Java面试题

1. 什么是Java虚拟机？为什么Java被称作是“平台无关的编程语言”？
2. JDK和JRE的区别是什么？
3. ”static”关键字是什么意思？Java中是否可以覆盖(override)一个private或者是static的方法？
4. 是否可以在static环境中访问非static变量？
5. Java支持的数据类型有哪些？什么是自动拆装箱？
6. Java中的方法覆盖(Overriding)和方法重载(Overloading)是什么意思？
7. Java中，什么是构造方法？什么是构造方法重载？什么是复制构造方法？
8. Java支持多继承么？
9. 接口和抽象类的区别是什么？
10. 什么是值传递和引用传递？
11. 进程和线程的区别是什么？
12. 创建线程有几种不同的方式？你喜欢哪一种？为什么？
13. 概括的解释下线程的几种可用状态。
14. 同步方法和同步代码块的区别是什么？
15. 在监视器(Monitor)内部，是如何做线程同步的？程序应该做哪种级别的同步？
16. 什么是死锁(deadlock)？
17. 如何确保N个线程可以访问N个资源同时又不导致死锁？
18. Java集合类框架的基本接口有哪些？
19. 为什么集合类没有实现Cloneable和Serializable接口？
20. 什么是迭代器(Iterator)？
21. Iterator和ListIterator的区别是什么？
22. 快速失败(fail-fast)和安全失败(fail-safe)的区别是什么？
23. Java中的HashMap的工作原理是什么？
24. hhashCode()和equals()方法的重要性体现在什么地方？
25. HashMap和Hashtable有什么区别？
26. 数组(Array)和列表(ArrayList)有什么区别？什么时候应该使用Array而不是ArrayList？
27. ArrayList和LinkedList有什么区别？
28. Comparable和Comparator接口是干什么的？列出它们的区别?
29. 什么是Java优先级队列(Priority Queue)？
30. 你了解大O符号(big-O notation)么？你能给出不同数据结构的例子么？
31. 如何权衡是使用无序的数组还是有序的数组？
32. Java集合类框架的最佳实践有哪些？
33. Enumeration接口和Iterator接口的区别有哪些？
34. HashSet和TreeSet有什么区别？
35. Java中垃圾回收有什么目的？什么时候进行垃圾回收？
36. System.gc()和Runtime.gc()会做什么事情？
37. finalize()方法什么时候被调用？析构函数(fina...
38. 如果对象的引用被置为null，垃圾收集器是否会立即释放对象占用的内存？
39. Java堆的结构是什么样子的？什么是堆中的永久代(Perm Gen space)?
40. 串行(serial)收集器和吞吐量(throughput)收集器的区别是什么？
41. 在Java中，对象什么时候可以被垃圾回收？
42. JVM的永久代中会发生垃圾回收么？
43. Java中的两种异常类型是什么？他们有什么区别？
44. Java中Exception和Error有什么区别？
45. throw和throws有什么区别？
46. 异常处理完成以后，Exception对象会发生什么变化？
47. finally代码块和finalize()方法有什么区别？
48. 什么是Applet？
49. 解释一下Applet的生命周期
50. 当applet被载入的时候会发生什么？
51. Applet和普通的Java应用程序有什么区别？
52. Java applet有哪些限制条件？
53. 什么是不受信任的applet？
54. 从网络上加载的applet和从本地文件系统加载的applet有什么区别？
55. applet类加载器是什么？它会做哪些工作？
56. applet安全管理器是什么？它会做哪些工作？
57. 弹出式选择菜单(Choice)和列表(List)有什么区别
58. 什么是布局管理器？
59. 滚动条(Scrollbar)和滚动面板(JScrollPane)有什么区别？
60. 哪些Swing的方法是线程安全的？
61. 说出三种支持重绘(painting)的组件?
62. 什么是裁剪(clipping)？
63. MenuItem和CheckboxMenuItem的区别是什么？
64. 边缘布局(BorderLayout)里面的元素是如何布局的？
65. 网格包布局(GridBagLayout)里面的元素是如何布局的？
66. Window和Frame有什么区别？
67. 裁剪(clipping)和重绘(repainting)有什么联系？
68. 事件监听器接口(event-listener interface)和事件适配器(event-adapter)有什么关系？
69. GUI组件如何来处理它自己的事件？
70. Java的布局管理器比传统的窗口系统有哪些优势？
71. Java的Swing组件使用了哪种设计模式？
72. 什么是JDBC？
73. 解释下驱动(Driver)在JDBC中的角色。
74. Class.forName()方法有什么作用？
75. PreparedStatement比Statement有什么优势？
76. 什么时候使用CallableStatement？用来准备CallableStatement的方法是什么？
77. 数据库连接池是什么意思？
78. 什么是RMI？
79. RMI体系结构的基本原则是什么？
80. RMI体系结构分哪几层？
81. RMI中的远程接口(Remote Interface)扮演了什么样的角色？
82. java.rmi.Naming类扮演了什么样的角色？
83. RMI的绑定(Binding)是什么意思？
84. Naming类的bind()和rebind()方法有什么区别？
85. 让RMI程序能正确运行有哪些步骤？
86. RMI的stub扮演了什么样的角色？
87. 什么是分布式垃圾回收(DGC)？它是如何工作的？
88. RMI中使用RMI安全管理器(RMISecurityManager)的目的是什么？
89. 解释下Marshalling和demarshalling。
90. 解释下Serialization和Deserialization?
91. 什么是Servlet？
92. 说一下Servlet的体系结构。
93. Applet和Servlet有什么区别？
94. GenericServlet和HttpServlet有什么区别？
95. 解释下Servlet的生命周期。
96. doGet()方法和doPost()方法有什么区别？
97. 什么是Web应用程序？
98. 什么是服务端包含(Server Side Include)？
99. 什么是Servlet链(Servlet Chaining)？
100. 如何知道是哪一个客户端的机器正在请求你的Servlet？
101. HTTP响应的结构是怎么样的？
102. 什么是cookie？session和cookie有什么区别？
103. 浏览器和Servlet通信使用的是什么协议？
104. 什么是HTTP隧道？
105. sendRedirect()和forward()方法有什么区别？
106. 什么是URL编码和URL解码？
107. 什么是JSP页面？
108. JSP请求是如何被处理的？
109. JSP有什么优点？
110. 什么是JSP指令(Directive)？JSP中有哪些不同类型的指令？
111. 什么是JSP动作(JSP action)？
112. 什么是Scriptlets？
113. 声明(Decalaration)在哪里？
114. 什么是表达式(Expression)？
115. 隐含对象是什么意思？有哪些隐含对象？
116. 面向对象软件开发的优点有哪些？
117. 封装的定义和好处有哪些？
118. 多态的定义？
119. 继承的定义？
120. 抽象的定义？抽象和封装的不同点？

**1、并发编程三要素？**

**1）原子性**

原子性指的是一个或者多个操作，要么全部执行并且在执行的过程中不被其他操作打断，要么就全部都不执行。

**2）可见性**

可见性指多个线程操作一个共享变量时，其中一个线程对变量进行修改后，其他线程可以立即看到修改的结果。

**实现可见性的方法：**

synchronized或者Lock：保证同一个时刻只有一个线程获取锁执行代码，锁释放之前把最新的值刷新到主内存，实现可见性。

**3）有序性**

**有序性，即程序的执行顺序按照代码的先后顺序来执行。**

**2、多线程的价值？**

**1）发挥多核CPU的优势**

多线程，可以真正发挥出多核CPU的优势来，达到充分利用CPU的目的，采用多线程的方式去同时完成几件事情而不互相干扰。

**2）防止阻塞**

从程序运行效率的角度来看，单核CPU不但不会发挥出多线程的优势，反而会因为在单核CPU上运行多线程导致线程上下文的切换，而降低程序整体的效率。但是单核CPU我们还是要应用多线程，就是为了防止阻塞。试想，如果单核CPU使用单线程，那么只要这个线程阻塞了，比方说远程读取某个数据吧，对端迟迟未返回又没有设置超时时间，那么你的整个程序在数据返回回来之前就停止运行了。多线程可以防止这个问题，多条线程同时运行，哪怕一条线程的代码执行读取数据阻塞，也不会影响其它任务的执行。

**3）便于建模**

这是另外一个没有这么明显的优点了。假设有一个大的任务A，单线程编程，那么就要考虑很多，建立整个程序模型比较麻烦。但是如果把这个大的任务A分解成几个小任务，任务B、任务C、任务D，分别建立程序模型，并通过多线程分别运行这几个任务，那就简单很多了。

**3、创建线程的有哪些方式？**

1）继承Thread类创建线程类

2）通过Runnable接口创建线程类

3）通过Callable和Future创建线程

**4.创建线程的三种方式的对比？**

**1）采用实现Runnable、Callable接口的方式创建多线程。**

**优势是**：

线程类只是实现了Runnable接口或Callable接口，还可以继承其他类。

在这种方式下，多个线程可以共享同一个target对象，所以非常适合多个相同线程来处理同一份资源的情况，从而可以将CPU、代码和数据分开，形成清晰的模型，较好地体现了面向对象的思想。

**劣势是：**

编程稍微复杂，如果要访问当前线程，则必须使用Thread.currentThread()方法。

**2）使用继承Thread类的方式创建多线程**

**优势是：**

编写简单，如果需要访问当前线程，则无需使用Thread.currentThread()方法，直接使用this即可获得当前线程。

**劣势是：**

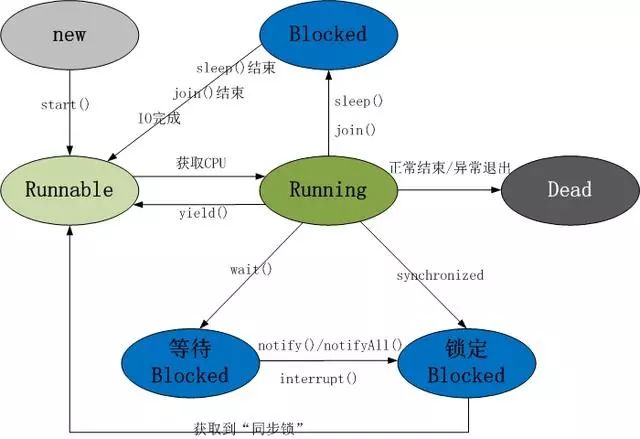
线程类已经继承了Thread类，所以不能再继承其他父类。

**3）Runnable和Callable的区别**

* Callable规定（重写）的方法是call()，Runnable规定（重写）的方法是run()。
* Callable的任务执行后可返回值，而Runnable的任务是不能返回值的。
* Call方法可以抛出异常，run方法不可以。
* 运行Callable任务可以拿到一个Future对象，表示异步计算的结果。它提供了检查计算是否完成的方法，以等待计算的完成，并检索计算的结果。通过Future对象可以了解任务执行情况，可取消任务的执行，还可获取执行结果。

