**面试题目**

**Java 基础**

Java基础  
一般什么情况下要重写hashCode和equals方法

HashMap如何实现，多线程环境下如何使用Map

a. 了解碰撞检测的标准方法，能够延伸到java如何处理碰撞冲突问题，能够回答出使用链表来解决这个问题，并能简述链表加节点的方式  
b. 了解使用锁/hashtable/concurrentHashMap来解决

自实现hashmap需要考虑哪些因素

LinkedHashMap和HashMap什么区别，是如何实现

hashmap增加数据过期功能，如何设计

IO与NIO区别，分别适用于哪些场景

IO是面向流的，NIO是面向缓冲区的  
IO流是阻塞的，NIO流是不阻塞的  
IO下载器，NIO聊天工具

介绍一下volatile关键字，为什么volatile不能保证原子性，volatile修饰的Map能否起到保护map内对象的作用

解释原子性的含义，CAS synchronized实现方式的区别

CAS 通过将内存中的值与数据进行比较，当数值一样时将内存中的数据替换为新的值。通过将这个过程封装在一个CPU指令内完成，从而保证了原子性  
synchronized 加锁来保证原子性

如果让你去实现类似String这样的非可变共享类你会怎么设计？

1.考察如何做到让一个类不可变。  
2.考察如何让一个类的实例做到可共享(比如String类型"abc"在堆内存里只有一份)。

对象和数组内存布局的主要区别？

1.数组有length对象没有。  
2.对象数据区往往不是整齐的，而数组的数据区是整齐的。

Java 进程间通讯的有几种方法

描述一下JVM加载class文件的原理机制

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

多线程并发  
什么是线程安全  
Java中如何停止一个线程？  
Java提供了很丰富的API但没有为停止线程提供API。JDK 1.0本来有一些像stop(), suspend() 和 resume()的控制方法但是由于潜在的死锁威胁因此在后续的JDK版本中他们被弃用了，之后Java API的设计者就没有提供一个兼容且线程安全的方法来停止一个线程。当run() 或者 call() 方法执行完的时候线程会自动结束,如果要手动结束一个线程，你可以用volatile 布尔变量来退出run()方法的循环或者是取消任务来中断线程

什么是线程池，为什么要使用它，哪几种线程拒绝策略  
创建线程的成本较高（继续提问高在哪里），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加速创建的效率

线程池  
Java有哪些线程池，各自有什么特点

FixedThreadPool/SingleThreadExecutor/CachedThreadPool

任务提交给线程池之后的处理策略

如果当前线程池中的线程数目小于corePoolSize，则每来一个任务，就会创建一个线程去执行这个任务  
如果当前线程池中的线程数目>=corePoolSize，则每来一个任务，会尝试将其添加到任务缓存队列当中，若添加成功，则该任务会等待空闲线程将其取出去执行；若添加失败（一般来说是任务缓存队列已满），则会尝试创建新的线程去执行这个任务  
如果当前线程池中的线程数目达到maximumPoolSize，则会采取任务拒绝策略进行处理  
如果线程池中的线程数量大于 corePoolSize时，如果某线程空闲时间超过keepAliveTime，线程将被终止，直至线程池中的线程数目不大于corePoolSize；如果允许为核心池中的线程设置存活时间，那么核心池中的线程空闲时间超过keepAliveTime，线程也会被终止

任务缓存队列及排队策略

workQueue的类型为BlockingQueue，通常可以取下面三种类型  
ArrayBlockingQueue：基于数组的先进先出队列，此队列创建时必须指定大小  
LinkedBlockingQueue：基于链表的先进先出队列，如果创建时没有指定此队列大小，则默认为Integer.MAX\_VALUE  
SynchronousQueue：这个队列比较特殊，它不会保存提交的任务，而是将直接新建一个线程来执行新来的任务

任务拒绝策略

ThreadPoolExecutor.AbortPolicy;//丢弃任务并抛出RejectedExecutionException异常  
ThreadPoolExecutor.DiscardPolicy;//也是丢弃任务，但是不抛出异常  
ThreadPoolExecutor.DiscardOldestPolicy;//丢弃队列最前面的任务，然后重新尝试执行任务（重复此过程）  
ThreadPoolExecutor.CallerRunsPolicy;//由调用线程处理该任务

如何实现一个BlockingQueue?

方式一：wait/notify  
方式二：lock/condition  
加分项：wait的条件为什么使用 while？ 而不是if？

简述CountDownLatch, CyclicBarrier, Semaphore区别

a. CountDownLatch锁的状态是通过初始化一个计数器的方式实现；减少计数来尝试释放锁；释放锁失败一直轮训队列并尝试释放；（共享锁）  
b. CyclicBarrier锁的状态是通过初始化一个参与者数量来实现；每个参与者执行完毕，调用await减少参与者数量，直到所有参与者执行完毕，执行（若存在）后续任务；  
c. Semaphore锁的状态是通过许可证(permits)的数量来实现，通过增加/减少许可证数量来获取/释放锁；许可证的数量不够一直尝试获取（默认：非公平，共享锁）

