解耦了你的处理过程，所以增大消息入队和处理的频率是很容易的，只要另外增加处理过程即可。

#### 灵活性 & 峰值处理能力：

在访问量剧增的情况下，应用仍然需要继续发挥作用，但是这样的突发流量并不常见。如果为以能处理这类峰值访问为标准来投入资源随时待命无疑是巨大的浪费。使用消息队列能够使关键组件顶住突发的访问压力，而不会因为突发的超负荷的请求而完全崩溃。

#### 可恢复性：

系统的一部分组件失效时，不会影响到整个系统。消息队列降低了进程间的耦合度，所以即使一个处理消息的进程挂掉，加入队列中的消息仍然可以在系统恢复后被处理。

#### 顺序保证：

在大多使用场景下，数据处理的顺序都很重要。大部分消息队列本来就是排序的， 并且能保证数据会按照特定的顺序来处理。（Kafka 保证一个 Partition 内的消息的有序性）

#### 缓冲：

有助于控制和优化数据流经过系统的速度，解决生产消息和消费消息的处理速度不一致的情况。

#### 异步通信：

很多时候，用户不想也不需要立即处理消息。消息队列提供了异步处理机制，允许用户把一个消息放入队列，但并不立即处理它。想向队列中放入多少消息就放多少，然后在需要的时候再去处理它们。

### 7、Zookeeper 对于 Kafka 的作用是什么？

Zookeeper 是一个开放源码的、高性能的协调服务，它用于 Kafka 的分布式应用。

Zookeeper 主要用于在集群中不同节点之间进行通信

在 Kafka 中，它被用于提交偏移量，因此如果节点在任何情况下都失败了，它都可以从之前提交的偏移量中获取除此之外，它还执行其他活动，如: leader 检测、分布式同步、配置管理、识别新节点何时离开或连接、集群、节点实时状态等等。

8、数据传输的事务定义有哪三种？ 和 MQTT 的事务定义一样都是 3 种。

1. 最多一次: 消息不会被重复发送，最多被传输一次，但也有可能一次不传输
2. 最少一次: 消息不会被漏发送，最少被传输一次，但也有可能被重复传输.
3. 精确的一次（Exactly once）: 不会漏传输也不会重复传输,每个消息都传输被一次而且仅仅被传输一次，这是大家所期望的

### 9、Kafka 判断一个节点是否还活着有那两个条件？

1. 节点必须可以维护和 ZooKeeper 的连接，Zookeeper 通过心跳机制检查每个节点的连接
2. 如果节点是个 follower,他必须能及时的同步 leader 的写操作，延时不能太久

### 10、Kafka 与传统 MQ 消息系统之间有三个关键区别

(1).Kafka 持久化日志，这些日志可以被重复读取和无限期保留

(2).Kafka 是一个分布式系统：它以集群的方式运行，可以灵活伸缩，在内部通过复制数据提升容错能力和高可用性

(3).Kafka 支持实时的流式处理

### 11、讲一讲 kafka 的 ack 的三种机制

request.required.acks 有三个值 0 1 -1(all)

0:生产者不会等待 broker 的 ack，这个延迟最低但是存储的保证最弱当server 挂掉的时候就会丢数据。

1：服务端会等待 ack 值 leader 副本确认接收到消息后发送 ack 但是如果leader 挂掉后他不确保是否复制完成新 leader 也会导致数据丢失。

-1(all)：服务端会等所有的 follower 的副本受到数据后才会受到 leader 发出的 ack，这样数据不会丢失

### 12、消费者如何不自动提交偏移量，由应用提交？

将 auto.commit.offset 设为 false，然后在处理一批消息后 commitSync() 或者异步提交 commitAsync()

即：

ConsumerRecords<> records = consumer.poll();for (ConsumerRecord<> record : records){

。。。

tyr{

consumer.commitSync()

}

。。。

}复制代码

### 13、消费者故障，出现活锁问题如何解决？

出现“活锁”的情况，是它持续的发送心跳，但是没有处理。为了预防消费者在这种情况下一直持有分区，我们使用 max.poll.interval.ms 活跃检测机制。在此基础上，如果你调用的 poll 的频率大于最大间隔，则客户端将主动地离开组， 以便其他消费者接管该分区。 发生这种情况时，你会看到 offset 提交失败（调用 commitSync（）引发的 CommitFailedException）。这是一种安全机制，保障只有活动成员能够提交 offset。所以要留在组中，你必须持续调用 poll。消费者提供两个配置设置来控制 poll 循环：

max.poll.interval.ms：增大 poll 的间隔，可以为消费者提供更多的时间去处理返回的消息（调用 poll(long)返回的消息，通常返回的消息都是一批）。缺点是此值越大将会延迟组重新平衡。

max.poll.records：此设置限制每次调用 poll 返回的消息数，这样可以更容易的预测每次 poll 间隔要处理的最大值。通过调整此值，可以减少 poll 间隔， 减少重新平衡分组的

对于消息处理时间不可预测地的情况，这些选项是不够的。 处理这种情况的推荐方法是将消息处理移到另一个线程中，让消费者继续调用 poll。 但是必须注意确保已提交的 offset 不超过实际位置。另外，你必须禁用自动提交，并只有在线程完成处理后才为记录手动提交偏移量（取决于你）。 还要注意，你需要pause 暂停分区，不会从 poll 接收到新消息，让线程处理完之前返回的消息（如果你的处理能力比拉取消息的慢，那创建新线程将导致你机器内存溢出）。

### 14、如何控制消费的位置

kafka 使用 seek(TopicPartition, long)指定新的消费位置。用于查找服务器保留的最早和最新的 offset 的特殊的方法也可用

（seekToBeginning(Collection) 和 seekToEnd(Collection)）

### 15、kafka 分布式（不是单机）的情况下，如何保证消息的顺序消费?

Kafka 分布式的单位是 partition，同一个 partition 用一个 write ahead log 组织，所以可以保证 FIFO 的顺序。不同 partition 之间不能保证顺序。但是绝大多数用户都可以通过 message key 来定义，因为同一个 key 的 message 可以保证只发送到同一个 partition。

Kafka 中发送 1 条消息的时候，可以指定(topic, partition, key) 3 个参数。partiton 和 key 是可选的。如果你指定了 partition，那就是所有消息发往同1 个 partition，就是有序的。并且在消费端，Kafka 保证，1 个 partition 只能被 1 个 consumer 消费。或者你指定 key（ 比如 order id），具有同 1 个key 的所有消息，会发往同 1 个 partition。

### 16、kafka 的高可用机制是什么？

这个问题比较系统，回答出 kafka 的系统特点，leader 和 follower 的关系， 消息读写的顺序即可。

### 17、kafka 如何减少数据丢失

Kafka 到底会不会丢数据(data loss)? 通常不会，但有些情况下的确有可能会发生。下面的参数配置及 Best practice 列表可以较好地保证数据的持久性(当然是 trade-off，牺牲了吞吐量)。



block.on.buffer.full = true





acks = all





retries = MAX\_VALUE





max.in.flight.requests.per.connection = 1





使用 KafkaProducer.send(record, callback)





callback 逻辑中显式关闭 producer：close(0)





unclean.leader.election.enable=false





replication.factor = 3





min.insync.replicas = 2





replication.factor > min.insync.replicas





enable.auto.commit=false





消息处理完成之后再提交位移



### 18、kafka 如何不消费重复数据？比如扣款，我们不能重复的扣。

其实还是得结合业务来思考，我这里给几个思路：

比如你拿个数据要写库，你先根据主键查一下，如果这数据都有了，你就别插入了，update 一下好吧。

比如你是写 Redis，那没问题了，反正每次都是 set，天然幂等性。

比如你不是上面两个场景，那做的稍微复杂一点，你需要让生产者发送每条数据的时候，里面加一个全局唯一的 id，类似订单 id 之类的东西，然后你这里消费到了之后，先根据这个 id 去比如 Redis 里查一下，之前消费过吗？如果没有消费过，你就处理，然后这个 id 写 Redis。如果消费过了，那你就别处理了， 保证别重复处理相同的消息即可。