**5、线程的状态流转图**

线程的生命周期及五种基本状态：



**Java线程具有五中基本状态**

**1）新建状态（New）**：当线程对象对创建后，即进入了新建状态，如：Thread t = new MyThread()；

**2）就绪状态（Runnable）**：当调用线程对象的start()方法（t.start();），线程即进入就绪状态。处于就绪状态的线程，只是说明此线程已经做好了准备，随时等待CPU调度执行，并不是说执行了t.start()此线程立即就会执行；

**3）运行状态（Running）**：当CPU开始调度处于就绪状态的线程时，此时线程才得以真正执行，即进入到运行状态。注：就 绪状态是进入到运行状态的唯一入口，也就是说，线程要想进入运行状态执行，首先必须处于就绪状态中；

**4）阻塞状态（Blocked）**：处于运行状态中的线程由于某种原因，暂时放弃对CPU的使用权，停止执行，此时进入阻塞状态，直到其进入到就绪状态，才 有机会再次被CPU调用以进入到运行状态。根据阻塞产生的原因不同，阻塞状态又可以分为三种：

1.等待阻塞：运行状态中的线程执行wait()方法，使本线程进入到等待阻塞状态；

2.同步阻塞 -- 线程在获取synchronized同步锁失败(因为锁被其它线程所占用)，它会进入同步阻塞状态；

3.其他阻塞 -- 通过调用线程的sleep()或join()或发出了I/O请求时，线程会进入到阻塞状态。当sleep()状态超时、join()等待线程终止或者超时、或者I/O处理完毕时，线程重新转入就绪状态。

**5）死亡状态（Dead）**：线程执行完了或者因异常退出了run()方法，该线程结束生命周期。

**6.什么是线程池？ 有哪几种创建方式？**

线程池就是提前创建若干个线程，如果有任务需要处理，线程池里的线程就会处理任务，处理完之后线程并不会被销毁，而是等待下一个任务。由于创建和销毁线程都是消耗系统资源的，所以当你想要频繁的创建和销毁线程的时候就可以考虑使用线程池来提升系统的性能。

java 提供了一个 java.util.concurrent.Executor接口的实现用于创建线程池。

**四种线程池的创建**：

（1）newCachedThreadPool创建一个可缓存线程池

（2）newFixedThreadPool 创建一个定长线程池，可控制线程最大并发数。

（3）newScheduledThreadPool 创建一个定长线程池，支持定时及周期性任务执行。

（4）newSingleThreadExecutor 创建一个单线程化的线程池，它只会用唯一的工作线程来执行任务。

**7.线程池的优点？**

1）重用存在的线程，减少对象创建销毁的开销。

2）可有效的控制最大并发线程数，提高系统资源的使用率，同时避免过多资源竞争，避免堵塞。

3）提供定时执行、定期执行、单线程、并发数控制等功能。

**8.Java中的同步集合与并发集合有什么区别？**

**同步集合类：**

* Vector
* Stack
* HashTable
* Collections.synchronized方法生成

**并发集合类：**

* ConcurrentHashMap
* CopyOnWriteArrayList
* CopyOnWriteArraySet等

**9.同步集合与并发集合的区别**

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。同步集合比并发集合会慢得多，主要原因是锁，同步集合会对整个May或List加锁，而并发集合例如ConcurrentHashMap， 把整个Map 划分成几个片段，只对相关的几个片段上锁，同时允许多线程访问其他未上锁的片段(JDK1.8版本底层加入了红黑树)。

**10.常用的并发工具类有哪些？**

* CountDownLatch
* CyclicBarrier
* Semaphore
* Exchanger

**11.CyclicBarrier和CountDownLatch的应用场景？**

**CountDownLatch** : **一个线程**(或者多个)， 等待另外**N个线程**完成**某个事情**之后才能执行。**CyclicBarrier** : **N个线程**相互等待，任何一个线程完成之前，所有的线程都必须等待。

**CountDownLatch的使用场景：**

在一些应用场合中，需要等待某个条件达到要求后才能做后面的事情；同时当线程都完成后也会触发事件，以便进行后面的操作, 这个时候就可以使用CountDownLatch。

**CyclicBarrier 使用场景**

CyclicBarrier可以用于多线程计算数据，最后合并计算结果的应用场景。

**12.CyclicBarrier和CountDownLatch的区别**

1)CountDownLatch简单的说就是一个线程等待，直到他所等待的其他线程都执行完成并且调用countDown()方法发出通知后，当前线程才可以继续执行。

2)cyclicBarrier是所有线程都进行等待，直到所有线程都准备好进入await()方法之后，所有线程同时开始执行！

3)CountDownLatch的计数器只能使用一次。而CyclicBarrier的计数器可以使用reset() 方法重置。所以CyclicBarrier能处理更为复杂的业务场景，比如如果计算发生错误，可以重置计数器，并让线程们重新执行一次。

4)CyclicBarrier还提供其他有用的方法，比如getNumberWaiting方法可以获得CyclicBarrier阻塞的线程数量。isBroken方法用来知道阻塞的线程是否被中断。如果被中断返回true，否则返回false。

**13.synchronized的作用？**

在Java中，synchronized关键字是用来控制线程同步的，就是在多线程的环境下，控制synchronized代码段不被多个线程同时执行。

**synchronized既可以加在一段代码上，也可以加在方法上。**

**14.volatile关键字的作用**

对于可见性，Java提供了volatile关键字来保证可见性。

当一个共享变量被volatile修饰时，它会保证修改的值会立即被更新到主存，当有其他线程需要读取时，它会去内存中读取新值。

从实践角度而言，volatile的一个重要作用就是和CAS结合，保证了原子性，详细的可以参见java.util.concurrent.atomic包下的类，比如AtomicInteger。

**15.什么是CAS**

CAS是compare and swap的缩写，即我们所说的比较交换。

cas是一种基于锁的操作，而且是乐观锁。在java中锁分为乐观锁和悲观锁。悲观锁是将资源锁住，等一个之前获得锁的线程释放锁之后，下一个线程才可以访问。而乐观锁采取了一种宽泛的态度，通过某种方式不加锁来处理资源，比如通过给记录加version来获取数据，性能较悲观锁有很大的提高。

CAS 操作包含三个操作数 —— 内存位置（V）、预期原值（A）和新值(B)。如果内存地址里面的值和A的值是一样的，那么就将内存里面的值更新成B。CAS是通过无限循环来获取数据的，若果在第一轮循环中，a线程获取地址里面的值被b线程修改了，那么a线程需要自旋，到下次循环才有可能机会执行。

java.util.concurrent.atomic 包下的类大多是使用CAS操作来实现的( AtomicInteger,AtomicBoolean,AtomicLong)。

**16. CAS的问题**

1）CAS容易造成ABA问题。一个线程a将数值改成了b，接着又改成了a，此时CAS认为是没有变化，其实是已经变化过了，而这个问题的解决方案可以使用版本号标识，每操作一次version加1。在java5中，已经提供了AtomicStampedReference来解决问题。

2） 不能保证代码块的原子性

CAS机制所保证的知识一个变量的原子性操作，而不能保证整个代码块的原子性。比如需要保证3个变量共同进行原子性的更新，就不得不使用synchronized了。

3）CAS造成CPU利用率增加。之前说过了CAS里面是一个循环判断的过程，如果线程一直没有获取到状态，cpu资源会一直被占用。

**17.什么是Future？**

在并发编程中，我们经常用到非阻塞的模型，在之前的多线程的三种实现中，不管是继承thread类还是实现runnable接口，都无法保证获取到之前的执行结果。通过实现Callback接口，并用Future可以来接收多线程的执行结果。

Future表示一个可能还没有完成的异步任务的结果，针对这个结果可以添加Callback以便在任务执行成功或失败后作出相应的操作。

**.什么是AQS**

**AQS是AbustactQueuedSynchronizer的简称，它是一个Java提高的底层同步工具类，用一个int类型的变量表示同步状态，并提供了一系列的CAS操作来管理这个同步状态。**

AQS是一个用来构建锁和同步器的框架，使用AQS能简单且高效地构造出应用广泛的大量的同步器，比如我们提到的ReentrantLock，Semaphore，其他的诸如ReentrantReadWriteLock，SynchronousQueue，FutureTask等等皆是基于AQS的。

**AQS支持两种同步方式：**

**1.独占式**

**2.共享式**

这样方便使用者实现不同类型的同步组件，独占式如ReentrantLock，共享式如Semaphore，CountDownLatch，组合式的如ReentrantReadWriteLock。总之，AQS为使用提供了底层支撑，如何组装实现，使用者可以自由发挥。

**19.ReadWriteLock是什么**

首先明确一下，不是说ReentrantLock不好，只是ReentrantLock某些时候有局限。如果使用ReentrantLock，可能本身是为了防止线程A在写数据、线程B在读数据造成的数据不一致，但这样，如果线程C在读数据、线程D也在读数据，读数据是不会改变数据的，没有必要加锁，但是还是加锁了，降低了程序的性能。

因为这个，才诞生了读写锁ReadWriteLock。ReadWriteLock是一个读写锁接口，ReentrantReadWriteLock是ReadWriteLock接口的一个具体实现，实现了读写的分离，**读锁是共享的，写锁是独占的**，读和读之间不会互斥，读和写、写和读、写和写之间才会互斥，提升了读写的性能。

**20.FutureTask是什么**

这个其实前面有提到过，FutureTask表示一个异步运算的任务。FutureTask里面可以传入一个Callable的具体实现类，可以对这个异步运算的任务的结果进行等待获取、判断是否已经完成、取消任务等操作。当然，由于FutureTask也是Runnable接口的实现类，所以FutureTask也可以放入线程池中。

**21.synchronized和ReentrantLock的区别**

synchronized是和if、else、for、while一样的关键字，ReentrantLock是类，这是二者的本质区别。既然ReentrantLock是类，那么它就提供了比synchronized更多更灵活的特性，可以被继承、可以有方法、可以有各种各样的类变量，ReentrantLock比synchronized的扩展性体现在几点上：

（1）ReentrantLock可以对获取锁的等待时间进行设置，这样就避免了死锁

（2）ReentrantLock可以获取各种锁的信息

（3）ReentrantLock可以灵活地实现多路通知

另外，二者的锁机制其实也是不一样的。ReentrantLock底层调用的是Unsafe的park方法加锁，synchronized操作的应该是对象头中mark word，这点我不能确定。