ArrayList并发删除元素会有什么问题？如何做到并发删除？

a. 了解ArrayList线程不安全，简述ArrayList的fast-fail机制  
b. 问题引申：如何构建线程安全的ArrayList  
加分项：Collections工具类包装，CopyOnWriteArrayList，以及两者有什么区别？

如何实现一个LRU缓存？

掌握LinkedHashMap原理  
考虑缓存定时过期处理  
弱引用（WeakReference）的使用  
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中使用SimpleDateFormat

双重检查是解决什么问题，哪里需要特别注意

并发执行时的重复创建问题

描述下volatile关键字的作用及其原理

操作系统基本使用  
如何找到指定端口被哪个进程占用了，比如8080端口  
/usr/sbin/lsof -i :8080 或者 netstat -natp | grep 8080

Linux下面有个top命令，里面能看到 load指标，能否讲讲Load具体指的是什么意思  
load 要结合cpu数量来看，我们假设cpu数量为4，每个cpu都正好满负荷运转时，load为4.如果每个cpu处理任务过程中，还有一个任务在排队等待，那么load为8

进程之间如何通信，进程和线程的区别，多进程和多线程如何选择  
考察下awk命令的使用，如何分析日志  
数据库  
事务的概念， mysql的索引如何实现

MySQL SQL查询较慢，如何排查

a. 先看explain sql，看看SQL的执行计划  
b. 执行计划中重点关注，走到了哪个索引，如果没有索引，则建立索引。原因，好的索引可以减少查找全表的数据遍历

简述TRUNCATE与DELETE之间的区别

TRUNCATE：DDL，隐式事务提交，不可回滚；执行快速，后台操作等同于DROP TABLE+CREATE TABLE，数据文件无关性，与表锁冲突，不支持外键关联表（自关联除外）  
DELETE：DML，显式事务提交，可回滚； 执行较慢，产生大量事务日志

笔试  
请实现两个线程，使之交替打印1 - 100，如：两个线程分别为：Printer1和Printer2，最后输出结果为：  
Printer1 — 1  
Printer2 — 2  
Printer1 — 3  
Printer2 — 4

翻转二叉树  
class TreeNode {  
TreeNode left ;  
TreeNode right ;  
Object value;  
}  
void reverse(TreeNode root){  
}

计算long类型的二级制表示1的个数

private static int count1(int n) {  
int sum = 0;  
while (n != 0) {  
n = n & (n - 1);  
sum++;  
}  
return sum;  
}  
<http://www.cnblogs.com/graphics/archive/2010/06/21/1752421.html>

string字符串反转  
String str="abcdefg"  
String reverse(String str){  
}

四个线程，分别执行一个任务，然后主线程汇总四个线程的计算结果

CyclicBarrier，FutureTask，CountDownLatch

设计一个函数，计算一个数组[7, 2, 4, 1, 5, 6, 3, ...]内，连续n个数的和的最大值

private static int maxSum(int[] nums, int n) {  
int maxSum = 0;  
for (int i = 0; i < nums.length - n + 1; i++) {  
int[] numsTemp = Arrays.copyOfRange(nums, i, i + n);  
System.out.println(JSON.toJSONString(numsTemp));  
int tempSum = Arrays.stream(numsTemp).sum();  
maxSum = tempSum > maxSum ? tempSum : maxSum;  
}  
return maxSum;  
}

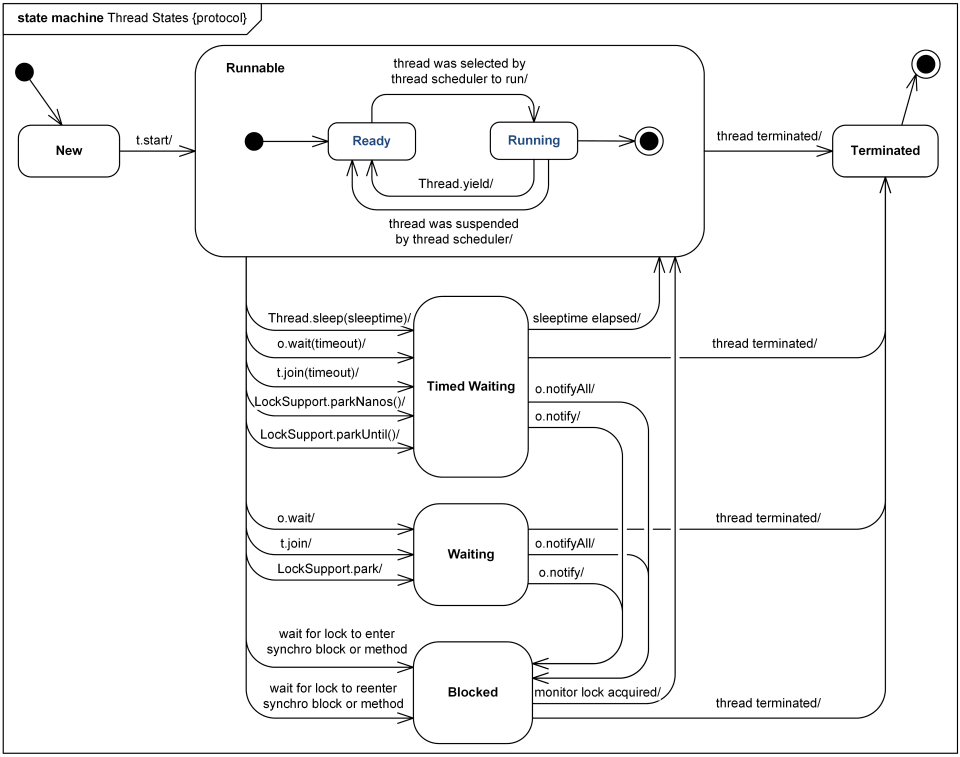
public static void main(String[] args) {  
int[] nums = new int[] { 7, 2, 4, 1, 5, 6, 3 };  
System.out.println(maxSum(nums, 3));  
}  
给一个字符串，请给出它的全部子字符串集合。