比如基于数据库的唯一键来保证重复数据不会重复插入多条。因为有唯一键约束了，重复数据插入只会报错，不会导致数据库中出现脏数据。

## 十一、RabbitMQ 面试题

## 1、什么是 rabbitmq

采用 AMQP 高级消息队列协议的一种消息队列技术,最大的特点就是消费并不需要确保提供方存在,实现了服务之间的高度解耦

## 2、为什么要使用 rabbitmq

1. 在分布式系统下具备异步,削峰,负载均衡等一系列高级功能;
2. 拥有持久化的机制，进程消息，队列中的信息也可以保存下来。
3. 实现消费者和生产者之间的解耦。
4. 对于高并发场景下，利用消息队列可以使得同步访问变为串行访问达到一定量的限流，利于数据库的操作。
5. 可以使用消息队列达到异步下单的效果，排队中，后台进行逻辑下单。

## 3、使用 rabbitmq 的场景

1. 服务间异步通信
2. 顺序消费
3. 定时任务
4. 请求削峰

## 4、如何确保消息正确地发送至 RabbitMQ？ 如何确保消息接收方消费了消息？

### 发送方确认模式

将信道设置成 confirm 模式（发送方确认模式），则所有在信道上发布的消息都会被指派一个唯一的 ID。

一旦消息被投递到目的队列后，或者消息被写入磁盘后（可持久化的消息），信道会发送一个确认给生产者（包含消息唯一 ID）。

如果 RabbitMQ 发生内部错误从而导致消息丢失，会发送一条 nack

（notacknowledged，未确认）消息。

发送方确认模式是异步的，生产者应用程序在等待确认的同时，可以继续发送消息。当确认消息到达生产者应用程序，生产者应用程序的回调方法就会被触发来处理确认消息。

### 接收方确认机制

消费者接收每一条消息后都必须进行确认（消息接收和消息确认是两个不同操作）。只有消费者确认了消息，RabbitMQ 才能安全地把消息从队列中删除。这里并没有用到超时机制，RabbitMQ 仅通过 Consumer 的连接中断来确认是否需要重新发送消息。也就是说，只要连接不中断，RabbitMQ 给了 Consumer 足够长的时间来处理消息。保证数据的最终一致性；

下面罗列几种特殊情况

（1）如果消费者接收到消息，在确认之前断开了连接或取消订阅，RabbitMQ 会认为消息没有被分发，然后重新分发给下一个订阅的消费者。（可能存在消息重复消费的隐患，需要去重）

（1）2 如果消费者接收到消息却没有确认消息，连接也未断开，则 RabbitMQ 认为该消费者繁忙，将不会给该消费者分发更多的消息。

## 5.如何避免消息重复投递或重复消费？

在消息生产时，MQ 内部针对每条生产者发送的消息生成一个 inner-msg-id，作为去重的依据（消息投递失败并重传），避免重复的消息进入队列；在消息消费时，要求消息体中必须要有一个 bizId（对于同一业务全局唯一，如支付 ID、订单 ID、帖子 ID 等）作为去重的依据，避免同一条消息被重复消费。

## 6、消息基于什么传输？

由于 TCP 连接的创建和销毁开销较大，且并发数受系统资源限制，会造成性能瓶颈。RabbitMQ 使用信道的方式来传输数据。信道是建立在真实的 TCP 连接内的虚拟连接，且每条 TCP 连接上的信道数量没有限制。

## 7、消息如何分发？

若该队列至少有一个消费者订阅，消息将以循环（round-robin）的方式发送给消费者。每条消息只会分发给一个订阅的消费者（前提是消费者能够正常处理消息并进行确认）。通过路由可实现多消费的功能

## 8、消息怎么路由？

消息提供方->路由->一至多个队列消息发布到交换器时，消息将拥有一个路由键

（routing key），在消息创建时设定。通过队列路由键，可以把队列绑定到交换器上。消息到达交换器后，RabbitMQ 会将消息的路由键与队列的路由键进行匹配（针对不同的交换器有不同的路由规则）；

常用的交换器主要分为一下三种：

fanout：如果交换器收到消息，将会广播到所有绑定的队列上

direct：如果路由键完全匹配，消息就被投递到相应的队列

topic：可以使来自不同源头的消息能够到达同一个队列。 使用 topic 交换器时，可以使用通配符

## 9、如何确保消息不丢失？

消息持久化，当然前提是队列必须持久化

RabbitMQ 确保持久性消息能从服务器重启中恢复的方式是，将它们写入磁盘上的一个持久化日志文件，当发布一条持久性消息到持久交换器上时，Rabbit 会在消息提交到日志文件后才发送响应。一旦消费者从持久队列中消费了一条持久化消息，RabbitMQ 会在持久化日志中把这条消息标记为等待垃圾收集。如果持久化消息在被消费之前 RabbitMQ 重启，那么 Rabbit 会自动重建交换器和队列

（以及绑定），并重新发布持久化日志文件中的消息到合适的队列。

## 10、使用 RabbitMQ 有什么好处？

1. 服务间高度解耦
2. 异步通信性能高
3. 流量削峰

## 11、RabbitMQ 的集群

镜像集群模式

你创建的 queue，无论元数据还是 queue 里的消息都会存在于多个实例上，然后每次你写消息到 queue 的时候，都会自动把消息到多个实例的 queue 里进行消息同步。

好处在于，你任何一个机器宕机了，没事儿，别的机器都可以用。坏处在于，第一，这个性能开销也太大了吧，消息同步所有机器，导致网络带宽压力和消耗很重！第二，这么玩儿，就没有扩展性可言了，如果某个 queue 负载很重，你加机器，新增的机器也包含了这个 queue 的所有数据，并没有办法线性扩展你的queue

## 12、mq 的缺点

### 系统可用性降低

系统引入的外部依赖越多，越容易挂掉，本来你就是 A 系统调用 BCD 三个系统的接口就好了，人 ABCD 四个系统好好的，没啥问题，你偏加个 MQ 进来，万一MQ 挂了咋整？MQ 挂了，整套系统崩溃了，你不就完了么。

### 系统复杂性提高

硬生生加个 MQ 进来，你怎么保证消息没有重复消费？怎么处理消息丢失的情况？怎么保证消息传递的顺序性？头大头大，问题一大堆，痛苦不已

### 一致性问题

A 系统处理完了直接返回成功了，人都以为你这个请求就成功了；但是问题是， 要是 BCD 三个系统那里，BD 两个系统写库成功了，结果 C 系统写库失败了， 咋整？你这数据就不一致了。

所以消息队列实际是一种非常复杂的架构，你引入它有很多好处，但是也得针对它带来的坏处做各种额外的技术方案和架构来规避掉，最好之后，你会发现，妈呀，系统复杂度提升了一个数量级，也许是复杂了 10 倍。但是关键时刻，用， 还是得用的。

# 十二、Dubbo 面试题

### 1、为什么要用 Dubbo？

随着服务化的进一步发展，服务越来越多，服务之间的调用和依赖关系也越来越复杂，诞生了面向服务的架构体系(SOA)，也因此衍生出了一系列相应的技术， 如对服务提供、服务调用、连接处理、通信协议、序列化方式、服务发现、服务路由、日志输出等行为进行封装的服务框架。就这样为分布式系统的服务治理框架就出现了，Dubbo 也就这样产生了。

### 2、Dubbo 的整体架构设计有哪些分层?

**接口服务层（Service）：**该层与业务逻辑相关，根据 provider 和 consumer 的业务设计对应的接口和实现

**配置层（Config）：**对外配置接口，以 ServiceConfig 和 ReferenceConfig 为中心

**服务代理层（Proxy）：**服务接口透明代理，生成服务的客户端 Stub 和 服务端的 Skeleton，以 ServiceProxy 为中心，扩展接口为 ProxyFactory

**服务注册层（Registry）：**封装服务地址的注册和发现，以服务 URL 为中心， 扩展接口为 RegistryFactory、Registry、RegistryService

**路由层（Cluster）：**封装多个提供者的路由和负载均衡，并桥接注册中心，以Invoker 为中心，扩展接口为 Cluster、Directory、Router 和 LoadBlancce **监控层（Monitor）：**RPC 调用次数和调用时间监控，以 Statistics 为中心， 扩展接口为 MonitorFactory、Monitor 和 MonitorService