**22.什么是乐观锁和悲观锁**

（1）乐观锁：就像它的名字一样，对于并发间操作产生的线程安全问题持乐观状态，乐观锁认为竞争不总是会发生，因此它不需要持有锁，将**比较-替换**这两个动作作为一个原子操作尝试去修改内存中的变量，如果失败则表示发生冲突，那么就应该有相应的重试逻辑。

（2）悲观锁：还是像它的名字一样，对于并发间操作产生的线程安全问题持悲观状态，悲观锁认为竞争总是会发生，因此每次对某资源进行操作时，都会持有一个独占的锁，就像synchronized，不管三七二十一，直接上了锁就操作资源了。

**23.线程B怎么知道线程A修改了变量**

* volatile修饰变量
* synchronized修饰修改变量的方法
* wait/notify
* while轮询

**24.synchronized、volatile、CAS比较**

* synchronized是悲观锁，属于抢占式，会引起其他线程阻塞。
* volatile提供多线程共享变量可见性和禁止指令重排序优化。
* CAS是基于冲突检测的乐观锁（非阻塞）

**25.sleep方法和wait方法有什么区别?**

这个问题常问，sleep方法和wait方法都可以用来放弃CPU一定的时间，不同点在于如果线程持有某个对象的监视器，sleep方法不会放弃这个对象的监视器，wait方法会放弃这个对象的监视器

**26.ThreadLocal是什么？有什么用？**

ThreadLocal是一个本地线程副本变量工具类。主要用于将私有线程和该线程存放的副本对象做一个映射，各个线程之间的变量互不干扰，在高并发场景下，可以实现无状态的调用，特别适用于各个线程依赖不通的变量值完成操作的场景。

简单说ThreadLocal就是一种以**空间换时间**的做法，在每个Thread里面维护了一个以开地址法实现的ThreadLocal.ThreadLocalMap，把数据进行隔离，数据不共享，自然就没有线程安全方面的问题了。

**27.为什么wait()方法和notify()/notifyAll()方法要在同步块中被调用**

这是JDK强制的，wait()方法和notify()/notifyAll()方法在调用前都必须先获得对象的锁

**28.多线程同步有哪几种方法？**

**Synchronized关键字，Lock锁实现，分布式锁等。**

**29.线程的调度策略**

线程调度器选择优先级最高的线程运行，但是，如果发生以下情况，就会终止线程的运行：

（1）线程体中调用了yield方法让出了对cpu的占用权利

（2）线程体中调用了sleep方法使线程进入睡眠状态

（3）线程由于IO操作受到阻塞

（4）另外一个更高优先级线程出现

（5）在支持时间片的系统中，该线程的时间片用完

**30.ConcurrentHashMap的并发度是什么**

ConcurrentHashMap的并发度就是segment的大小，默认为16，这意味着最多同时可以有16条线程操作ConcurrentHashMap，这也是ConcurrentHashMap对Hashtable的最大优势，任何情况下，Hashtable能同时有两条线程获取Hashtable中的数据吗？

**31.Linux环境下如何查找哪个线程使用CPU最长**

（1）获取项目的pid，jps或者ps -ef | grep java，这个前面有讲过

（2）top -H -p pid，顺序不能改变

**32.Java死锁以及如何避免？**

**Java中的死锁是一种编程情况，其中两个或多个线程被永久阻塞，Java死锁情况出现至少两个线程和两个或更多资源。**

Java发生死锁的根本原因是：在申请锁时发生了交叉闭环申请。

**死锁的原因**

**1）是多个线程涉及到多个锁，这些锁存在着交叉，所以可能会导致了一个锁依赖的闭环。**

**例如：线程**在获得了锁A并且没有释放的情况下去申请锁B，这时，另一个线程已经获得了锁B，在释放锁B之前又要先获得锁A，因此闭环发生，陷入死锁循环。

**2）默认的锁申请操作是阻塞的**。

所以要避免死锁，就要在一遇到多个对象锁交叉的情况，就要仔细审查这几个对象的类中的所有方法，是否存在着导致锁依赖的环路的可能性。 总之是尽量避免在一个同步方法中调用其它对象的延时方法和同步方法。

**33.怎么唤醒一个阻塞的线程**

如果线程是因为调用了wait()、sleep()或者join()方法而导致的阻塞，可以中断线程，并且通过抛出InterruptedException来唤醒它；如果线程遇到了IO阻塞，无能为力，因为IO是操作系统实现的，Java代码并没有办法直接接触到操作系统。

**34.不可变对象对多线程有什么帮助**

前面有提到过的一个问题，不可变对象保证了对象的内存可见性，对不可变对象的读取不需要进行额外的同步手段，提升了代码执行效率。

**35.什么是多线程的上下文切换**

多线程的上下文切换是指CPU控制权由一个已经正在运行的线程切换到另外一个就绪并等待获取CPU执行权的线程的过程。

**36.如果你提交任务时，线程池队列已满，这时会发生什么**

这里区分一下：

1. 如果使用的是无界队列LinkedBlockingQueue，也就是无界队列的话，没关系，继续添加任务到阻塞队列中等待执行，因为LinkedBlockingQueue可以近乎认为是一个无穷大的队列，可以无限存放任务
2. 如果使用的是有界队列比如ArrayBlockingQueue，任务首先会被添加到ArrayBlockingQueue中，ArrayBlockingQueue满了，会根据maximumPoolSize的值增加线程数量，如果增加了线程数量还是处理不过来，ArrayBlockingQueue继续满，那么则会使用拒绝策略RejectedExecutionHandler处理满了的任务，默认是AbortPolicy

**37.Java中用到的线程调度算法是什么**

抢占式。一个线程用完CPU之后，操作系统会根据线程优先级、线程饥饿情况等数据算出一个总的优先级并分配下一个时间片给某个线程执行。

**38.什么是线程调度器(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。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。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）。

**39.什么是自旋**

很多synchronized里面的代码只是一些很简单的代码，执行时间非常快，此时等待的线程都加锁可能是一种不太值得的操作，因为线程阻塞涉及到用户态和内核态切换的问题。既然synchronized里面的代码执行得非常快，不妨让等待锁的线程不要被阻塞，而是在synchronized的边界做忙循环，这就是自旋。如果做了多次忙循环发现还没有获得锁，再阻塞，这样可能是一种更好的策略。

**40.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

**41.单例模式的线程安全性**

老生常谈的问题了，首先要说的是单例模式的线程安全意味着：**某个类的实例在多线程环境下只会被创建一次出来**。单例模式有很多种的写法，我总结一下：

（1）饿汉式单例模式的写法：线程安全

（2）懒汉式单例模式的写法：非线程安全

（3）双检锁单例模式的写法：线程安全

**42.Semaphore有什么作用**

Semaphore就是一个信号量，它的作用是**限制某段代码块的并发数**。Semaphore有一个构造函数，可以传入一个int型整数n，表示某段代码最多只有n个线程可以访问，如果超出了n，那么请等待，等到某个线程执行完毕这段代码块，下一个线程再进入。由此可以看出如果Semaphore构造函数中传入的int型整数n=1，相当于变成了一个synchronized了。

**43.Executors类是什么？**

Executors为Executor，ExecutorService，ScheduledExecutorService，ThreadFactory和Callable类提供了一些工具方法。

Executors可以用于方便的创建线程池

**44.线程类的构造方法、静态块是被哪个线程调用的**

这是一个非常刁钻和狡猾的问题。请记住：线程类的构造方法、静态块是被new这个线程类所在的线程所调用的，而run方法里面的代码才是被线程自身所调用的。

如果说上面的说法让你感到困惑，那么我举个例子，假设Thread2中new了Thread1，main函数中new了Thread2，那么：

（1）Thread2的构造方法、静态块是main线程调用的，Thread2的run()方法是Thread2自己调用的

（2）Thread1的构造方法、静态块是Thread2调用的，Thread1的run()方法是Thread1自己调用的

**45.同步方法和同步块，哪个是更好的选择**

同步块，这意味着同步块之外的代码是异步执行的，这比同步整个方法更提升代码的效率。请知道一条原则：**同步的范围越小越好**。

**46.Java线程数过多会造成什么异常？**

1)线程的生命周期开销非常高

2)消耗过多的CPU资源

如果可运行的线程数量多于可用处理器的数量，那么有线程将会被闲置。大量空闲的线程会占用许多内存，给垃圾回收器带来压力，而且大量的线程在竞争CPU资源时还将产生其他性能的开销。

3)降低稳定性

JVM在可创建线程的数量上存在一个限制，这个限制值将随着平台的不同而不同，并且承受着多个因素制约，包括JVM的启动参数、Thread构造函数中请求栈的大小，以及底层操作系统对线程的限制等。如果破坏了这些限制，那么可能抛出OutOfMemoryError异常。

**Mysql中有哪几种锁？**

1.表级锁：开销小，加锁快；不会出现死锁；锁定粒度大，发生锁冲突的概率最高，并发度最低。

2.行级锁：开销大，加锁慢；会出现死锁；锁定粒度最小，发生锁冲突的概率最低，并发度也最高。

3. 页面锁：开销和加锁时间界于表锁和行锁之间；会出现死锁；锁定粒度界于表锁和行锁之间，并发度一般。

**Mysql中有哪些不同的表格？**

共有5种类型的表格：

1. MyISAM
2. Heap
3. Merge
4. INNODB
5. ISAM

**简述在MySQL数据库中MyISAM和InnoDB的区别**

**MyISAM：**

1. 不支持事务，但是每次查询都是原子的；
2. 支持表级锁，即每次操作是对整个表加锁；
3. 存储表的总行数；
4. 一个MYISAM表有三个文件：索引文件、表结构文件、数据文件；
5. 采用菲聚集索引，索引文件的数据域存储指向数据文件的指针。辅索引与主索引基本一致，但是辅索引不用保证唯一性。

**InnoDb：**

1. 支持ACID的事务，支持事务的四种隔离级别；
2. 支持行级锁及外键约束：因此可以支持写并发；
3. 不存储总行数；
4. 一个InnoDb引擎存储在一个文件空间（共享表空间，表大小不受操作系统控制，一个表可能分布在多个文件里），也有可能为多个（设置为独立表空，表大小受操作系统文件大小限制，一般为2G），受操作系统文件大小的限制；
5. 主键索引采用聚集索引（索引的数据域存储数据文件本身），辅索引的数据域存储主键的值；因此从辅索引查找数据，需要先通过辅索引找到主键值，再访问辅索引；最好使用自增主键，防止插入数据时，为维持B+树结构，文件的大调整。