使用与字符串长度一样长度的二进制作为标识，每次+1即可  
如果需要指定子字符集元素个数，可以采用移位运算处理

经验：  
最好的项目是哪个？怎么设计和考虑的  
参考  
百度常用200问题  
133个Java面试问题列表  
JAVA面试题全集(上)

Java中线程的状态

New, Runnable, Blocked, Waiting, Time\_waiting, Terminated



在Java中线程的状态一共被分成6种：初始态：NEW创建一个Thread对象，但还未调用start()启动线程时，线程处于初始态。运行态：RUNNABLE在Java中，运行态包括就绪态 和 运行态。就绪态 该状态下的线程已经获得执行所需的所有资源，只要CPU分配执行权就能运行。所有就绪态的线程存放在就绪队列中。运行态 获得CPU执行权，正在执行的线程。由于一个CPU同一时刻只能执行一条线程，因此每个CPU每个时刻只有一条运行态的线程。阻塞态当一条正在执行的线程请求某一资源失败时，就会进入阻塞态。而在Java中，阻塞态专指请求锁失败时进入的状态。由一个阻塞队列存放所有阻塞态的线程。处于阻塞态的线程会不断请求资源，一旦请求成功，就会进入就绪队列，等待执行。PS：锁、IO、Socket等都资源。等待态当前线程中调用wait、join、park函数时，当前线程就会进入等待态。也有一个等待队列存放所有等待态的线程。线程处于等待态表示它需要等待其他线程的指示才能继续运行。进入等待态的线程会释放CPU执行权，并释放资源（如：锁）超时等待态当运行中的线程调用sleep(time)、wait、join、parkNanos、parkUntil时，就会进入该状态；它和等待态一样，并不是因为请求不到资源，而是主动进入，并且进入后需要其他线程唤醒；进入该状态后释放CPU执行权 和 占有的资源。与等待态的区别：到了超时时间后自动进入阻塞队列，开始竞争锁。终止态线程执行结束后的状态。注意：wait()方法会释放CPU执行权 和 占有的锁。sleep(long)方法仅释放CPU使用权，锁仍然占用；线程被放入超时等待队列，与yield相比，它会使线程较长时间得不到运行。yield()方法仅释放CPU执行权，锁仍然占用，线程会被放入就绪队列，会在短时间内再次执行。wait和notify必须配套使用，即必须使用同一把锁调用；wait和notify必须放在一个同步块中调用wait和notify的对象必须是他们所处同步块的锁对象。

**守护线程**

Java中有两种线程，一种是用户线程，另一种是守护线程。当进程中不存在非守护线程了，则守护线程自动销毁。通过setDaemon(true)设置线程为后台线程。注意thread.setDaemon(true)必须在thread.start()之前设置，否则会报IllegalThreadStateException异常；在Daemon线程中产生的新线程也是Daemon的；在使用ExecutorSerice等多线程框架时，会把守护线程转换为用户线程，并且也会把优先级设置为Thread.NORM\_PRIORITY。在构建Daemon线程时，不能依靠finally块中的内容来确保执行关闭或清理资源的逻辑

**ThreadLocal**

ThreadLocal可以实现每个线程绑定自己的值，即每个线程有各自独立的副本而互相不受影响。一共有四个方法：get, set, remove, initialValue。可以重写initialValue()方法来为ThreadLocal赋初值。如下：

private static final ThreadLocal TIME\_THREADLOCAL = new ThreadLocal(){

@Override

protected Long initialValue()

{

return System.currentTimeMillis();

}

};

ThreadLocal建议设置为static类型的

通常线程被阻塞后便进入内核（Linux）调度状态，会导致在用户态与内核态之间来回切换，影响锁的性能，所以引入了自旋(spin，其实就是一段啥都不做的循环)，在自旋一定时间后(这个时间一般是个经验值，一般和CPU核数相关)，如果还没有获得想要的数据，再进入阻塞。 最后，线程阻塞 和 锁 不是一个概念。

**CAS**

CAS缺点  
CAS存在一个很明显的问题，即ABA问题。

问题：如果变量V初次读取的时候是A，并且在准备赋值的时候检查到它仍然是A，那能说明它的值没有被其他线程修改过了吗？

如果在这段期间曾经被改成B，然后又改回A，那CAS操作就会误认为它从来没有被修改过。针对这种情况，java并发包中提供了一个带有标记的原子引用类AtomicStampedReference，它可以通过控制变量值的版本来保证CAS的正确性。