**远程调用层（Protocal）：**封装 RPC 调用，以 Invocation 和 Result 为中心， 扩展接口为 Protocal、Invoker 和 Exporter

**信息交换层（Exchange）：**封装请求响应模式，同步转异步。以 Request 和Response 为中心，扩展接口为 Exchanger、ExchangeChannel、ExchangeClient 和 ExchangeServer

**网络 传输 层（Transport）：**抽象 mina 和 netty 为统一接口，以 Message 为中心，扩展接口为 Channel、Transporter、Client、Server 和 Codec

**数据序列化层（Serialize）：**可复用的一些工具，扩展接口为 Serialization、ObjectInput、ObjectOutput 和 ThreadPool

### 3、默认使用的是什么通信框架，还有别的选择吗?

默认也推荐使用 netty 框架，还有 mina。

### 4、服务调用是阻塞的吗？

默认是阻塞的，可以异步调用，没有返回值的可以这么做。Dubbo 是基于 NIO 的非阻塞实现并行调用，客户端不需要启动多线程即可完成并行调用多个远程服务，相对多线程开销较小，异步调用会返回一个 Future 对象。

### 5、一般使用什么注册中心？还有别的选择吗？

推荐使用 Zookeeper 作为注册中心，还有 Redis、Multicast、Simple 注册中心，但不推荐。

### 6、默认使用什么序列化框架，你知道的还有哪些？

推荐使用 Hessian 序列化，还有 Duddo、FastJson、Java 自带序列化。

**7、服务提供者能实现失效踢出是什么原理？** 服务失效踢出基于 zookeeper 的临时节点原理。**8、服务上线怎么不影响旧版本？**

采用多版本开发，不影响旧版本。

**9、如何解决服务调用链过长的问题？** 可以结合 zipkin 实现分布式服务追踪。**10、说说核心的配置有哪些？**

### 11、Dubbo 推荐用什么协议？

**12、同一个服务多个注册的情况下可以直连某一个服务吗？**

可以点对点直连，修改配置即可，也可以通过 telnet 直接某个服务。

### 13、画一画服务注册与发现的流程图？

**14、Dubbo 集群容错有几种方案？**

**15、Dubbo 服务降级，失败重试怎么做？**

可以通过 dubbo:reference 中设置 mock="return null"。mock 的值也可以修改为 true，然后再跟接口同一个路径下实现一个 Mock 类，命名规则是 “接口名称+Mock” 后缀。然后在 Mock 类里实现自己的降级逻辑

### 16、Dubbo 使用过程中都遇到了些什么问题？

在注册中心找不到对应的服务,检查 service 实现类是否添加了@service 注解无法连接到注册中心,检查配置文件中的对应的测试 ip 是否正确

### 17、Dubbo Monitor 实现原理？

Consumer 端在发起调用之前会先走 filter 链；provider 端在接收到请求时也是先走 filter 链，然后才进行真正的业务逻辑处理。默认情况下，在 consumer 和 provider 的 filter 链中都会有 Monitorfilter。

1、MonitorFilter 向 DubboMonitor 发送数据

2、DubboMonitor 将数据进行聚合后（默认聚合 1min 中的统计数据）暂存到ConcurrentMap<Statistics, AtomicReference> statisticsMap，然后使用一个含有 3 个线程（线程名字：DubboMonitorSendTimer）的线程池每隔 1min 钟，调用SimpleMonitorService 遍历发送 statisticsMap 中的统计数据，每发送完毕一个， 就重置当前的 Statistics 的 AtomicReference

3、SimpleMonitorService 将这些聚合数据塞入 BlockingQueue queue 中（队列大写为 100000）

4 、 SimpleMonitorService 使 用 一 个 后 台 线 程 （ 线 程 名 为 ： DubboMonitorAsyncWriteLogThread）将 queue 中的数据写入文件（该线程以死循环的形式来写）

5、SimpleMonitorService 还会使用一个含有 1 个线程（线程名字：

DubboMonitorTimer）的线程池每隔 5min 钟，将文件中的统计数据画成图表

### 18、Dubbo 用到哪些设计模式？

Dubbo 框架在初始化和通信过程中使用了多种设计模式，可灵活控制类加载、权限控制等功能。

#### 工厂模式

Provider 在 export 服务时，会调用 ServiceConfig 的 export 方法。ServiceConfig 中有个字段：

private static final Protocol protocol = ExtensionLoader.getExtensionLoader(Protocol.class).getAdaptiveExtensi on();复制代码

Dubbo 里有很多这种代码。这也是一种工厂模式，只是实现类的获取采用了

JDKSPI 的机制。这么实现的优点是可扩展性强，想要扩展实现，只需要在classpath 下增加个文件就可以了，代码零侵入。另外，像上面的 Adaptive 实

现，可以做到调用时动态决定调用哪个实现，但是由于这种实现采用了动态代理， 会造成代码调试比较麻烦，需要分析出实际调用的实现类。

#### 装饰器模式

Dubbo 在启动和调用阶段都大量使用了装饰器模式。以 Provider 提供的调用链为例，具体的调用链代码是在 ProtocolFilterWrapper 的 buildInvokerChain 完成的，具体是将注解中含有 group=provider 的 Filter 实现，按照 order 排序，最后的调用顺序是：

EchoFilter -> ClassLoaderFilter -> GenericFilter -> ContextFilter -> ExecuteLimitFilter -> TraceFilter -> TimeoutFilter -> MonitorFilter ->

ExceptionFilter 复制代码

更确切地说，这里是装饰器和责任链模式的混合使用。例如，EchoFilter 的作用是判断是否是回声测试请求，是的话直接返回内容，这是一种责任链的体现。而像 ClassLoaderFilter 则只是在主功能上添加了功能，更改当前线程的ClassLoader，这是典型的装饰器模式。

#### 观察者模式

Dubbo 的 Provider 启动时，需要与注册中心交互，先注册自己的服务，再订阅自己的服务，订阅时，采用了观察者模式，开启一个 listener。注册中心会每 5 秒定时检查是否有服务更新，如果有更新，向该服务的提供者发送一个 notify 消息，provider 接受到 notify 消息后，运行 NotifyListener 的 notify 方法，执行监听器方法。

#### 动态代理模式

Dubbo 扩展 JDK SPI 的类 ExtensionLoader 的 Adaptive 实现是典型的动态代理实现。Dubbo 需要灵活地控制实现类，即在调用阶段动态地根据参数决定调用哪个实现类，所以采用先生成代理类的方法，能够做到灵活的调用。生成代理类的代码是 ExtensionLoader 的 createAdaptiveExtensionClassCode 方法。代理类主要逻辑是，获取 URL 参数中指定参数的值作为获取实现类的 key。

### 19、Dubbo 配置文件是如何加载到 Spring 中的？

Spring 容器在启动的时候，会读取到 Spring 默认的一些 schema 以及 Dubbo 自定义的 schema，每个 schema 都会对应一个自己的 NamespaceHandler， NamespaceHandler 里面通过 BeanDefinitionParser 来解析配置信息并转化为需要加载的 bean 对象！

### 20、Dubbo SPI 和 Java SPI 区别？

#### JDK SPI：

JDK 标准的 SPI 会一次性加载所有的扩展实现，如果有的扩展吃实话很耗时， 但也没用上，很浪费资源。所以只希望加载某个的实现，就不现实了

#### DUBBO SPI：

1、对 Dubbo 进行扩展，不需要改动 Dubbo 的源码

2、延迟加载，可以一次只加载自己想要加载的扩展实现。

3、增加了对扩展点 IOC 和 AOP 的支持，一个扩展点可以直接 setter 注入其它扩展点。

4、Dubbo 的扩展机制能很好的支持第三方 IoC 容器，默认支持 Spring Bean。

### 21、Dubbo 支持分布式事务吗？

目前暂时不支持，可与通过 tcc-transaction 框架实现

介绍：tcc-transaction 是开源的 TCC 补偿性分布式事务框架

TCC-Transaction 通过 Dubbo 隐式传参的功能，避免自己对业务代码的入侵。

### 22、Dubbo 可以对结果进行缓存吗？

为了提高数据访问的速度。Dubbo 提供了声明式缓存，以减少用户加缓存的工作量<dubbo:reference cache="true" />

其实比普通的配置文件就多了一个标签 cache="true"