**Mysql中InnoDB支持的四种事务隔离级别名称，以及逐级之间的区别？**

SQL标准定义的四个隔离级别为：

1. read uncommited ：读到未提交数据
2. read committed：脏读，不可重复读
3. repeatable read：可重读
4. serializable ：串行事物

**CHAR和VARCHAR的区别？**

1.CHAR和VARCHAR类型在存储和检索方面有所不同

2.CHAR列长度固定为创建表时声明的长度，长度值范围是1到255

当CHAR值被存储时，它们被用空格填充到特定长度，检索CHAR值时需删除尾随空格。

**主键和候选键有什么区别？**

表格的每一行都由主键唯一标识,一个表只有一个主键。

主键也是候选键。按照惯例，候选键可以被指定为主键，并且可以用于任何外键引用。

**你怎么看到为表格定义的所有索引？**

索引是通过以下方式为表格定义的：

SHOW INDEX FROM

<tablename>;

**LIKE声明中的％和\_是什么意思？**

％对应于0个或更多字符，\_只是LIKE语句中的一个字符。

**如何在Unix和Mysql时间戳之间进行转换？**

UNIX\_TIMESTAMP是从Mysql时间戳转换为Unix时间戳的命令

FROM\_UNIXTIME是从Unix时间戳转换为Mysql时间戳的命令

**列对比运算符是什么？**

在SELECT语句的列比较中使用=，<>，<=，<，> =，>，<<，>>，<=>，AND，OR或LIKE运算符。

**BLOB和TEXT有什么区别？**

BLOB是一个二进制对象，可以容纳可变数量的数据。TEXT是一个不区分大小写的BLOB。

BLOB和TEXT类型之间的唯一区别在于对BLOB值进行排序和比较时区分大小写，对TEXT值不区分大小写。

**mysql\_fetch\_array和mysql\_fetch\_object的区别是什么？**

以下是mysql\_fetch\_array和mysql\_fetch\_object的区别：

mysql\_fetch\_array（） – 将结果行作为关联数组或来自数据库的常规数组返回。

mysql\_fetch\_object – 从数据库返回结果行作为对象。

**Mysql如何优化DISTINCT？**

DISTINCT在所有列上转换为GROUP BY，并与ORDER BY子句结合使用。

**如何显示前50行？**

在Mysql中，使用以下代码查询显示前50行：

SELECT\*FROM

LIMIT 0,50;

**可以使用多少列创建索引？**

任何标准表最多可以创建16个索引列。

**如果一个表有一列定义为TIMESTAMP，将发生什么？**

每当行被更改时，时间戳字段将获取当前时间戳。

**列设置为AUTO INCREMENT时，如果在表中达到最大值，会发生什么情况？**

它会停止递增，任何进一步的插入都将产生错误，因为密钥已被使用。

**怎样才能找出最后一次插入时分配了哪个自动增量？**

LAST\_INSERT\_ID将返回由Auto\_increment分配的最后一个值，并且不需要指定表名称。

**NOW（）和CURRENT\_DATE（）有什么区别？**

NOW（）命令用于显示当前年份，月份，日期，小时，分钟和秒。

CURRENT\_DATE（）仅显示当前年份，月份和日期。

**什么是非标准字符串类型？**

1. TINYTEXT
2. TEXT
3. MEDIUMTEXT
4. LONGTEXT

**什么是通用SQL函数？**

1. CONCAT(A, B) – 连接两个字符串值以创建单个字符串输出。通常用于将两个或多个字段合并为一个字段。
2. FORMAT(X, D)- 格式化数字X到D有效数字。
3. CURRDATE(), CURRTIME()- 返回当前日期或时间。
4. NOW（） – 将当前日期和时间作为一个值返回。
5. MONTH（），DAY（），YEAR（），WEEK（），WEEKDAY（） – 从日期值中提取给定数据。
6. HOUR（），MINUTE（），SECOND（） – 从时间值中提取给定数据。
7. DATEDIFF（A，B） – 确定两个日期之间的差异，通常用于计算年龄
8. SUBTIMES（A，B） – 确定两次之间的差异。
9. FROMDAYS（INT） – 将整数天数转换为日期值。

**MYSQL支持事务吗？**

在缺省模式下，MYSQL是autocommit模式的，所有的数据库更新操作都会即时提交，所以在缺省情况下，mysql是不支持事务的。

但是如果你的MYSQL表类型是使用InnoDB Tables 或 BDB tables的话，你的MYSQL就可以使用事务处理,使用SET

AUTOCOMMIT=0就可以使MYSQL允许在非autocommit模式，在非autocommit模式下，你必须使用COMMIT来提交你的更改，或者用ROLLBACK来回滚你的更改。

**mysql里记录货币用什么字段类型好**

NUMERIC和DECIMAL类型被Mysql实现为同样的类型，这在SQL92标准允许。他们被用于保存值，该值的准确精度是极其重要的值，例如与金钱有关的数据。当声明一个类是这些类型之一时，精度和规模的能被(并且通常是)指定。

例如：

salary DECIMAL(9,2)

在这个例子中，9(precision)代表将被用于存储值的总的小数位数，而2(scale)代表将被用于存储小数点后的位数。

因此，在这种情况下，能被存储在salary列中的值的范围是从-9999999.99到9999999.99。

**mysql有关权限的表都有哪几个？**

Mysql服务器通过权限表来控制用户对数据库的访问，权限表存放在mysql数据库里，由mysql\_install\_db脚本初始化。这些权限表分别user，db，table\_priv，columns\_priv和host。

**列的字符串类型可以是什么？**

字符串类型是：

1. SET
2. BLOB
3. ENUM
4. CHAR
5. TEXT

**MySQL数据库作发布系统的存储，数据量增大的情况，怎么优化？**

1. 设计良好的数据库结构，允许部分数据冗余，尽量避免join查询，提高效率。

2. 选择合适的表字段数据类型和存储引擎，适当的添加索引。

3. mysql库主从读写分离。

4. 找规律分表，减少单表中的数据量提高查询速度。

5.添加缓存机制，比如memcached，apc等。

6. 不经常改动的页面，生成静态页面。

7. 书写高效率的SQL。比如 SELECT \* FROM TABEL 改为 SELECT field\_1, field\_2, field\_3 FROM TABLE.

**锁的优化策略**

1. 读写分离

2. 分段加锁

3. 减少锁持有的时间

4. 多个线程尽量以相同的顺序去获取资源

不能将锁的粒度过于细化，不然可能会出现线程的加锁和释放次数过多，反而效率不如一次加一把大锁。

**索引的底层实现原理和优化**

B+树，经过优化的B+树

主要是在所有的叶子结点中增加了指向下一个叶子节点的指针，因此InnoDB建议为大部分表使用默认自增的主键作为主索引。

**什么情况下设置了索引但无法使用**

1.以“%”开头的LIKE语句，模糊匹配

2. OR语句前后没有同时使用索引

3. 数据类型出现隐式转化（如varchar不加单引号的话可能会自动转换为int型）

**实践中如何优化MySQL**

最好是按照以下顺序优化：

1.SQL语句及索引的优化

2. 数据库表结构的优化

3.系统配置的优化

4.硬件的优化

**详细可以查看:** [阿里P8架构师谈：MySQL慢查询优化、索引优化、以及表等优化总结](http://mp.weixin.qq.com/s?__biz=MzI0NzU3MTk2MQ==&mid=2247485396&idx=1&sn=f14d646235377f3db8b69f3b44d6b549&chksm=e9acb342dedb3a54cd02e317b2969a2f951d087b59d47adfce7d4e4e8b456a0d6830369f5d43&scene=21#wechat_redirect)

**优化数据库的方法**

1. 选取最适用的字段属性，尽可能减少定义字段宽度，尽量把字段设置NOTNULL，例如’省份’、’性别’最好适用ENUM
2. 使用连接(JOIN)来代替子查询
3. 适用联合(UNION)来代替手动创建的临时表
4. 事务处理
5. 锁定表、优化事务处理
6. 适用外键，优化锁定表
7. 建立索引
8. 优化查询语句

**简单描述mysql中，索引，主键，唯一索引，联合索引的区别，对数据库的性能有什么影响（从读写两方面）**

索引是一种特殊的文件(InnoDB数据表上的索引是表空间的一个组成部分)，它们包含着对数据表里所有记录的引用指针。

普通索引(由关键字KEY或INDEX定义的索引)的唯一任务是加快对数据的访问速度。

普通索引允许被索引的数据列包含重复的值。如果能确定某个数据列将只包含彼此各不相同的值，在为这个数据列创建索引的时候就应该用关键字UNIQUE把它定义为一个唯一索引。也就是说，唯一索引可以保证数据记录的唯一性。

主键，是一种特殊的唯一索引，在一张表中只能定义一个主键索引，主键用于唯一标识一条记录，使用关键字 PRIMARY KEY 来创建。

索引可以覆盖多个数据列，如像INDEX(columnA, columnB)索引，这就是联合索引。

索引可以极大的提高数据的查询速度，但是会降低插入、删除、更新表的速度，因为在执行这些写操作时，还要操作索引文件。

**数据库中的事务是什么?**

事务（transaction）是作为一个单元的一组有序的数据库操作。如果组中的所有操作都成功，则认为事务成功，即使只有一个操作失败，事务也不成功。如果所有操作完成，事务则提交，其修改将作用于所有其他数据库进程。如果一个操作失败，则事务将回滚，该事务所有操作的影响都将取消。

**事务特性：**

（1）原子性：即不可分割性，事务要么全部被执行，要么就全部不被执行。

（2）一致性或可串性。事务的执行使得数据库从一种正确状态转换成另一种正确状态

（3）隔离性。在事务正确提交之前，不允许把该事务对数据的任何改变提供给任何其他事务，

（4） 持久性。事务正确提交后，其结果将永久保存在数据库中，即使在事务提交后有了其他故障，事务的处理结果也会得到保存。

或者这样理解：

事务就是被绑定在一起作为一个逻辑工作单元的SQL语句分组，如果任何一个语句操作失败那么整个操作就被失败，以后操作就会回滚到操作前状态，或者是上有个节点。为了确保要么执行，要么不执行，就可以使用事务。要将有组语句作为事务考虑，就需要通过ACID测试，即原子性，一致性，隔离性和持久性。