**Java中的线程池ThreadPoolExecutor**

可以通过ThreadPoolExecutor来创建一个线程池：

ThreadPoolExecutor(int corePoolSize, int maximumPoolSize, long keepAliveTime, TimeUnit unit, BlockingQueue workQueue, ThreadFactory threadFactory, RejectedExecutionHandler handler)  
1  
corePoolSize（线程池基本大小）：当向线程池提交一个任务时，若线程池已创建的线程数小于corePoolSize，即便此时存在空闲线程，也会通过创建一个新线程来执行该任务，直到已创建的线程数大于或等于corePoolSize时，才会根据是否存在空闲线程，来决定是否需要创建新的线程。除了利用提交新任务来创建和启动线程（按需构造），也可以通过 prestartCoreThread() 或 prestartAllCoreThreads() 方法来提前启动线程池中的基本线程。  
maximumPoolSize（线程池最大大小）：线程池所允许的最大线程个数。当队列满了，且已创建的线程数小于maximumPoolSize，则线程池会创建新的线程来执行任务。另外，对于无界队列，可忽略该参数。  
keepAliveTime（线程存活保持时间）：默认情况下，当线程池的线程个数多于corePoolSize时，线程的空闲时间超过keepAliveTime则会终止。但只要keepAliveTime大于0，allowCoreThreadTimeOut(boolean) 方法也可将此超时策略应用于核心线程。另外，也可以使用setKeepAliveTime()动态地更改参数。  
unit（存活时间的单位）：时间单位，分为7类，从细到粗顺序：NANOSECONDS（纳秒），MICROSECONDS（微妙），MILLISECONDS（毫秒），SECONDS（秒），MINUTES（分），HOURS（小时），DAYS（天）；  
workQueue（任务队列）：用于传输和保存等待执行任务的阻塞队列。可以使用此队列与线程池进行交互：

如果运行的线程数少于 corePoolSize，则 Executor 始终首选添加新的线程，而不进行排队。  
如果运行的线程数等于或多于 corePoolSize，则 Executor 始终首选将请求加入队列，而不添加新的线程。  
如果无法将请求加入队列，则创建新的线程，除非创建此线程超出 maximumPoolSize，在这种情况下，任务将被拒绝。  
threadFactory（线程工厂）：用于创建新线程。由同一个threadFactory创建的线程，属于同一个ThreadGroup，创建的线程优先级都为Thread.NORM\_PRIORITY，以及是非守护进程状态。threadFactory创建的线程也是采用new Thread()方式，threadFactory创建的线程名都具有统一的风格：pool-m-thread-n（m为线程池的编号，n为线程池内的线程编号）;  
handler（线程饱和策略）：当线程池和队列都满了，则表明该线程池已达饱和状态。

ThreadPoolExecutor.AbortPolicy：处理程序遭到拒绝，则直接抛出运行时异常 RejectedExecutionException。(默认策略)  
ThreadPoolExecutor.CallerRunsPolicy：调用者所在线程来运行该任务，此策略提供简单的反馈控制机制，能够减缓新任务的提交速度。  
ThreadPoolExecutor.DiscardPolicy：无法执行的任务将被删除。  
ThreadPoolExecutor.DiscardOldestPolicy：如果执行程序尚未关闭，则位于工作队列头部的任务将被删除，然后重新尝试执行任务（如果再次失败，则重复此过程）。  
可以使用两个方法向线程池提交任务，分别为execute()和submit()方法。execute()方法用于提交不需要返回值的任务，所以无法判断任务是否被线程池执行成功。submit()方法用于提交需要返回值的任务，线程池会返回一个Future类型的对象，通过这个对象可以判断任务是否执行成功。如Future future = executor.submit(task);

利用线程池提供的参数进行监控，参数如下：

getTaskCount()：线程池需要执行的任务数量。  
getCompletedTaskCount()：线程池在运行过程中已完成的任务数量，小于或等于taskCount。  
getLargestPoolSize()：线程池曾经创建过的最大线程数量，通过这个数据可以知道线程池是否满过。如等于线程池的最大大小，则表示线程池曾经满了。  
getPoolSize()：线程池的线程数量。如果线程池不销毁的话，池里的线程不会自动销毁，所以这个大小只增不减。  
getActiveCount()：获取活动的线程数。

**Synchronized**

1.Synchronized修饰非静态方法，实际上是对调用该方法的对象加锁，俗称“对象锁”。

Java中每个对象都有一个锁，并且是唯一的。假设分配的一个对象空间，里面有多个方法，相当于空间里面有多个小房间，如果我们把所有的小房间都加锁，因为这个对象只有一把钥匙，因此同一时间只能有一个人打开一个小房间，然后用完了还回去，再由JVM 去分配下一个获得钥匙的人。

情况1：同一个对象在两个线程中分别访问该对象的两个同步方法

结果：会产生互斥。

解释：因为锁针对的是对象，当对象调用一个synchronized方法时，其他同步方法需要等待其执行结束并释放锁后才能执行。

情况2：不同对象在两个线程中调用同一个同步方法

结果：不会产生互斥。

解释：因为是两个对象，锁针对的是对象，并不是方法，所以可以并发执行，不会互斥。形象的来说就是因为我们每个线程在调用方法的时候都是new 一个对象，那么就会出现两个空间，两把钥匙，