### 23、服务上线怎么兼容旧版本？

可以用版本号（version）过渡，多个不同版本的服务注册到注册中心，版本号不同的服务相互间不引用。这个和服务分组的概念有一点类似。

**24、Dubbo 必须依赖的包有哪些？** Dubbo 必须依赖 JDK，其他为可选。**25、Dubbo telnet 命令能做什么？**

dubbo 服务发布之后，我们可以利用 telnet 命令进行调试、管理。Dubbo2.0.5 以上版本服务提供端口支持 telnet 命令

连接服务

telnet localhost 20880 //键入回车进入 Dubbo 命令模式。查看服务列表

dubbo>ls com.test.TestService

dubbo>ls com.test.TestService create

delete

query 复制代码

* ls (list services and methods)
* ls : 显示服务列表。
* ls -l : 显示服务详细信息列表。
* ls XxxService：显示服务的方法列表。
* ls -l XxxService：显示服务的方法详细信息列表。

### 26、Dubbo 支持服务降级吗？

以通过 dubbo:reference 中设置 mock="return null"。mock 的值也可以修改为 true，然后再跟接口同一个路径下实现一个 Mock 类，命名规则是 “接口名称+Mock” 后缀。然后在 Mock 类里实现自己的降级逻辑

### 27、Dubbo 如何优雅停机？

Dubbo 是通过 JDK 的 ShutdownHook 来完成优雅停机的，所以如果使用 kill -9 PID 等强制关闭指令，是不会执行优雅停机的，只有通过 kill PID 时，才会执行。

### 28、Dubbo 和 Dubbox 之间的区别？

Dubbox 是继 Dubbo 停止维护后，当当网基于 Dubbo 做的一个扩展项目，如加了服务可 Restful 调用，更新了开源组件等。

### 29、Dubbo 和 Spring Cloud 的区别？

根据微服务架构在各方面的要素，看看 Spring Cloud 和 Dubbo 都提供了哪些支持。

使用 Dubbo 构建的微服务架构就像组装电脑，各环节我们的选择自由度很高， 但是最终结果很有可能因为一条内存质量不行就点不亮了，总是让人不怎么放 心，但是如果你是一名高手，那这些都不是问题；而 Spring Cloud 就像品牌机， 在 Spring Source 的整合下，做了大量的兼容性测试，保证了机器拥有更高的稳定性，但是如果要在使用非原装组件外的东西，就需要对其基础有足够的了解。

### 30、你还了解别的分布式框架吗？

别的还有 spring 的 spring cloud，facebook 的 thrift，twitter 的 finagle 等

# 十三、Elasticsearch 面试题

## 1、elasticsearch 了解多少，说说你们公司 es 的集群架构，索引数据大小，分片有多少，以及一些调优手段 。

面试官：想了解应聘者之前公司接触的 ES 使用场景、规模，有没有做过比较大规模的索引设计、规划、调优。

解答：如实结合自己的实践场景回答即可。

比如：ES 集群架构 13 个节点，索引根据通道不同共 20+索引，根据日期，每日递增 20+，索引：10 分片，每日递增 1 亿+数据，每个通道每天索引大小控制：150GB 之内。

仅索引层面调优手段：

### 、设计阶段调优

1. 根据业务增量需求，采取基于日期模板创建索引，通过 roll over API 滚动索引；
2. 使用别名进行索引管理；
3. 每天凌晨定时对索引做 force\_merge 操作，以释放空间；
4. 采取冷热分离机制，热数据存储到 SSD，提高检索效率；冷数据定期进行shrink 操作，以缩减存储；
5. 采取 curator 进行索引的生命周期管理；
6. 仅针对需要分词的字段，合理的设置分词器；
7. Mapping 阶段充分结合各个字段的属性，是否需要检索、是否需要存储等。……..

### 、写入调优

1. 写入前副本数设置为 0；
2. 写入前关闭 refresh\_interval 设置为-1，禁用刷新机制；
3. 写入过程中：采取 bulk 批量写入；
4. 写入后恢复副本数和刷新间隔；
5. 尽量使用自动生成的 id。

### 、查询调优

1. 禁用 wildcard；
2. 禁用批量 terms（成百上千的场景）；
3. 充分利用倒排索引机制，能 keyword 类型尽量 keyword；
4. 数据量大时候，可以先基于时间敲定索引再检索；
5. 设置合理的路由机制。

### 、其他调优

部署调优，业务调优等。

上面的提及一部分，面试者就基本对你之前的实践或者运维经验有所评估了。

## 2、elasticsearch 的倒排索引是什么

面试官：想了解你对基础概念的认知。解答：通俗解释一下就可以。

传统的我们的检索是通过文章，逐个遍历找到对应关键词的位置。

而倒排索引，是通过分词策略，形成了词和文章的映射关系表，这种词典+映射表即为倒排索引。有了倒排索引，就能实现 o（1）时间复杂度的效率检索文章了，极大的提高了检索效率。

学术的解答方式：

倒排索引，相反于一篇文章包含了哪些词，它从词出发，记载了这个词在哪些文档中出现过，由两部分组成——词典和倒排表。

加分项：倒排索引的底层实现是基于：FST（Finite State Transducer）数据结构。

lucene 从 4+版本后开始大量使用的数据结构是 FST。FST 有两个优点：

1. 空间占用小。通过对词典中单词前缀和后缀的重复利用，压缩了存储空间；
2. 查询速度快。O(len(str))的查询时间复杂度。

## 3、elasticsearch 索引数据多了怎么办，如何调优， 部署

面试官：想了解大数据量的运维能力。

解答：索引数据的规划，应在前期做好规划，正所谓“设计先行，编码在后”， 这样才能有效的避免突如其来的数据激增导致集群处理能力不足引发的线上客 户检索或者其他业务受到影响。

如何调优，正如问题 1 所说，这里细化一下：

### 动态索引层面

基于模板+时间+rollover api 滚动创建索引，举例：设计阶段定义：blog 索引的模板格式为：blog\_index\_时间戳的形式，每天递增数据。这样做的好处：不至于数据量激增导致单个索引数据量非常大，接近于上线 2 的 32 次幂-1，索引存储达到了 TB+甚至更大。

一旦单个索引很大，存储等各种风险也随之而来，所以要提前考虑+及早避免。

### 存储层面

冷热数据分离存储，热数据（比如最近 3 天或者一周的数据），其余为冷数据。对于冷数据不会再写入新数据，可以考虑定期 force\_merge 加 shrink 压缩操作，节省存储空间和检索效率。

### 部署层面

一旦之前没有规划，这里就属于应急策略。

结合 ES 自身的支持动态扩展的特点，动态新增机器的方式可以缓解集群压力， 注意：如果之前主节点等规划合理，不需要重启集群也能完成动态新增的。

## 4、elasticsearch 是如何实现 master 选举的

面试官：想了解 ES 集群的底层原理，不再只关注业务层面了。解答：

前置前提：

1. 只有候选主节点（master：true）的节点才能成为主节点。
2. 最小主节点数（min\_master\_nodes）的目的是防止脑裂。

核对了一下代码，核心入口为 findMaster，选择主节点成功返回对应 Master， 否则返回 null。选举流程大致描述如下：

第一步：确认候选主节点数达标，elasticsearch.yml 设置的值discovery.zen.minimum\_master\_nodes；

第二步：比较：先判定是否具备 master 资格，具备候选主节点资格的优先返回； 若两节点都为候选主节点，则 id 小的值会主节点。注意这里的 id 为 string 类型。