**SQL注入漏洞产生的原因？如何防止？**

SQL注入产生的原因：程序开发过程中不注意规范书写sql语句和对特殊字符进行过滤，导致客户端可以通过全局变量POST和GET提交一些sql语句正常执行。

防止SQL注入的方式：

开启配置文件中的magic\_quotes\_gpc 和 magic\_quotes\_runtime设置

执行sql语句时使用addslashes进行sql语句转换

Sql语句书写尽量不要省略双引号和单引号。

过滤掉sql语句中的一些关键词：update、insert、delete、select、 \* 。

提高数据库表和字段的命名技巧，对一些重要的字段根据程序的特点命名，取不易被猜到的。

**为表中得字段选择合适得数据类型**

字段类型优先级: 整形>date,time>enum,char>varchar>blob,text

优先考虑数字类型，其次是日期或者二进制类型，最后是字符串类型，同级别得数据类型，应该优先选择占用空间小的数据类型

**存储时期**

Datatime:以 YYYY-MM-DD HH:MM:SS 格式存储时期时间，精确到秒，占用8个字节得存储空间，datatime类型与时区无关

Timestamp:以时间戳格式存储，占用4个字节，范围小1970-1-1到2038-1-19，显示依赖于所指定得时区，默认在第一个列行的数据修改时可以自动得修改timestamp列得值

Date:（生日）占用得字节数比使用字符串.datatime.int储存要少，使用date只需要3个字节，存储日期月份，还可以利用日期时间函数进行日期间得计算

Time:存储时间部分得数据

注意:不要使用字符串类型来存储日期时间数据（通常比字符串占用得储存空间小，在进行查找过滤可以利用日期得函数）

使用int存储日期时间不如使用timestamp类型

**对于关系型数据库而言，索引是相当重要的概念，请回答有关索引的几个问题：**

**1.索引的目的是什么？**

快速访问数据表中的特定信息，提高检索速度

创建唯一性索引，保证数据库表中每一行数据的唯一性。

加速表和表之间的连接

使用分组和排序子句进行数据检索时，可以显著减少查询中分组和排序的时间

**2.索引对数据库系统的负面影响是什么？**

负面影响：

创建索引和维护索引需要耗费时间，这个时间随着数据量的增加而增加；索引需要占用物理空间，不光是表需要占用数据空间，每个索引也需要占用物理空间；当对表进行增、删、改、的时候索引也要动态维护，这样就降低了数据的维护速度。

**3.为数据表建立索引的原则有哪些？**

在最频繁使用的、用以缩小查询范围的字段上建立索引。

在频繁使用的、需要排序的字段上建立索引

**4.什么情况下不宜建立索引？**

对于查询中很少涉及的列或者重复值比较多的列，不宜建立索引。

对于一些特殊的数据类型，不宜建立索引，比如文本字段（text）等

**解释MySQL外连接、内连接与自连接的区别**

先说什么是交叉连接: 交叉连接又叫笛卡尔积，它是指不使用任何条件，直接将一个表的所有记录和另一个表中的所有记录一一匹配。

内连接 则是只有条件的交叉连接，根据某个条件筛选出符合条件的记录，不符合条件的记录不会出现在结果集中，即内连接只连接匹配的行。

外连接 其结果集中不仅包含符合连接条件的行，而且还会包括左表、右表或两个表中的所有数据行，这三种情况依次称之为左外连接，右外连接，和全外连接。

左外连接，也称左连接，左表为主表，左表中的所有记录都会出现在结果集中，对于那些在右表中并没有匹配的记录，仍然要显示，右边对应的那些字段值以NULL来填充。右外连接，也称右连接，右表为主表，右表中的所有记录都会出现在结果集中。左连接和右连接可以互换，MySQL目前还不支持全外连接。

**Myql中的事务回滚机制概述**

事务是用户定义的一个数据库操作序列，这些操作要么全做要么全不做，是一个不可分割的工作单位，事务回滚是指将该事务已经完成的对数据库的更新操作撤销。

要同时修改数据库中两个不同表时，如果它们不是一个事务的话，当第一个表修改完，可能第二个表修改过程中出现了异常而没能修改，此时就只有第二个表依旧是未修改之前的状态，而第一个表已经被修改完毕。而当你把它们设定为一个事务的时候，当第一个表修改完，第二表修改出现异常而没能修改，第一个表和第二个表都要回到未修改的状态，这就是所谓的事务回滚

**SQL语言包括哪几部分？每部分都有哪些操作关键字？**

SQL语言包括数据定义(DDL)、数据操纵(DML),数据控制(DCL)和数据查询（DQL）四个部分。

数据定义：Create Table,Alter Table,Drop Table, Craete/Drop Index等

数据操纵：Select ,insert,update,delete,

数据控制：grant,revoke

数据查询：select

**完整性约束包括哪些？**

数据完整性(Data Integrity)是指数据的精确(Accuracy)和可靠性(Reliability)。

**分为以下四类：**

1) 实体完整性：规定表的每一行在表中是惟一的实体。

2) 域完整性：是指表中的列必须满足某种特定的数据类型约束，其中约束又包括取值范围、精度等规定。

3) 参照完整性：是指两个表的主关键字和外关键字的数据应一致，保证了表之间的数据的一致性，防止了数据丢失或无意义的数据在数据库中扩散。

4) 用户定义的完整性：不同的关系数据库系统根据其应用环境的不同，往往还需要一些特殊的约束条件。用户定义的完整性即是针对某个特定关系数据库的约束条件，它反映某一具体应用必须满足的语义要求。

与表有关的约束：包括列约束(NOT NULL（非空约束）)和表约束(PRIMARY KEY、foreign key、check、UNIQUE) 。

**什么是锁？**

答：数据库是一个多用户使用的共享资源。当多个用户并发地存取数据时，在数据库中就会产生多个事务同时存取同一数据的情况。若对并发操作不加控制就可能会读取和存储不正确的数据，破坏数据库的一致性。

加锁是实现数据库并发控制的一个非常重要的技术。当事务在对某个数据对象进行操作前，先向系统发出请求，对其加锁。加锁后事务就对该数据对象有了一定的控制，在该事务释放锁之前，其他的事务不能对此数据对象进行更新操作。

**基本锁类型：锁包括行级锁和表级锁**

**什么叫视图？游标是什么？**

答：视图是一种虚拟的表，具有和物理表相同的功能。可以对视图进行增，改，查，操作，视图通常是有一个表或者多个表的行或列的子集。对视图的修改不影响基本表。它使得我们获取数据更容易，相比多表查询。

游标：是对查询出来的结果集作为一个单元来有效的处理。游标可以定在该单元中的特定行，从结果集的当前行检索一行或多行。可以对结果集当前行做修改。一般不使用游标，但是需要逐条处理数据的时候，游标显得十分重要。

**什么是存储过程？用什么来调用？**

存储过程是一个预编译的SQL语句，优点是允许模块化的设计，就是说只需创建一次，以后在该程序中就可以调用多次。如果某次操作需要执行多次SQL，使用存储过程比单纯SQL语句执行要快。可以用一个命令对象来调用存储过程。

**如何通俗地理解三个范式？**

答：第一范式：1NF是对属性的原子性约束，要求属性具有原子性，不可再分解；

第二范式：2NF是对记录的惟一性约束，要求记录有惟一标识，即实体的惟一性；

第三范式：3NF是对字段冗余性的约束，即任何字段不能由其他字段派生出来，它要求字段没有冗余。。

范式化设计优缺点:

**优点:**

可以尽量得减少数据冗余，使得更新快，体积小

缺点:对于查询需要多个表进行关联，减少写得效率增加读得效率，更难进行索引优化

**反范式化:**

优点:可以减少表得关联，可以更好得进行索引优化

缺点:数据冗余以及数据异常，数据得修改需要更多的成本

**什么是基本表？什么是视图？**

答：基本表是本身独立存在的表，在 SQL 中一个关系就对应一个表。 视图是从一个或几个基本表导出的表。视图本身不独立存储在数据库中，是一个虚表

**试述视图的优点？**

答：(1) 视图能够简化用户的操作 (2) 视图使用户能以多种角度看待同一数据； (3) 视图为数据库提供了一定程度的逻辑独立性； (4) 视图能够对机密数据提供安全保护。

**NULL是什么意思**

答：NULL这个值表示UNKNOWN(未知):它不表示“”(空字符串)。对NULL这个值的任何比较都会生产一个NULL值。您不能把任何值与一个 NULL值进行比较，并在逻辑上希望获得一个答案。

使用IS NULL来进行NULL判断

**主键、外键和索引的区别？**

主键、外键和索引的区别

**定义：**

主键–唯一标识一条记录，不能有重复的，不允许为空

外键–表的外键是另一表的主键, 外键可以有重复的, 可以是空值

索引–该字段没有重复值，但可以有一个空值

**作用：**

主键–用来保证数据完整性

外键–用来和其他表建立联系用的

索引–是提高查询排序的速度

**个数：**

主键–主键只能有一个

外键–一个表可以有多个外键

索引–一个表可以有多个唯一索引

**你可以用什么来确保表格里的字段只接受特定范围里的值?**

答：Check限制，它在数据库表格里被定义，用来限制输入该列的值。

触发器也可以被用来限制数据库表格里的字段能够接受的值，但是这种办法要求触发器在表格里被定义，这可能会在某些情况下影响到性能。

**说说对SQL语句优化有哪些方法？（选择几条）**

（1）Where子句中：where表之间的连接必须写在其他Where条件之前，那些可以过滤掉最大数量记录的条件必须写在Where子句的末尾.HAVING最后。

（2）用EXISTS替代IN、用NOT EXISTS替代NOT IN。

（3） 避免在索引列上使用计算

（4）避免在索引列上使用IS NULL和IS NOT NULL

（5）对查询进行优化，应尽量避免全表扫描，首先应考虑在 where 及 order by 涉及的列上建立索引。

（6）应尽量避免在 where 子句中对字段进行 null 值判断，否则将导致引擎放弃使用索引而进行全表扫描

（7）应尽量避免在 where 子句中对字段进行表达式操作，这将导致引擎放弃使用索引而进行全表扫描。

**Redis支持哪几种数据类型？**

支持多种类型的数据结构

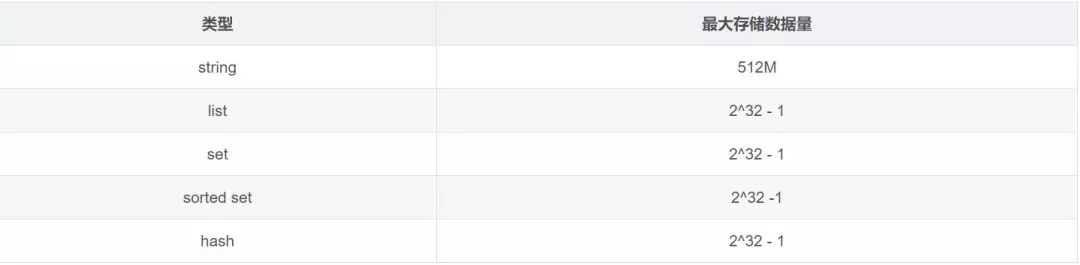
1.string：最基本的数据类型，二进制安全的字符串，最大512M。

2.list：按照添加顺序保持顺序的字符串列表。

3.set：无序的字符串集合，不存在重复的元素。

4.sorted set：已排序的字符串集合。

5.hash：key-value对的一种集合。



**Redis主要有哪些功能？**

**1.哨兵（Sentinel）和复制（Replication）**

Redis服务器毫无征兆的罢工是个麻烦事，如何保证备份的机器是原始服务器的完整备份呢？这时候就需要哨兵和复制。

哨兵Sentinel可以管理多个Redis服务器，它提供了监控，提醒以及自动的故障转移的功能，Replication则是负责让一个Redis服务器可以配备多个备份的服务器。