2.Synchronized修饰静态方法，实际上是对该类对象加锁，俗称“类锁”。

情况1：用类直接在两个线程中调用两个不同的同步方法

结果：会产生互斥。

解释：因为对静态对象加锁实际上对类（.class）加锁，类对象只有一个，可以理解为任何时候都只有一个空间，里面有N个房间，一把锁，因此房间（同步方法）之间一定是互斥的。

注：上述情况和用单例模式声明一个对象来调用非静态方法的情况是一样的，因为永远就只有这一个对象。所以访问同步方法之间一定是互斥的。

情况2：用一个类的静态对象在两个线程中调用静态方法或非静态方法

结果：会产生互斥。  
解释：因为是一个对象调用，同上。

情况3：一个对象在两个线程中分别调用一个静态同步方法和一个非静态同步方法  
结果：不会产生互斥。  
解释：因为虽然是一个对象调用，但是两个方法的锁类型不同，调用的静态方法实际上是类对象在调用，即这两个方法产生的并不是同一个对象锁，因此不会互斥，会并发执行。

**SimpleDateFormat非线程安全**

当多个线程共享一个SimpleDateFormat实例的时候，就会出现难以预料的异常。

主要原因是parse()方法使用calendar来生成返回的Date实例，而每次parse之前，都会把calendar里的相关属性清除掉。问题是这个calendar是个全局变量，也就是线程共享的。因此就会出现一个线程刚把calendar设置好，另一个线程就把它给清空了，这时第一个线程再parse的话就会有问题了。

解决方案:1. 每次使用时创建一个新的SimpleDateFormat实例；2. 创建一个共享的SimpleDateFormat实例变量，并对这个变量进行同步；3. 使用ThreadLocal为每个线程都创建一个独享的SimpleDateFormat实例变量

**java虚拟机**

1）Java 的数据结构相关的类实现原理，比如 LinkedList，ArrayList，HashMap，TreeMap 这一类的。以下简单模拟一个数据结构的连环炮。  
HashMap 是不是有序的？有没有有顺序的 Map 实现类？TreeMap 和 LinkedHashMap 是如何保证它的顺序的？

2）Java 的虚拟机的内容。这部分主要包括三部分，GC、类加载机制，以及内存。  
什么时候一个对象会被 GC？  
为什么要在这种时候对象才会被 GC？  
GC 策略都有哪些分类？  
这些策略分别都有什么优劣势？都适用于什么场景？  
举个实际的场景，选择一个 GC 策略？  
为什么要选择这个策略？

内存分为哪几部分，这些部分分别都存储哪些数据？  
一个对象从创建到销毁都是怎么在这些部分里存活和转移的？  
内存的哪些部分会参与 GC 的回收？  
Java 的内存模型是怎么设计的？  
结合内存模型的设计谈谈 volatile 关键字的作用？  
提到可见性，那么接着可见性这三个字，还可以继续问并发的内容。

<http://ifeve.com/doug-lea/>

**缓存**

缓存穿透

查询一个必然不存在的数据。比如文章表，查询一个不存在的id，每次都会访问DB，如果有人恶意破坏，很可能直接对DB造成影响。

缓存失效

如果缓存集中在一段时间内失效，DB的压力凸显。这个没有完美解决办法，但可以分析用户行为，尽量让失效时间点均匀分布。  
缓存系统不得不考虑的另一个问题是缓存穿透与失效时的雪崩效应。缓存穿透是指查询一个一定不存在的数据，由于缓存是不命中时被动写的，并且出于容错考虑，如果从存储层查不到数据则不写入缓存，这将导致这个存在的数据每次请求都要到存储层去查询，失去了缓存的意义。

有 很多种方法可以有效地解决缓存穿透问题，最常见的则是采用布隆过滤器，将所有可能存在的数据哈希到一个足够大的bitmap中，一个一定不存在的数据会被 这个bitmap拦截掉，从而避免了对底层存储系统的查询压力。  
*一个更为简单粗暴的方法，如果一个查询返回的数据为空（不管是数 据不存在，还是系统故障），我们仍然把这个空结果进行缓存，但它的过期时间会很短，最长不超过五分钟。*

缓存失效时的雪崩效应对底层系统的冲击非常可怕。遗憾的是，这个问题目前并没有很完美的解决方案。大多数系统设计者考虑用加锁或者队列的方式保证缓存的单线 程（进程）写，从而避免失效时大量的并发请求落到底层存储系统上。设计缓存过期机制理论上能够将各个客户端的数据失效时间均匀地分布在时间轴上，一定程度上能够避免缓存同时失效带来的雪崩效应。

**Redis**

Redis是一个支持内存存储及持久化存储的K-V存储系统  
支持复杂数据结构，相比与Memcached仅支持简单的key-value存储，Redis原生支持几类常用的存储结构

单线程  
高性能，避免过多考虑并发、锁、上下文切换  
数据一致性好，例如对一个计数的并发操作，不会有‘读者写者’问题  
单线程无法利用多核，单可以通过启动多个实例方式，充分利用多核  
原生支持Master-Slave  
过期机制被动过期——client访问key时，判断过期时间选择是否过期  
主动过期——默认使用valatile-lru  
volatile-lru：从已设置过期时间的数据集中挑选最近最少使用的数据淘汰