题外话：获取节点 id 的方法。

1GET /\_cat/nodes?v&h=ip,port,heapPercent,heapMax,id,name 2ip port heapPercent heapMax id name 复制代码

## 5、详细描述一下 Elasticsearch 索引文档的过程

面试官：想了解 ES 的底层原理，不再只关注业务层面了。解答：

这里的索引文档应该理解为文档写入 ES，创建索引的过程。

文档写入包含：单文档写入和批量 bulk 写入，这里只解释一下：单文档写入流程。

记住官方文档中的这个图。

第一步：客户写集群某节点写入数据，发送请求。（如果没有指定路由/协调节点，请求的节点扮演路由节点的角色。）

第二步：节点 1 接受到请求后，使用文档\_id 来确定文档属于分片 0。请求会被转到另外的节点，假定节点 3。因此分片 0 的主分片分配到节点 3 上。

第三步：节点 3 在主分片上执行写操作，如果成功，则将请求并行转发到节点 1

和节点 2 的副本分片上，等待结果返回。所有的副本分片都报告成功，节点 3 将向协调节点（节点 1）报告成功，节点 1 向请求客户端报告写入成功。

如果面试官再问：第二步中的文档获取分片的过程？

回答：借助路由算法获取，路由算法就是根据路由和文档 id 计算目标的分片 id 的过程。

1shard = hash(\_routing) % (num\_of\_primary\_shards)复制代码

## 6、详细描述一下 Elasticsearch 搜索的过程？

面试官：想了解 ES 搜索的底层原理，不再只关注业务层面了。解答：

搜索拆解为“query then fetch” 两个阶段。query 阶段的目的：定位到位置，但不取。

步骤拆解如下：

1. 假设一个索引数据有 5 主+1 副本 共 10 分片，一次请求会命中（主或者副本分片中）的一个。
2. 每个分片在本地进行查询，结果返回到本地有序的优先队列中。
3. 第 2）步骤的结果发送到协调节点，协调节点产生一个全局的排序列表。fetch 阶段的目的：取数据。

路由节点获取所有文档，返回给客户端。

## 7、Elasticsearch 在部署时，对 Linux 的设置有哪些优化方法

面试官：想了解对 ES 集群的运维能力。解答：

1. 关闭缓存 swap;
2. 堆内存设置为：Min（节点内存/2, 32GB）;
3. 设置最大文件句柄数；
4. 线程池+队列大小根据业务需要做调整；
5. 磁盘存储 raid 方式——存储有条件使用 RAID10，增加单节点性能以及避免单节点存储故障。

## 8、lucence 内部结构是什么？

面试官：想了解你的知识面的广度和深度。解答：

Lucene 是有索引和搜索的两个过程，包含索引创建，索引，搜索三个要点。可以基于这个脉络展开一些。

## 9、Elasticsearch 是如何实现 Master 选举的？

1. Elasticsearch 的选主是 ZenDiscovery 模块负责的，主要包含 Ping（ 节点之间通过这个 RPC 来发现彼此）和 Unicast（单播模块包含一个主机列表以控制哪些节点需要 ping 通）这两部分；
2. 对所有可以成为 master 的节点（node.master: true）根据 nodeId 字典排序，每次选举每个节点都把自己所知道节点排一次序，然后选出第一个（第 0 位）节点，暂且认为它是 master 节点。
3. 如果对某个节点的投票数达到一定的值（可以成为 master 节点数n/2+1）并且该节点自己也选举自己，那这个节点就是 master。否则重新选举一直到满足上述条件。
4. 补充：master 节点的职责主要包括集群、节点和索引的管理，不负责文档级别的管理；data 节点可以关闭 http 功能\*。

## 10、Elasticsearch 中的节点（比如共 20 个），其中的 10 个

选了一个 master，另外 10 个选了另一个 master，怎么办？

（1）当集群 master 候选数量不小于 3 个时，可以通过设置最少投票通过数量

（discovery.zen.minimum\_master\_nodes）超过所有候选节点一半以上来解决脑裂问题；

（3）当候选数量为两个时，只能修改为唯一的一个 master 候选，其他作为 data节点，避免脑裂问题。

## 11、客户端在和集群连接时，如何选择特定的节点执行请求的？

TransportClient 利用 transport 模块远程连接一个 elasticsearch 集群。它并不加入到集群中，只是简单的获得一个或者多个初始化的 transport 地址， 并以 轮询 的方式与这些地址进行通信。

## 12、详细描述一下 Elasticsearch 索引文档的过程。

协调节点默认使用文档 ID 参与计算（也支持通过 routing），以便为路由提供合适的分片。

shard = hash(document\_id) % (num\_of\_primary\_shards)复制代码

1. 当分片所在的节点接收到来自协调节点的请求后，会将请求写入到MemoryBuffer，然后定时（默认是每隔 1 秒）写入到 Filesystem Cache，这个从 MomeryBuffer 到 Filesystem Cache 的过程就叫做 refresh；
2. 当然在某些情况下，存在 Momery Buffer 和 Filesystem Cache 的数据可能会丢失，ES 是通过 translog 的机制来保证数据的可靠性的。其实现机制是接收到请求后，同时也会写入到 translog 中 ，当 Filesystem cache 中的数据写入到磁盘中时，才会清除掉，这个过程叫做 flush；
3. 在 flush 过程中，内存中的缓冲将被清除，内容被写入一个新段，段的fsync 将创建一个新的提交点，并将内容刷新到磁盘，旧的 translog 将被删除并开始一个新的 translog。
4. flush 触发的时机是定时触发（默认 30 分钟）或者 translog 变得太大

（默认为 512M）时；

补充：关于 Lucene 的 Segement：

1. Lucene 索引是由多个段组成，段本身是一个功能齐全的倒排索引。
2. 段是不可变的，允许 Lucene 将新的文档增量地添加到索引中，而不用从头重建索引。
3. 对于每一个搜索请求而言，索引中的所有段都会被搜索，并且每个段会消耗 CPU 的时钟周、文件句柄和内存。这意味着段的数量越多，搜索性能会越低。
4. 为了解决这个问题，Elasticsearch 会合并小段到一个较大的段，提交新的合并段到磁盘，并删除那些旧的小段。

### 13、详细描述一下 Elasticsearch 更新和删除文档的过程。

1. 删除和更新也都是写操作，但是 Elasticsearch 中的文档是不可变的， 因此不能被删除或者改动以展示其变更；
2. 磁盘上的每个段都有一个相应的.del 文件。当删除请求发送后，文档并没有真的被删除，而是在.del 文件中被标记为删除。该文档依然能匹配查询， 但是会在结果中被过滤掉。当段合并时，在.del 文件中被标记为删除的文档将不会被写入新段。
3. 在新的文档被创建时，Elasticsearch 会为该文档指定一个版本号，当执行更新时，旧版本的文档在.del 文件中被标记为删除，新版本的文档被索引到一个新段。旧版本的文档依然能匹配查询，但是会在结果中被过滤掉。

### 14、详细描述一下 Elasticsearch 搜索的过程。

1. 搜索被执行成一个两阶段过程，我们称之为 Query Then Fetch；
2. 在初始查询阶段时，查询会广播到索引中每一个分片拷贝（主分片或者副本分片）。 每个分片在本地执行搜索并构建一个匹配文档的大小为 from + size 的优先队列。

PS：在搜索的时候是会查询 Filesystem Cache 的，但是有部分数据还在MemoryBuffer，所以搜索是近实时的。

1. 每个分片返回各自优先队列中 所有文档的 ID 和排序值 给协调节点，它合并这些值到自己的优先队列中来产生一个全局排序后的结果列表。
2. 接下来就是 取回阶段，协调节点辨别出哪些文档需要被取回并向相关的分片提交多个 GET 请求。每个分片加载并 丰 富 文档，如果有需要的话，接着返回文档给协调节点。一旦所有的文档都被取回了，协调节点返回结果给客户端。
3. 补充：Query Then Fetch 的搜索类型在文档相关性打分的时候参考的是本分片的数据，这样在文档数量较少的时候可能不够准确，DFS Query Then Fetch 增加了一个预查询的处理，询问 Term 和 Document frequency，这个评分更准确，但是性能会变差。\*