Redis也是利用这两个功能来保证Redis的高可用的。

**2.事务**

很多情况下我们需要一次执行不止一个命令，而且需要其同时成功或者失败。redis对事务的支持也是源自于这部分需求，即支持一次性按顺序执行多个命令的能力，并保证其原子性。

**3.LUA脚本**

在事务的基础上，如果我们需要在服务端一次性的执行更复杂的操作（包含一些逻辑判断），则lua就可以排上用场了。

**4.持久化**

redis的持久化指的是redis会把内存的中的数据写入到硬盘中，在redis重新启动的时候加载这些数据，从而最大限度的降低缓存丢失带来的影响。

**5.集群（Cluster）**

单台服务器资源的总是有上限的，CPU资源和IO资源我们可以通过主从复制，进行读写分离，把一部分CPU和IO的压力转移到从服务器上，这也有点类似mysql数据库的主从同步。

在Redis官方的分布式方案出来之前，有twemproxy和codis两种方案，这两个方案总体上来说都是依赖proxy来进行分布式的，**下面的内容有具体集群方案详解。**

**Redis是单进程单线程的？**

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。

**Redis为什么是单线程的？**

多线程处理会涉及到锁，而且多线程处理会涉及到线程切换而消耗CPU。因为CPU不是Redis的瓶颈，Redis的瓶颈最有可能是机器内存或者网络带宽。单线程无法发挥多核CPU性能，不过可以通过在单机开多个Redis实例来解决。

**其它开源软件采用的模型**

Nginx：多进程单线程模型

Memcached：单进程多线程模型

**使用Redis的优势？**

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O(1)

2. 支持丰富数据类型，支持string，list，set，sorted set，hash

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

**Redis单点吞吐量**

单点TPS达到8万/秒，QPS达到10万/秒，补充下TPS和QPS的概念

**1.QPS: 应用系统每秒钟最大能接受的用户访问量**

每秒钟处理完请求的次数，注意这里是处理完，具体是指发出请求到服务器处理完成功返回结果。可以理解在server中有个counter，每处理一个请求加1，1秒后counter=QPS。

**2.TPS：** **每秒钟最大能处理的请求数**

每秒钟处理完的事务次数，一个应用系统1s能完成多少事务处理，一个事务在分布式处理中，可能会对应多个请求，对于衡量单个接口服务的处理能力，用QPS比较合理。

**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型

2.Redis的速度比memcached快很多

3.Redis可以持久化其数据

4.Redis支持数据的备份，即master-slave模式的数据备份。

**Redis有哪几种数据淘汰策略？**

在Redis中，允许用户设置最大使用内存大小server.maxmemory，当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。

1.volatile-lru:从已设置过期的数据集中挑选最近最少使用的淘汰

2.volatile-ttr:从已设置过期的数据集中挑选将要过期的数据淘汰

3.volatile-random:从已设置过期的数据集中任意挑选数据淘汰

4.allkeys-lru:从数据集中挑选最近最少使用的数据淘汰

5.allkeys-random:从数据集中任意挑选数据淘汰

6.noenviction:禁止淘汰数据

redis淘汰数据时还会同步到aof

**Redis集群方案应该怎么做？都有哪些方案？**

1.twemproxy

2.codis，目前用的最多的集群方案，基本和twemproxy一致的效果，但它支持在 节点数量改变情况下，旧节点数据可恢复到新hash节点。

3.Redis cluster3.0自带的集，特点在于他的分布式算法不是一致性hash，而是hash槽的概念，以及自身支持节点设置从节点。

具体请查看：[高并发架构系列：详解Redis的存储类型、集群架构、以及应用场景](http://mp.weixin.qq.com/s?__biz=MzI0NzU3MTk2MQ==&mid=2247485838&idx=1&sn=082614321b6b713134a7e2134f844aaf&chksm=e9acbd18dedb340e9885a7e62da54f4dbe419942a816411ebfbfe2654520d12aae2792d8a0bf&scene=21#wechat_redirect)

**Redis读写分离模型**

通过增加Slave DB的数量，读的性能可以线性增长。为了避免Master DB的单点故障，集群一般都会采用两台Master DB做双机热备，所以整个集群的读和写的可用性都非常高。

读写分离架构的缺陷在于，不管是Master还是Slave，每个节点都必须保存完整的数据，如果在数据量很大的情况下，集群的扩展能力还是受限于单个节点的存储能力，而且对于Write-intensive类型的应用，读写分离架构并不适合。

**Redis数据分片模型**

为了解决读写分离模型的缺陷，可以将数据分片模型应用进来。

可以将每个节点看成都是独立的master，然后通过业务实现数据分片。

结合上面两种模型，可以将每个master设计成由一个master和多个slave组成的模型。

**Redis提供了哪几种持久化方式？**

RDB持久化方式能够在指定的时间间隔能对你的数据进行快照存储

AOF持久化方式记录每次对服务器写的操作,当服务器重启的时候会重新执行这些命令来恢复原始的数据,AOF命令以Redis协议追加保存每次写的操作到文件末尾.Redis还能对AOF文件进行后台重写,使得AOF文件的体积不至于过大.

如果你只希望你的数据在服务器运行的时候存在,你也可以不使用任何持久化方式.

你也可以同时开启两种持久化方式, 在这种情况下, 当Redis重启的时候会优先载入AOF文件来恢复原始的数据,因为在通常情况下AOF文件保存的数据集要比RDB文件保存的数据集要完整.

最重要的事情是了解RDB和AOF持久化方式的不同,让我们以RDB持久化方式开始。

**如何选择合适的持久化方式？**

1. Redis主要提供了两种持久化机制：**RDB和AOF**

**2.RDB**

默认开启，会按照配置的指定时间将内存中的数据快照到磁盘中，创建一个dump.rdb文件，Redis启动时再恢复到内存中。

Redis会单独创建fork()一个子进程，将当前父进程的数据库数据复制到子进程的内存中，然后由子进程写入到临时文件中，持久化的过程结束了，再用这个临时文件替换上次的快照文件，然后子进程退出，内存释放。

需要注意的是，每次快照持久化都会将主进程的数据库数据复制一遍，导致内存开销加倍，若此时内存不足，则会阻塞服务器运行，直到复制结束释放内存；都会将内存数据完整写入磁盘一次，所以如果数据量大的话，而且写操作频繁，必然会引起大量的磁盘I/O操作，严重影响性能，并且最后一次持久化后的数据可能会丢失；

**3.AOF**

以日志的形式记录每个写操作（读操作不记录），只需追加文件但不可以改写文件，Redis启动时会根据日志从头到尾全部执行一遍以完成数据的恢复工作。包括flushDB也会执行。

主要有两种方式触发：有写操作就写、每秒定时写（也会丢数据）。

因为AOF采用追加的方式，所以文件会越来越大，针对这个问题，新增了重写机制，就是当日志文件大到一定程度的时候，会fork出一条新进程来遍历进程内存中的数据，每条记录对应一条set语句，写到临时文件中，然后再替换到旧的日志文件（类似rdb的操作方式）。默认触发是当aof文件大小是上次重写后大小的一倍且文件大于64M时触发。

当两种方式同时开启时，数据恢复Redis会优先选择AOF恢复。一般情况下，只要使用默认开启的RDB即可，因为相对于AOF，RDB便于进行数据库备份，并且恢复数据集的速度也要快很多。

开启持久化缓存机制，对性能会有一定的影响，特别是当设置的内存满了的时候，更是下降到几百reqs/s。所以如果只是用来做缓存的话，可以关掉持久化。

**Redis常见性能问题和解决方案？**

(1) Master最好不要做任何持久化工作，如RDB内存快照和AOF日志文件

(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次

(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内

(4) 尽量避免在压力很大的主库上增加从库

(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。

**Redis支持的Java客户端都有哪些？官方推荐用哪个？**

Redisson、Jedis、lettuce等等，官方推荐使用Redisson。

**Redis哈希槽的概念？**

Redis集群没有使用一致性hash,而是引入了哈希槽的概念，当需要在 Redis 集群中放置一个 key-value 时，根据 CRC16(key) mod 16384的值，决定将一个key放到哪个桶中。

**Redis集群最大节点个数是多少？**

Redis集群预分好16384个桶(哈希槽)

**Redis集群的主从复制模型是怎样的？**

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.

**Redis集群会有写操作丢失吗？为什么？**

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。

**Redis集群之间是如何复制的？**

异步复制

**Redis如何做内存优化？**

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.