**分布式Redis架构如何实现高可用(HA)？**

采用M-S高可用方案，原因也式由于其Master-Slave的特性  
服务域名化是必要的，目前大型的Redis集群应用大多采用域名方式

**RPC**

HTTP也同样能够实现远程服务调用，那么HTTP和RPC的区别是什么呢？HTTP协议是一种语言无关的应用层资源访问协议，与RPC实现方式和侧重点各有不同，HTTP建立在TCP协议至上，优点是简单、易用、可理解性强且语言无关，HTTP的缺点是协议头较重，一般请求到具体服务器的链路较长，可能会有DNS解析、Nginx代理等，比较适合外部服务的交互。例如某个第三方请求微博开放平台的外部接口。

RPC与语言有一定耦合，但使用方式更自然，就像是使用一个类的API，RPC服务的自动化程度比较高，能够实现强大的服务治理功能，性能十分优秀。与HTTP相比，RPC的缺点就是相对复杂，学习成本稍高。RPC比较适合内部服务的访问与统一管理。

**分布式**

极热访问的缓存数据失效,容易导致数据库load瞬间飙高  
复制  
m-s方式，场景  
多写方式：数据一致性  
多副本的场景下,如何实现cas? 怎么保证master和所有副本数据的一致性?

**SQL**

写一个sql查询Log表Num字段连续出现至少三次的数字。例如只有数字1是下表连续出现至少3次的数字。

| Id | Num |
| --- | --- |
| 1 | 1 |
| 2 | 1 |
| 3 | 1 |
| 4 | 2 |
| 5 | 1 |
| 6 | 2 |
| 7 | 2 |

【解答】

select DISTINCT(l1.Num) from Logs l1, Logs l2, Logs l3 where l1.Id+1=l2.Id and l1.Id+2=l3.Id and l1.Num=l2.Num and l1.Num=l3.Num;

写SQL查询Person表所有重复的邮箱地址

| Id | Email |
| --- | --- |
| 1 | [a@b.com](https://yuque.antfin-inc.com/zhiyu.ws/note/rns2hu) |
| 2 | [c@d.com](https://yuque.antfin-inc.com/zhiyu.ws/note/rns2hu) |
| 3 | [a@b.com](https://yuque.antfin-inc.com/zhiyu.ws/note/rns2hu) |

如上表，返回如下结果

| Email |
| --- |
| [a@b.com](https://yuque.antfin-inc.com/zhiyu.ws/note/rns2hu) |

select distinct(p.Email) from Person p, Person q where [p.Id](http://p.id/)!=[q.Id](http://q.id/) and p.Email=q.Email;

**大数据**

spark rdd原理 持久化类型， Spark REPL（spark-shell）中的代码是如何分布式执行的

**基础题库**

1、如何排查一个OOM问题？  
思路清晰，jstat等工具清楚用途，懂得内存分析工具的通过。  
2、Nginx有哪些特点？  
高性能的 HTTP和反向代理，低系统占用以及高并发处理，擅长处理静态内容。  
3、从网络角度看，访问 www.taobao.com 发生了哪些事情？（考察网络协议栈，顺带可考察常用负载均衡、cdn等）  
    - 回答出 DNS、四层协议栈无误  
    - 回答出 DNS 、四层负载、七层负载  
4、DB 常用索引类型  
    - 主键、普通二级索引、唯一索引  
    - 从数据结构来讲：B-Tree、位图、Hash  
    - 能回答出聚簇与非聚簇索引  
5、解决哈希冲突的方法？哈希算法的要求？一致性哈希的原理？  
6、JVM内存模型  
7、面试官自行扩展

**Spring（可选，如果基础部分答得很好）**

1. Spring 关键技术  
       - IoC和AOP
2. 什么是 IoC，有什么好处，DI 的几种实现方式  
       - 将依赖交由第三方容器管理和注入，主要目的是解耦合  
       - 构造器注入、setter注入、接口注入，构造器注入无法处理循环依赖问题
3. 什么是 Aop，为什么使用 Aop，Spring 内部有没有使用 Aop？Spring Aop 的实现  
       - 处理系统中的横切关注点，比如日志、安全、事务  
       - 典型的事务处理使用了 Aop  
       - 基于 Jdk 动态代理和 Cglib 字节码增强，Jdk 动态代理依赖接口
4. 有一个标记为 [*@*Transactional](https://yuque.antfin-inc.com/Transactional)的方法，实际调用发现事务并未生效，可能是什么原因？（考察对 Aop 的理解）   
       - 有可能是类调用自身的被标记为 [*@*Transactional](https://yuque.antfin-inc.com/Transactional)的方法，由于 Spring 基于代理模式的 Aop 实现，[*@*Tranactional](https://yuque.antfin-inc.com/Tranactional)注解不会生效