### 15、在 Elasticsearch 中，是怎么根据一个词找到对应的倒排索引的？

1. Lucene 的索引过程，就是按照全文检索的基本过程，将倒排表写成此文件格式的过程。
2. Lucene 的搜索过程，就是按照此文件格式将索引进去的信息读出来，然后计算每篇文档打分(score)的过程。

### 16、Elasticsearch 在部署时，对 Linux 的设置有哪些优化方法？

（1）64 GB 内存的机器是非常理想的， 但是 32 GB 和 16 GB 机器也是很常见的。少于 8 GB 会适得其反。

1. 如果你要在更快的 CPUs 和更多的核心之间选择，选择更多的核心更好。多个内核提供的额外并发远胜过稍微快一点点的时钟频率。
2. 如果你负担得起 SSD，它将远远超出任何旋转介质。 基于 SSD 的节点， 查询和索引性能都有提升。如果你负担得起，SSD 是一个好的选择。
3. 即使数据中心们近在咫尺，也要避免集群跨越多个数据中心。绝对要避免集群跨越大的地理距离。
4. 请确保运行你应用程序的 JVM 和服务器的 JVM 是完全一样的。 在Elasticsearch 的几个地方，使用 Java 的本地序列化。
5. 通过设置 gateway.recover\_after\_nodes、gateway.expected\_nodes、gateway.recover\_after\_time 可以在集群重启的时候避免过多的分片交换，这可能会让数据恢复从数个小时缩短为几秒钟。
6. Elasticsearch 默认被配置为使用单播发现，以防止节点无意中加入集群。只有在同一台机器上运行的节点才会自动组成集群。最好使用单播代替组播。
7. 不要随意修改垃圾回收器（CMS）和各个线程池的大小。
8. 把你的内存的（少于）一半给 Lucene（但不要超过 32 GB！），通过ES\_HEAP\_SIZE 环境变量设置。
9. 内存交换到磁盘对服务器性能来说是致命的。如果内存交换到磁盘上， 一个 100 微秒的操作可能变成 10 毫秒。 再想想那么多 10 微秒的操作时延累加起来。 不难看出 swapping 对于性能是多么可怕。
10. Lucene 使用了大 量 的文件。同时，Elasticsearch 在节点和 HTTP 客户端之间进行通信也使用了大量的套接字。 所有这一切都需要足够的文件描述符。你应该增加你的文件描述符，设置一个很大的值，如 64,000。

补充：索引阶段性能提升方法

1. 使用批量请求并调整其大小：每次批量数据 5–15 MB 大是个不错的起始点。
2. 存储：使用 SSD
3. 段和合并：Elasticsearch 默认值是 20 MB/s，对机械磁盘应该是个不错的设置。如果你用的是 SSD，可以考虑提高到 100–200 MB/s。如果你在做批量导入，完全不在意搜索，你可以彻底关掉合并限流。另外还可以增加index.translog.flush\_threshold\_size 设置，从默认的 512 MB 到更大一些的值，比如 1 GB，这可以在一次清空触发的时候在事务日志里积累出更大的段。
4. 如果你的搜索结果不需要近实时的准确度，考虑把每个索引的index.refresh\_interval 改到 30s。
5. 如果你在做大批量导入，考虑通过设置 index.number\_of\_replicas: 0 关闭副本。

### 17、对于 GC 方面，在使用 Elasticsearch 时要注意什么？

1. 倒排词典的索引需要常驻内存，无法 GC，需要监控 data node 上segmentmemory 增长趋势。
2. 各类缓存，field cache, filter cache, indexing cache, bulk queue 等等，要设置合理的大小，并且要应该根据最坏的情况来看 heap 是否够用，也就是各类缓存全部占满的时候，还有 heap 空间可以分配给其他任务吗？避免采用clear cache 等“自欺欺人”的方式来释放内存。
3. 避免返回大量结果集的搜索与聚合。确实需要大量拉取数据的场景，可以采用 scan & scroll api 来实现。
4. cluster stats 驻留内存并无法水平扩展，超大规模集群可以考虑分拆成多个集群通过 tribe node 连接。
5. 想知道 heap 够不够，必须结合实际应用场景，并对集群的 heap 使用情况做持续的监控。
6. 根据监控数据理解内存需求，合理配置各类 circuit breaker，将内存溢出风险降低到最低

### 18、Elasticsearch 对于大数据量（上亿量级）的聚合如何实现？

Elasticsearch 提供的首个近似聚合是 cardinality 度量。它提供一个字段的基数，即该字段的 distinct 或者 unique 值的数目。它是基于 HLL 算法的。HLL 会先对我们的输入作哈希运算，然后根据哈希运算的结果中的 bits 做概率估算从而得到基数。其特点是：可配置的精度，用来控制内存的使用（更精确 ＝ 更多内存）；小的数据集精度是非常高的；我们可以通过配置参数，来设置去重需要的固定内存使用量。无论数千还是数十亿的唯一值，内存使用量只与你配置的精确度相关。

### 19、在并发情况下，Elasticsearch 如果保证读写一致？

1. 可以通过版本号使用乐观并发控制，以确保新版本不会被旧版本覆盖，由应用层来处理具体的冲突；
2. 另外对于写操作，一致性级别支持 quorum/one/all，默认为 quorum，即只有当大多数分片可用时才允许写操作。但即使大多数可用，也可能存在因为网络等原因导致写入副本失败，这样该副本被认为故障，分片将会在一个不同的节点上重建。
3. 对于读操作，可以设置 replication 为 sync(默认)，这使得操作在主分片和副本分片都完成后才会返回；如果设置 replication 为 async 时，也可以通过设置搜索请求参数\_preference 为 primary 来查询主分片，确保文档是最新版本。

### 20、如何监控 Elasticsearch 集群状态？

Marvel 让你可以很简单的通过 Kibana 监控 Elasticsearch。你可以实时查看你的集群健康状态和性能，也可以分析过去的集群、索引和节点指标。

### 22、介绍一下你们的个性化搜索方案？

基于 word2vec 和 Elasticsearch 实现个性化搜索

1. 基于 word2vec、Elasticsearch 和自定义的脚本插件，我们就实现了一个个性化的搜索服务，相对于原有的实现，新版的点击率和转化率都有大幅的提升；
2. 基于 word2vec 的商品向量还有一个可用之处，就是可以用来实现相似商品的推荐；
3. 使用 word2vec 来实现个性化搜索或个性化推荐是有一定局限性的，因为它只能处理用户点击历史这样的时序数据，而无法全面的去考虑用户偏好，这个还是有很大的改进和提升的空间；

### 23、是否了解字典树？

常用字典数据结构如下所示：

Trie 的核心思想是空间换时间，利用字符串的公共前缀来降低查询时间的开销以达到提高效率的目的。它有 3 个基本性质：

1. 根节点不包含字符，除根节点外每一个节点都只包含一个字符。
2. 从根节点到某一节点，路径上经过的字符连接起来，为该节点对应的字符串。
3. 每个节点的所有子节点包含的字符都不相同。
4. 可以看到，trie 树每一层的节点数是 26^i 级别的。所以为了节省空间， 我们还可以用动态链表，或者用数组来模拟动态。而空间的花费，不会超过单词数×单词长度。
5. 实现：对每个结点开一个字母集大小的数组，每个结点挂一个链表，使用左儿子右兄弟表示法记录这棵树；
6. 对于中文的字典树，每个节点的子节点用一个哈希表存储，这样就不用浪费太大的空间，而且查询速度上可以保留哈希的复杂度 O(1)。

### 24、拼写纠错是如何实现的？

1. 拼写纠错是基于编辑距离来实现；编辑距离是一种标准的方法，它用来表示经过插入、删除和替换操作从一个字符串转换到另外一个字符串的最小操作步数；
2. 编辑距离的计算过程：比如要计算 batyu 和 beauty 的编辑距离，先创建一个 7×8 的表（batyu 长度为 5，coffee 长度为 6，各加 2），接着，在如下位置填入黑色数字。其他格的计算过程是取以下三个值的最小值：