**Redis回收进程如何工作的？**

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。

**Redis回收使用的是什么算法？**

LRU算法

**Redis有哪些适合的场景？**

1）Session共享(单点登录)

2）页面缓存

3）队列

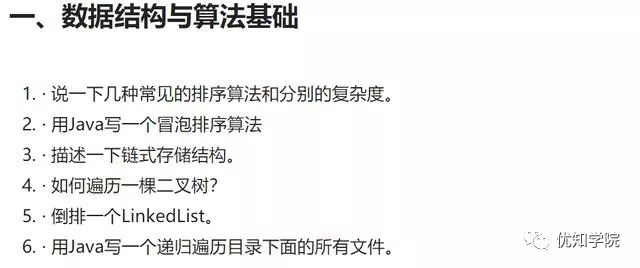
4）排行榜/计数器

5）发布/订阅

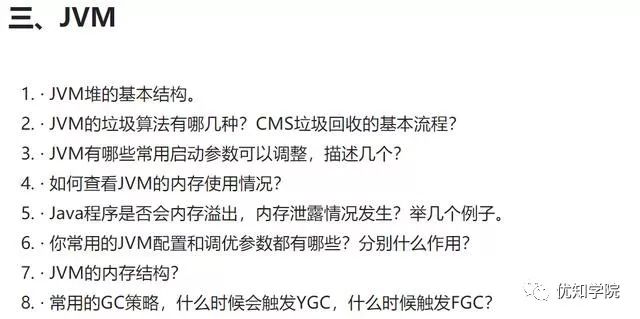
**BAT技术面试范围**

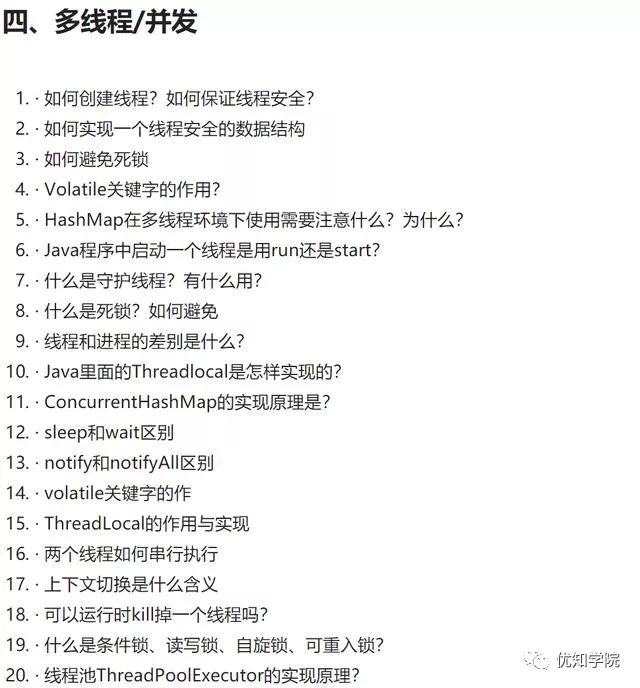
1. 数据结构与算法：最常见的各种排序，最好能手写
2. Java高级：JVM内存结构、垃圾回收器、回收算法、GC、并发编程相关(多线程、线程池等)、NIO/BIO、各种集合类的比较优劣势(底层数据结构也要掌握，特别是扩容等)等。
3. 性能优化、设计模式、UML的掌握
4. Spring框架：重点掌握(BAT每次必问)
5. 分布式相关：Redis缓存、一致Hash算法、分布式存储、负载均衡等。
6. 微服务以及Docker容器等。

**最全阿里面试题目**

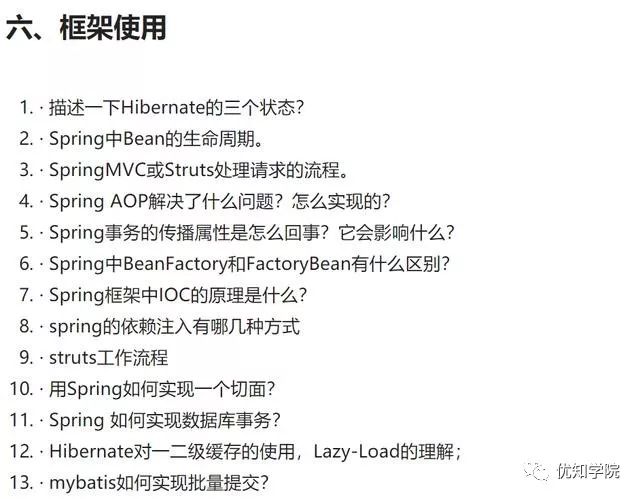


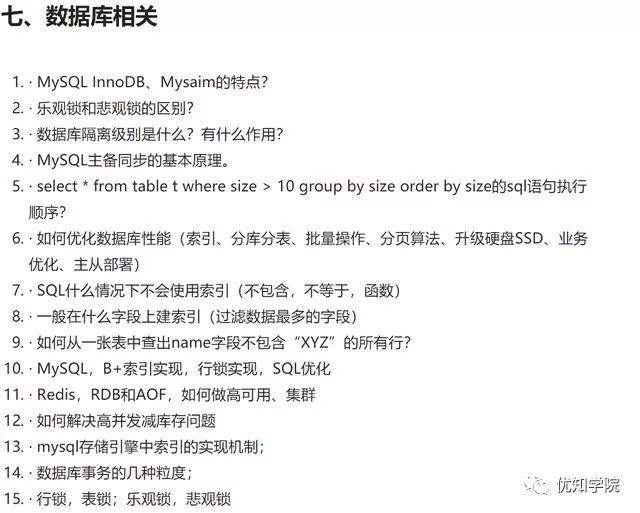


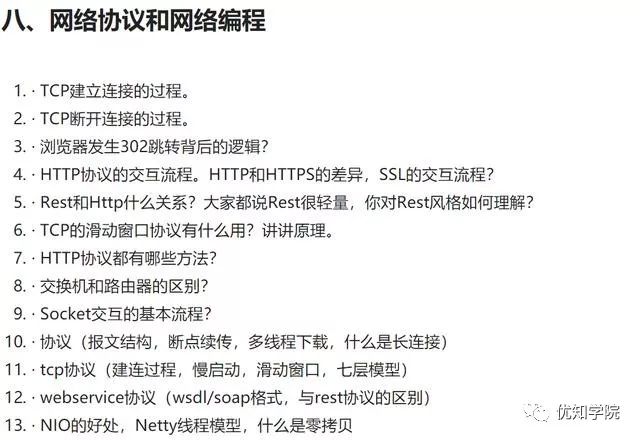


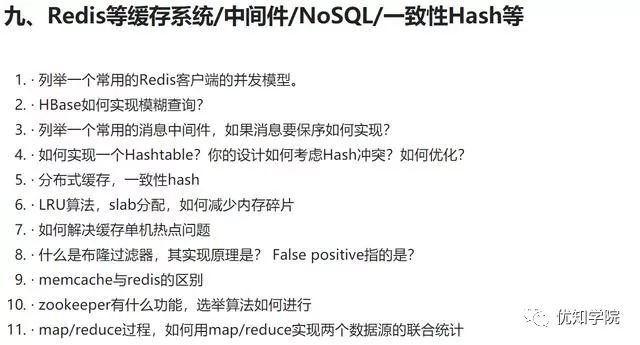


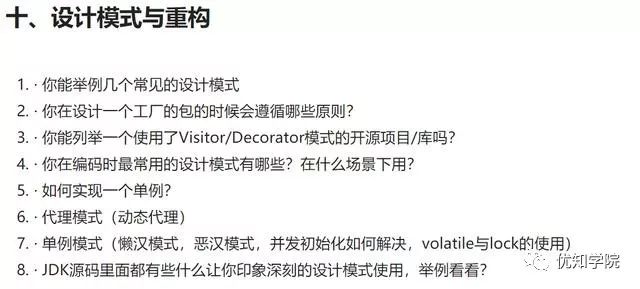


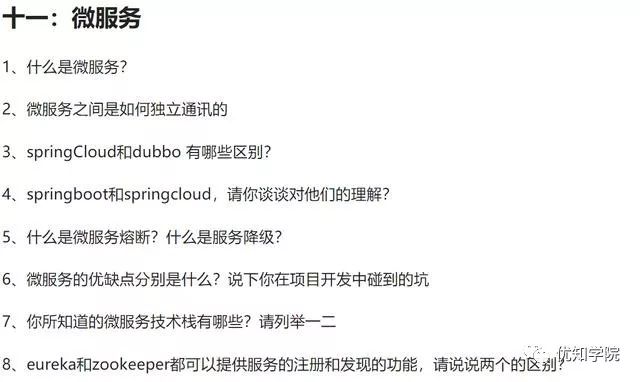












**阿里面试总结**

阿里的面试特别喜欢面试**技术原理**，特别是

1. 多线程
2. NIO
3. 异步消息框架
4. 分布式相关的缓存算法等
5. JVM的加载过程和原理
6. 回收算法
7. 以及具体使用过的框架，会问部分参数检验你是否熟用
8. 第一面能通过，后续被录用的可能性就比较高了，第一轮非常重要，建议系统性的学习面试题目！

2 讲一下ArrayList和linkedlist的区别，ArrayList的扩容方式，扩容时机。

3 hashmap的实现，以及hashmap扩容底层实现。

4 NIO了解么，讲一下和BIO的区别，AIO呢。阻塞，非阻塞，异步。具体。

5 你说了解分布式服务，那么你怎么理解分布式服务。

6 你说了解Tomcat的基本原理，了解的是哪一部分，基本架构，connector和container

7 你在项目中怎么用到并发的。

8 docker和虚拟机讲一下。

9 高并发情况下，如何使用线程池

10 并发juc了解么，有哪些线程安全的list。

2 说一下Spring源码把，它的架构，流程。

3 Spring的bean如果要在实例化过程中修改其某一个成员变量，应该怎么做呢。不通过构造方法，并且AOP也并不能实现。

4 Tomcat的类加载器了解么，回答不了解只了解Java的类加载器。

5 自定义类加载器怎么实现，其中哪个方法走双亲委派模型，哪个不走，不走的话怎么加载类（实现findclass方法，一般用defineclass加载外部类），如何才能不走双亲委派。（重写loadclass方法）

6 布隆过滤器了解么，讲了ip地址过滤的布隆过滤器实现。

7 听说你项目用过docker，讲一下docker的实现原理，说了虚拟机一般要对内核进行虚拟化，docker则用cgroup和namespace分别进行硬件和命名空间的隔离。