**高可用相关：**

保证高可用场景的一些方法和策略，能从实践角度阐明，能够说出实现原理更佳。（可能不全，面试官可扩展）  
开关&预案、限流降级（熔断保护）：限流的常见策略、使用场景，有哪些常见算法。  
性能优化：性能优化场景策略、性能诊断工具、静态化、秒杀&热点解法  
容量评估：容量评估的方式、经验值、具体的做法。  
服务治理、依赖治理：服务发现、服务权重、客户端容灾、安全管控，分布式调用链的方案和实现。  
监控：系统&业务监控的做法，推&拉模式的区别，监控数据的聚合方案等

**扩展性相关**

服务化：  
互联网服务的架构一般是怎样？一次RPC的过程？软负载、配置中心的原理等。  
 数据库：  
数据库水平切分、垂直切分的区别、优缺点、适合场景；水平切分的常见切片规则；  
数据库主从一致性的解法、如何保证DB和Cache的一致性

SQL优化  
分布式系统原理  
hive  
hive怎么实现分组排序，比如：数据脚本点击

找出每个部门前3高薪水的人。  
员工薪水表(employee)

+----+-------+--------+--------------+  
| Id | Name | Salary | DepartmentId |  
+----+-------+--------+--------------+  
| 1 | Joe | 70000 | 1 |  
| 2 | Henry | 60000 | 1 |  
| 3 | Sam | 60000 | 1 |  
| 4 | Max | 90000 | 1 |  
| 5 | Janet | 69000 | 1 |  
| 6 | Randy | 85000 | 1 |  
| 7 | Alice | 85000 | 2 |  
| 8 | Bob | 80000 | 2 |  
+----+-------+--------+--------------+  
部门表(department)

+----+----------+  
| Id | Name |  
+----+----------+  
| 1 | IT |  
| 2 | Sales |  
+----+----------+  
期望结果：

+----+----------+----+--------------+  
| Id | Name |rank|DepartmentName|  
+----+----------+----+--------------+  
| 4 | Max |1 |IT |  
| 6 | Randy |2 |IT |  
| 1 | Joe |3 |IT |  
| 7 | Alice |1 |Sales |  
| 8 | Bob |2 |Sales |  
+----+----------+----+--------------+  
考点：基础的SQL能力。需要解释下背后基本的mapreduce过程。  
参考答案（hive版，如果是mysql或者其他DB需要其他实现方法）：

select a.Id, a.Name, a.DepartmentId, b.Name DepartmentName  
from (  
select Id, Name,DepartmentId, row\_number() over (partition by DepartmentId order by Salary desc) rank  
from employee  
) a  
left join department b  
on a.DepartmentId=b.Id  
where rank <= 3;  
如果有出现并列的情况怎么处理？比如下面：

员工薪水表(employee)

+----+-------+--------+--------------+  
| Id | Name | Salary | DepartmentId |  
+----+-------+--------+--------------+  
| 1 | Joe | 70000 | 1 |  
| 2 | Henry | 60000 | 1 |  
| 3 | Sam | 60000 | 1 |  
| 4 | Max | 90000 | 1 |  
| 5 | Janet | 69000 | 1 |  
| 6 | Randy | 85000 | 1 |  
| 7 | Alice | 85000 | 2 |  
| 8 | Bob | 80000 | 2 |  
| 9 | Wendy | 90000 | 1 |  
+----+-------+--------+--------------+  
部门表(department)

+----+----------+  
| Id | Name |  
+----+----------+  
| 1 | IT |  
| 2 | Sales |  
+----+----------+  
期望结果：

+----+----------+----+--------------+  
| Id | Name |rank|DepartmentName|  
+----+----------+----+--------------+  
| 3 | Max |1 |IT |  
| 9 | Wendy |1 |IT |  
| 6 | Randy |2 |IT |  
| 1 | Joe |3 |IT |  
| 7 | Alice |1 |Sales |  
| 1 | Bob |2 |Sales |  
+----+----------+----+--------------+  
边界情况：如果出现并列的情况该怎么办，有并列第一，并列第二，并列第三的时候怎么解决？  
参考答案：

select a.Id, a.Name, a.DepartmentId, b.Name DepartmentName , rank  
from (  
select Id, Name,DepartmentId, dense\_rank() over (partition by DepartmentId order by Salary desc) rank  
from employee  
) a  
left join department b  
on a.DepartmentId=b.Id  
where rank <= 3;  
还是实现1，2的效果，但是不使用sql的函数特性，使用python或者java怎么来实现取到每个部门的前N高薪水的员工（分组排序取Top）呢？下面是python写的框架，当然使用java也是可以的。 如果有流式计算的经验，这道题考虑下在线算法怎么实现。

python模版

def findTopN(data, n):  
# code here

if **name** == "**main**":  
# 不用考虑数据量的问题，假设都可以放入到内存中  
data = [{'Id':1, 'Name':Joe, 'Salary':70000, 'DepartmentId':1},{...}]  
findTopN(data, 3)  
Java模版