如果最上方的字符等于最左方的字符，则为左上方的数字。否则为左上方的数字

+1。（对于 3,3 来说为 0）

左方数字+1（对于 3,3 格来说为 2） 上方数字+1（对于 3,3 格来说为 2）

最终取右下角的值即为编辑距离的值 3。

对于拼写纠错，我们考虑构造一个度量空间（Metric Space），该空间内任何关系满足以下三条基本条件：

d(x,y) = 0 -- 假如 x 与 y 的距离为 0，则 x=y

d(x,y) = d(y,x) -- x 到 y 的距离等同于 y 到 x 的距离d(x,y) + d(y,z) >= d(x,z) -- 三角不等式

1. 根据三角不等式，则满足与 query 距离在 n 范围内的另一个字符转 B， 其与 A 的距离最大为 d+n，最小为 d-n。
2. BK 树的构造就过程如下：每个节点有任意个子节点，每条边有个值表示编辑距离。所有子节点到父节点的边上标注 n 表示编辑距离恰好为 n。比如， 我们有棵树父节点是”book”和两个子节点”cake”和”books”，”book”

到”books”的边标号 1，”book”到”cake”的边上标号 4。从字典里构造好树后，无论何时你想插入新单词时，计算该单词与根节点的编辑距离，并且查找数值为 d(neweord, root)的边。递归得与各子节点进行比较，直到没有子节点， 你就可以创建新的子节点并将新单词保存在那。比如，插入”boo”到刚才上述例子的树中，我们先检查根节点，查找 d(“book”, “boo”) = 1 的边，然后检查标号为 1 的边的子节点，得到单词”books”。我们再计算距离d(“books”, “boo”)=2，则将新单词插在”books”之后，边标号为 2。

3、查询相似词如下：计算单词与根节点的编辑距离 d，然后递归查找每个子节点标号为 d-n 到 d+n（包含）的边。假如被检查的节点与搜索单词的距离 d 小于 n，则返回该节点并继续查询。比如输入 cape 且最大容忍距离为 1，则先计算和根的编辑距离 d(“book”, “cape”)=4，然后接着找和根节点之间编辑距离为 3 到 5 的，这个就找到了 cake 这个节点，计算 d(“cake”, “cape”)=1，满足条件所以返回 cake，然后再找和 cake 节点编辑距离是 0 到 2 的，分别找到 cape 和 cart 节点，这样就得到 cape 这个满足条件的结果。

## 十四、Linux 面试题

### 1、绝对路径用什么符号表示？当前目录、上层目录用什么表示？主目录用什么表示? 切换目录用什么命令？

答案：

绝对路径： 如/etc/init.d

当前目录和上层目录： ./ ../ 主目录： ~/

切换目录： cd

### 2、怎么查看当前进程？怎么执行退出？怎么查看当前路径？

答案：

查看当前进程： ps 执行退出： exit

查看当前路径： pwd

### 3、怎么清屏？怎么退出当前命令？怎么执行睡眠？怎么查看当前用户id？查看指定帮助用什么命令？

答案：

清屏： clear

退出当前命令： ctrl+c 彻底退出

执行睡眠 ： ctrl+z 挂起当前进程 fg 恢复后台

查看当前用户 id： ”id“：查看显示目前登陆账户的 uid 和 gid 及所属分组及用户名

查看指定帮助： 如 man adduser 这个很全 而且有例子； adduser --help 这个告诉你一些常用参数； info adduesr；

### 4、Ls 命令执行什么功能？ 可以带哪些参数，有什么区别？

答案：

ls 执行的功能： 列出指定目录中的目录，以及文件

哪些参数以及区别： a 所有文件 l 详细信息，包括大小字节数，可读可写可执行的权限等

### 5、建立软链接(快捷方式)，以及硬链接的命令。

答案：

软链接： ln -s slink source 硬链接： ln link source

### 6、目录创建用什么命令？创建文件用什么命令？复制文件用什么命令？

答案：

创建目录： mkdir

创建文件：典型的如 touch，vi 也可以创建文件，其实只要向一个不存在的文件输出，都会创建文件

复制文件： cp 7. 文件权限修改用什么命令？格式是怎么样的？ 文件权限修改： chmod

格式如下：

chmodu+xfile 给 file 的属主增加执行权限 chmod 751 file 给 file 的属主分配读、写、执行(7)的权限，给 file 的所在组分配读、执行(5)的权限，给其他用户分配执行(1)的权限

chmodu=rwx,g=rx,o=xfile 上例的另一种形式 chmod =r file 为所有用户分配读权限

chmod444file 同上例 chmod a-wx,a+r file 同上例

$ chmod -R u+r directory 递归地给 directory 目录下所有文件和子目录的属主分配读的权限

### 7、查看文件内容有哪些命令可以使用？

答案：

vi 文件名 #编辑方式查看，可修改cat 文件名 #显示全部文件内容more 文件名 #分页显示文件内容

less 文件名 #与 more 相似，更好的是可以往前翻页tail 文件名 #仅查看尾部，还可以指定行数

head 文件名 #仅查看头部,还可以指定行数

### 8、随意写文件命令？怎么向屏幕输出带空格的字符串，比如”hello world”?

答案：

写文件命令：vi

向屏幕输出带空格的字符串:echo hello world

### 9、终端是哪个文件夹下的哪个文件？黑洞文件是哪个文件夹下的哪个命令？

答案：

终端 /dev/tty

黑洞文件 /dev/null

### 10、移动文件用哪个命令？改名用哪个命令？

答案： mv mv

### 11、复制文件用哪个命令？如果需要连同文件夹一块复制呢？如果需要 有提示功能呢？

答案：

cp cp -r ？？？？

### 12、删除文件用哪个命令？如果需要连目录及目录下文件一块删除呢？ 删除空文件夹用什么命令？

答案：

rm rm -r rmdir

### 13、Linux 下命令有哪几种可使用的通配符？分别代表什么含义?

答案：

“？”可替代单个字符。“\*”可替代任意多个字符。

方括号“[charset]”可替代 charset 集中的任何单个字符，如[a-z]，[abABC]

### 14、用什么命令对一个文件的内容进行统计？(行号、单词数、字节数)

答案：

wc 命令 - c 统计字节数 - l 统计行数 - w 统计字数。

### 15、Grep 命令有什么用？ 如何忽略大小写？ 如何查找不含该串的行?

答案：

是一种强大的文本搜索工具，它能使用正则表达式搜索文本，并把匹 配的行打印出来。

grep [stringSTRING] filename grep [^string] filename

### 16、Linux 中进程有哪几种状态？在 ps 显示出来的信息中，分别用什么符号表示的？

答案：

1. 不可中断状态：进程处于睡眠状态，但是此刻进程是不可中断的。不可中断，指进程不响应异步信号。
2. 暂停状态/跟踪状态：向进程发送一个 SIGSTOP 信号，它就会因响应该信号 而进入 TASK\_STOPPED 状态;当进程正在被跟踪时，它处于 TASK\_TRACED 这个特殊的状态。正被跟踪”指的是进程暂停下来，等待跟踪它的进程对它进行操作。
3. 就绪状态：在 run\_queue 队列里的状态
4. 运行状态：在 run\_queue 队列里的状态
5. 可中断睡眠状态：处于这个状态的进程因为等待某某事件的发生（比如等待 socket 连接、等待信号量），而被挂起
6. zombie 状态（僵尸）：父亲没有通过 wait 系列的系统调用会顺便将子进程的尸体（task\_struct）也释放掉
7. 退出状态

D 不可中断 Uninterruptible（usually IO） R 正在运行，或在队列中的进程

S 处于休眠状态T 停止或被追踪Z 僵尸进程

W 进入内存交换（从内核 2.6 开始无效） X 死掉的进程

### 17、怎么使一个命令在后台运行?

答案：

一般都是使用 & 在命令结尾来让程序自动运行。(命令后可以不追加空格)