8 dubbo的基本架构，几个组件说一下

9 dubbo的负载均衡怎么做，讲一下具体代码实现。

10 参与项目的挑战在哪里，有哪些收获。

1 阿里的dubbo，rocketmq的事务消息的架构设计

2 mycat实现分表分库方案实现

3 消息队列kafka和rabbitmq等

4 分库分表，分片规则hash和取余数。

5 数据库万级变成亿级，怎么处理

6 双11秒杀的设计思路

1. 常见集合类的区别和适用场景
2. 并发容器了解哪些？
3. 如何判断链表是否有环
4. concurrentHashMap如何实现
5. 集群服务器 如何application 共享
6. JAVA网络编程中：BIO、NIO、AIO的区别和联系
7. jvm内存模型jmm 知道的全讲讲
8. JAVA的垃圾回收，标记算法和复制算法的区别，用在什么场合？
9. http和https的区别，http1.x和http2.0的区别，SSL和TSL之间的区别
10. GC、G1和ZGC的区别
11. B+树和B树的区别，和红黑树的区别
12. 内存泄漏与内存溢出的区别
13. session的生命周期是多久
14. 关于Mina框架了解多少？（因为我在项目里用到了Mina，所以提到了这个部分）
15. java cas原理
16. JAVA线程池有哪些参数，如果自己设计一个线程池要考虑哪些问题？
17. Java的lock的底层实现？
18. mysql数据库默认存储引擎，有什么优点
19. MySQL的事务隔离级别，分别解决什么问题。
20. 四个表 记录成绩，每个大约十万条记录，如何找到成绩最好的同学
21. 常见的负载均衡算法有哪些
22. 如果Redis有1亿个key，使用keys命令是否会影响线上服务
23. Redis的持久化方式，aod和rdb，具体怎么实现，追加日志和备份文件，底层实现原理的话知道么
24. 请画一个完整大型网站的分布式服务器集群部署图
25. 多个RPC请求进来，服务器怎么处理并发呢
26. 讲一下Redis的哨兵机制
27. 数据库分库分表一般数据量多大才需要？
28. 如何保证数据库与redis缓存一致的
29. 项目中消息队列怎么用的？使用哪些具体业务场景？
30. JVM相关的分析工具有使用过哪些？具体的性能调优步骤吗？
31. MySQL的慢sql优化一般如何来做？除此外还有什么方法优化？
32. 线上的服务器监控指标，你认为哪些指标是最需要关注的？为什么？
33. 如何做压测，抗压手段
34. 秒杀模块怎么设计的
35. java 线程池的实现原理，threadpoolexecutor关键参数解释
36. hashmap的原理，容量为什么是2的幂次
37. 为什么要同时重写hashcode和equals
38. ConcurrentHashMap如何实现线程安全？
39. 介绍Java多线程的5大状态，以及状态图流转过程
40. 介绍下Synchronized、Volatile、CAS、AQS，以及各自的使用场景
41. B+树和红黑树时间复杂度
42. 如果频繁老年代回收怎么分析解决
43. JVM内存模型，新生代和老年的回收机制
44. mysql limit分页如何保证可靠性
45. java nio，bio，aio，操作系统底层nio实现原理
46. Spring IOC，autowired如何实现
47. Spring事务传播机制
48. 线程死锁排查
49. MySQL引擎及区别，项目用的哪个，为什么
50. RPC为什么用http做通信？
51. RPC两端如何进行负载均衡？
52. mycat分库分表、读写分离的实现
53. 分布式数据如何保证数据一致性
54. 高并发请求处理，流量削峰措施有哪些
55. Redis持久化RDB和AOF 的区别
56. MQ底层实现原理
57. 详细介绍下分布式 一致性Hash算法
58. nginx负载均衡的算法
59. Nginx 的 upstream目前支持 哪4 种方式的分配
60. 分布式集群部署后，从应用端哪些需要怎么调整
61. Dubbo默认使用什么注册中心，还有别的选择吗？
62. mongoDB、redis和memcached的应用场景，各自优势
63. 谈谈你性能优化的实践案例，优化思路？
64. 两千万用户并发抢购，你怎么来设计
65. Java中的多态
66. 为什么要同时重写hashcode和equals
67. Hashmap的原理
68. Hashmap如何变线程安全，每种方式的优缺点
69. 垃圾回收机制
70. Jvm的参数你知道的说一下
71. 设计模式了解的说一下啊
72. 手撕一个单例模式

**算法题目**

1. 手撕算法：反转单链表
2. 手撕算法：实现类似微博子结构的数据结构，输入一系列父子关系，输出一个类似微博评论的父子结构图
3. 手写java多线程
4. 手写java的soeket编程，服务端和客户端
5. 手撕算法： 爬楼梯，写出状态转移方程
6. 智力题：时针分针什么时候重合
7. 服务器如何负载均衡，有哪些算法，哪个比较好，一致性哈希原理，怎么避免DDOS攻击请求打到少数机器。
8. TCP连接中的三次握手和四次挥手，四次挥手的最后一个ack的作用是什么，为什么要time wait，为什么是2msl。
9. 数据库的备份和恢复怎么实现的，主从复制怎么做的，什么时候会出现数据不一致，如何解决。
10. Linux查看cpu占用率高的进程
11. 手撕算法：给定一个数字三角形，找到从顶部到底部的最小路径和。每一步可以移动到下面一行的相邻数字上。
12. 然后继续在这个问题上扩展
13. 求出最短那条的路径
14. 递归求出所有的路径
15. 设计模式讲一下熟悉的
16. 会不会滥用设计模式
17. 多线程条件变量为什么要在while体里
18. Redis的特点
19. Redis的持久化怎么做，aof和rdb，有什么区别，有什么优缺点。
20. Redis使用哨兵部署会有什么问题，我说需要扩容的话还是得集群部署。
21. 说一下JVM内存模型把，有哪些区，分别干什么的
22. 说一下gc算法，分代回收说下
23. MySQL的引擎讲一下，有什么区别，使用场景呢
24. 分布式事务了解么
25. 反爬虫的机制，有哪些方式
26. concurrent包下面用过哪些？
27. countdownlatch功能实现
28. synchronized和lock区别，重入锁
29. thread和runnable的区别
30. AtomicInteger实现原理(CAS自旋)
31. java并发sleep与wait、notify与notifyAll的区别
32. 如何实现高效的同步链表
33. java都有哪些加锁方式（synchronized、ReentrantLock、共享锁、读写锁等）
34. 设计模式（工厂模式、单例模式（几种情况）、适配器模式、装饰者模式）
35. maven依赖树，maven的依赖传递，循环依赖
36. synchronized和reentrantLock的区别，synchronized用在代码快、方法、静态方法时锁的都是什么?
37. 介绍spring的IOC和AOP，分别如何实现(classloader、动态代理)
38. JVM的内存布局以及垃圾回收原理及过程讲一下，讲一下CMS垃圾收集器垃圾回收的流程，以及CMS的缺点
39. redis如何处理分布式服务器并发造成的不一致
40. OSGi的机制
41. spring中bean加载机制，bean生成的具体步骤，ioc注入的方式
42. spring何时创建applicationContext
43. listener是监听哪个事件？
44. 介绍ConcurrentHashMap原理，用的是哪种锁，segment有没可能增大?
45. 解释mysql索引、b树，为啥不用平衡二叉树、红黑树
46. Zookeeper如何同步配置
47. Java线程池ThreadPoolEcecutor参数，基本参数，使用场景
48. MySQL的ACID讲一下，延伸到隔离级别
49. 说说RPC的要点
50. GC停顿原因，如何降低停顿？
51. JVM如何调优、参数怎么调？
52. 如何用工具分析jvm状态
53. 描述一致性hash算法（Redis）
54. 分布式雪崩场景?
55. 再谈谈消息队列
56. springmvc和spring-boot区别
57. @Autowired的实现原理
58. Bean的默认作用范围是什么？其他的作用范围？
59. 索引是什么概念有什么作用？MySQL里主要有哪些索引结构？哈希索引和B+树索引比较？
60. Java线程池的原理？线程池有哪些？线程池工厂有哪些线程池类型，及其线程池参数是什么？
61. hashmap原理，处理哈希冲突用的哪种方法？
62. 还知道什么处理哈希冲突的方法？
63. Java GC机制？GC Roots有哪些？
64. Java怎么进行垃圾回收的？什么对象会进老年代？垃圾回收算法有哪些？为什么新生代使用复制算法？
65. HashMap的时间复杂度？HashMap中Hash冲突是怎么解决的？链表的上一级结构是什么？Java8中的HashMap有什么变化？红黑树需要比较大小才能进行插入，是依据什么进行比较的？其他Hash冲突解决方式？
66. hash和B+树的区别？分别应用于什么场景？哪个比较好？
67. 项目里有个数据安全的，aes和md5的区别？详细点
68. 为什么MyISAM查询性能好？
69. 事务特性（acid）
70. 隔离级别
71. SQL慢查询的常见优化步骤？
72. 说下乐观锁，悲观锁（select for update），并写出sql实现
73. TCP协议的三次握手和四次挥手过程？
74. 用到过哪些rpc框架
75. 数据库连接池怎么实现
76. Java web过滤器的生命周期
77. ConcurrentHashMap 在Java7和Java8中的区别？为什么Java8并发效率更好？什么情况下用HashMap，什么情况用ConcurrentHashMap？
78. 加锁有什么机制？
79. ThreadLocal？应用场景？
80. 数据库水平切分，垂直切分的设计思路和切分顺序
81. Redis如何解决key冲突
82. soa和微服务的区别？
83. 单机系统演变为分布式系统，会涉及到哪些技术的调整？请从前面负载到后端详细描述。
84. 设计一个秒杀系统？

2.说说B+树和B树的区别，优缺点等？

3聊聊Spring，主要IOC等等

4多线程JUC包下的一些常见的类，比如CountDownLatch、Semaphore等

5.锁的概念，锁相关的关键字，volatile，synchronized。还比较了ReentrantLock与synchronized。

6.你了解哪些收集器？CMS和G1。详细谈谈G1的优点？什么时候进行Full GC呢？

7.Spring中涉及的一些设计模式

8.算法题：无序数列中求第k大的数(维护最小堆，然后依次遍历，与堆顶比较)

9.MySQL创建索引的原则，好处

10. 怎么实现一个线程安全的计数器？

1. 设计模式：讲了单例，工厂方法，抽象工厂，策略模式，观察者模式，代理模式，还顺便讲了下spring动态代理的实现原理

2. 线程池有哪些参数？分别有什么用？如果任务数超过的核心线程数，会发生什么？阻塞队列大小是多少？

3. HashMap的底层数据结构

4. 红黑树的具体结构及实现，红黑树与查找树的区别体现

5. 接着聊ConcurrentHashMap，底层实现

6. HashMap哈希函数的认识，JDK1.8采用的hash函数

7. 数据库索引，索引底层的实现，B+树的结构以及与普通查找树的优点

9. TCP三次握手四次挥手，四次挥手过程中服务端的哪几种状态，哪几种包

10. 已经有一个查询好友的接口，设计一个微信朋友圈，可以实现发表朋友圈，添加评论，查看评论等功能。主要是设计数据结构

1. 项目中用到dubbo？那你说说什么是rpc框架？和http调用的区别是什么？
2. Redis有哪些数据结构？持久化方案和区别？
3. Redis哨兵、集群的设计原理和区别？
4. Redis缓存和数据库会存在一致性问题吗？怎么解决
5. Kafka怎么保证数据可靠性？讲了生产者端发送消息到broker持久化，分区和副本机制，消费者消费消息的at-least-once和at-most-once？怎么实现Exactly-Once？