/\*\*

* 其中涉及到的class的定义SalaryMap:
* public SalaryMap {
* private int id;
* private String name;
* ...
* public int getId() {
* return this.id;
* }
* public String getName() {
* return this.name
* }
* }
* Definition of EmployeeMap
* public EmployeeMap {
* private String name;
* public String getName() {
* return this.name
* }
* public void setName(String name){
* this.name = name
* }
* }  
  \*/

public class Solution {  
public List findTopN(List data, int N) {  
}  
public static void main(String[] args) {  
// 忽略data是怎么被赋值，也不考虑数据量过大内存放不下的问题  
List data = readDataFromFile();  
int N =3;  
List result = findTopN(data, N);  
}  
}  
查看点：程序设计、代码风格 和 时间空间复杂度分析  
边界情况：如果出现并列的情况该怎么办，有并列第一，并列第二，并列第三的时候怎么解决？  
参考答案：

还是实现1，2的效果，按照下面模版，写出mapreduce版本。

基础题目  
Java线程的状态

进程和线程的区别，进程间如何通讯，线程间如何通讯

HashMap的数据结构是什么？如何实现的。和HashTable，ConcurrentHashMap的区别

Cookie和Session的区别

索引有什么用？如何建索引？

ArrayList是如何实现的，ArrayList和LinkedList的区别？ArrayList如何实现扩容。

equals方法实现

面向对象

线程状态，BLOCKED和WAITING有什么区别

JVM如何加载字节码文件

JVM GC，GC算法。

什么情况会出现Full GC，什么情况会出现yong GC。

JVM内存模型

Java运行时数据区

事务的实现原理

CopyOnWriteArrayList实现原理

深克隆和浅克隆

事务隔离级别

事务的四个关键属性(ACID)

技术深度

有没有看过JDK源码，看过的类实现原理是什么。

HTTP协议

TCP协议

一致性Hash算法

JVM如何加载字节码文件

类加载器如何卸载字节码

IO和NIO的区别，NIO优点

Java线程池的实现原理，keepAliveTime等参数的作用。

HTTP连接池实现原理

数据库连接池实现原理

数据库的实现原理

技术框架

看过哪些开源框架的源码

为什么要用Redis，Redis有哪些优缺点？Redis如何实现扩容？

Netty是如何使用线程池的，为什么这么使用

为什么要使用Spring，Spring的优缺点有哪些

Spring的IOC容器初始化流程

Spring的IOC容器实现原理，为什么可以通过byName和ByType找到Bean

Spring AOP实现原理

消息中间件是如何实现的，技术难点有哪些

Zookeeper实现原理，以及选主算法

为什么需要配置中心，配置中心如何实现的

系统架构

如何搭建一个高可用系统

哪些设计模式可以增加系统的可扩展性

介绍设计模式，如模板模式，命令模式，策略模式，适配器模式、桥接模式、装饰模式，观察者模式，状态模式，访问者模式。

抽象能力，怎么提高研发效率。

什么是高内聚低耦合，请举例子如何实现

什么情况用接口，什么情况用消息

如果AB两个系统互相依赖，如何解除依赖

如何写一篇设计文档，目录是什么

什么场景应该拆分系统，什么场景应该合并系统

系统和模块的区别，分别在什么场景下使用

分布式系统

分布式事务，两阶段提交。

如何实现分布式锁

如何实现分布式Session

如何保证消息的一致性

负载均衡

正向代理（客户端代理）和反向代理（服务器端代理）

CDN实现原理

怎么提升系统的QPS和吞吐量

DNS的实现原理

介绍下PAXOS协议

介绍下Zookeeper的ZAB协议，如何选举LEADER？如何

实战能力

有没有处理过线上问题？出现内存泄露，CPU利用率标高，应用无响应时如何处理的。

开发中有没有遇到什么技术问题？如何解决的

如果有几十亿的白名单，每天白天需要高并发查询，晚上需要更新一次，如何设计这个功能。

新浪微博是如何实现把微博推给订阅者

Google是如何在一秒内把搜索结果返回给用户的。

12306网站的订票系统如何实现，如何保证不会票不被超卖。

如何实现一个秒杀系统，保证只有几位用户能买到某件商品。

缓存失效如何解决？

从数据库查询10G的数据并加载到内存中？

如何设计一个流控功能？

软能力

如何学习一项新技术，比如如何学习Java的，重点学习什么

有关注哪些新的技术

工作任务非常多非常杂时如何处理

项目出现延迟如何处理

和同事的设计思路不一样怎么处理

如何保证开发质量

职业规划是什么？短期，长期目标是什么

团队的规划是什么

能介绍下从工作到现在自己的成长在那里

**电话面试招聘总结**

|  |  |  |  |
| --- | --- | --- | --- |
| 招聘标准 | 重点考察 | 加分项 | 进阶问题 |
| 4-5年，小公司 | 潜力，  基本功，不用再考虑高并发，分布式之类，因为没有场景。  主动性 | 是否足够聪明，迫切需要挑战 |  |
| 工作年限在10年左右 | 基础性的问题，偶尔提1-2个，不宜多；  重点在架构经验，能否把事情讲清楚。 | 线上运维经验： 比如监控。 项目中最有成就，收获最大的是什么？ |  |

java基础问题：

concurrentMap与HashMap的区别

stringbuffer与 stringbuilder的区别

java进阶问题：

|  |
| --- |
| FullGC原因  jvm堆内存的结构  乐观锁和悲观锁  zk的开启的端口号 |

其他小技巧；

1. 要让面试者编写代码
2. 面对面面试比电话面试更合适。