### 18、利用 ps 怎么显示所有的进程? 怎么利用 ps 查看指定进程的信息？

答案：

ps -ef (system v 输出)

ps -aux bsd 格式输出

ps -ef | grep pid 复制代码

### 19、哪个命令专门用来查看后台任务?

答案： job -l

### 20、把后台任务调到前台执行使用什么命令?把停下的后台任务在后台 执行起来用什么命令?

答案：

把后台任务调到前台执行 fg

把停下的后台任务在后台执行起来 bg

### 21、终止进程用什么命令? 带什么参数?

答案：

kill [-s <信息名称或编号>][程序] 或 kill [-l <信息编号>] kill-9 pid

### 22、怎么查看系统支持的所有信号？

答案： kill -l

### 23、搜索文件用什么命令? 格式是怎么样的?

答案：

find <指定目录> <指定条件> <指定动作>

whereis 加参数与文件名locate 只加文件名

find 直接搜索磁盘，较慢。find / -name "string\*"

### 24、查看当前谁在使用该主机用什么命令? 查找自己所在的终端信息用什么命令?

答案：

查找自己所在的终端信息：who am i 查看当前谁在使用该主机：who

### 25、使用什么命令查看用过的命令列表?

答案： history

### 26、使用什么命令查看磁盘使用空间？ 空闲空间呢?

答案：

df -hl 复制代码

文件系统 容量 已用 可用 已用% 挂载点

Filesystem Size Used Avail Use% Mounted on /dev/hda2 45G 19G 24G 44% /

/dev/hda1 494M 19M 450M 4% /boot 复制代码

### 27、使用什么命令查看网络是否连通?

答案： netstat

### 28、使用什么命令查看 ip 地址及接口信息？

答 案 ： ifconfig

### 29、查看各类环境变量用什么命令?

答案：

查看所有 env

查看某个，如 home： env $HOME

### 30、通过什么命令指定命令提示符?

答案：

\u：显示当前用户账号

\h：显示当前主机名

\W：只显示当前路径最后一个目录

\w：显示当前绝对路径（当前用户目录会以~代替）

$PWD：显示当前全路径

$：显示命令行’$'或者’#'符号#：下达的第几个命令

\d：代表日期，格式为 week day month date，例如："MonAug1"

\t：显示时间为 24 小时格式，如：HH：MM：SS

\T：显示时间为 12 小时格式

\A：显示时间为 24 小时格式：HH：MM

\v：BASH 的版本信息 如 export PS1=’[\u@\h\w#]$‘

### 31、查找命令的可执行文件是去哪查找的? 怎么对其进行设置及添加?

答案：

whereis [-bfmsu][-B <目录>...][-M <目录>...][-S <目录>...][文件...] 补充说明：whereis 指令会在特定目录中查找符合条件的文件。这些文件的烈性应属于原始代码，二进制文件，或是帮助文件。

-b 只查找二进制文件。

-B <目录> 只在设置的目录下查找二进制文件。 -f 不显示文件名前的路径名称。

-m 只查找说明文件。

-M <目录> 只在设置的目录下查找说明文件。-s 只查找原始代码文件。

-S <目录> 只在设置的目录下查找原始代码文件。 -u 查找不包含指定类型的文件。

w -h ich 指令会在 PATH 变量指定的路径中，搜索某个系统命令的位置，并且返回第一个搜索结果。

-n 指定文件名长度，指定的长度必须大于或等于所有文件中最长的文件名。

-p 与-n 参数相同，但此处的包括了文件的路径。 -w 指定输出时栏位的宽度。

-V 显示版本信息

### 32、通过什么命令查找执行命令?

答案：

which 只能查可执行文件

whereis 只能查二进制文件、说明文档，源文件等

### 33、怎么对命令进行取别名？

答案：

alias la='ls -a'复制代码

### 34、du 和 df 的定义，以及区别？

答案：

du 显示目录或文件的大小

df 显示每个<文件>所在的文件系统的信息，默认是显示所有文件系统。（文件系统分配其中的一些磁盘块用来记录它自身的一些数据，如 i 节点，磁盘分布图，间接块，超级块等。这些数据对大多数用户级的程序来说是不可见的，通常称为 Meta Data。） du 命令是用户级的程序，它不考虑 Meta Data，而 df 命令则查看文件系统的磁盘分配图并考虑 Meta Data。

df 命令获得真正的文件系统数据，而 du 命令只查看文件系统的部分情况。

### 35、awk 详解。

答案：

awk '{pattern + action}' { filenames

}#cat /etc/passwd |awk -F ':' '{print 1"t"7}' //-F 的意思是以':'分隔root

/bin/bash

daemon /bin/sh 搜索/etc/passwd 有 root 关键字的所有行#awk -F: '/root/'

/etc/passwd root:x:0:0:root:/root:/bin/bash 复制代码

### 36、当你需要给命令绑定一个宏或者按键的时候，应该怎么做呢？

答案：

可以使用 bind 命令，bind 可以很方便地在 shell 中实现宏或按键的绑定。在进行按键绑定的时候，我们需要先获取到绑定按键对应的字符序列。

比如获取 F12 的字符序列获取方法如下：先按下 Ctrl+V,然后按下 F12 .我们就可以得到 F12 的字符序列 ^[[24~。

接着使用 bind 进行绑定。

[root@localhost ~]# bind ‘”e[24~":"date"'复制代码

注意：相同的按键在不同的终端或终端模拟器下可能会产生不同的字符序列。

【附】也可以使用 showkey -a 命令查看按键对应的字符序列。

### 37、如果一个 linux 新手想要知道当前系统支持的所有命令的列表， 他需要怎么做？

答案：

使用命令 compgen -c，可以打印出所有支持的命令列表。

[root@localhost ~]$ compgen -c l.

ll

lswhichifthen elseelifficaseesacfor

selectwhile untildodone

…复制代码

### 38、如果你的助手想要打印出当前的目录栈，你会建议他怎么做？

答案：

使用 Linux 命令 dirs 可以将当前的目录栈打印出来。

[root@localhost ~]# dirs

/usr/share/X11 复制代码

【附】：目录栈通过 pushd popd 来操作。

### 39、你的系统目前有许多正在运行的任务，在不重启机器的条件下，有 什么方法可以把所有正在运行的进程移除呢？

答案：

使用 linux 命令 ’disown -r ’可以将所有正在运行的进程移除。

### 40、bash shell 中的 hash 命令有什么作用？

答案：

linux 命令’hash’管理着一个内置的哈希表，记录了已执行过的命令的完整路径,用该命令可以打印出你所使用过的命令以及执行的次数。

[root@localhost ~]# hash

hits command

2 /bin/ls

2 /bin/su 复制代码

### 41、哪一个 bash 内置命令能够进行数学运算。

答案：

bash shell 的内置命令 let 可以进行整型数的数学运算。#! /bin/bash

…

…let c=a+b

…

…复制代码

### 42、怎样一页一页地查看一个大文件的内容呢？

答案：

通过管道将命令”cat file\_name.txt” 和 ’more’ 连接在一起可以实现这个需要.

[root@localhost ~]# cat file\_name.txt | more 复制代码

### 43、数据字典属于哪一个用户的？

答案：

数据字典是属于’SYS’用户的，用户‘SYS’ 和 ’SYSEM’是由系统默认自动创建的

### 44、怎样查看一个 linux 命令的概要与用法？假设你在/bin 目录中偶然看到一个你从没见过的的命令，怎样才能知道它的作用和用法呢？

答案：

使用命令 whatis 可以先出显示出这个命令的用法简要，比如，你可以使用whatiszcat 去查看‘zcat’的介绍以及使用简要。

[root@localhost ~]# whatis zcat

zcat [gzip] (1) – compress or expand files 复制代码

### 45、使用哪一个命令可以查看自己文件系统的磁盘空间配额呢？

答案：

使用命令 repquota 能够显示出一个文件系统的配额信息

【附】只有 root 用户才能够查看其它用户的配额。