|  |
| --- |
| 技术问题(至少50个) |
| 问题1：线程与进程区别?  回答：进程是所有线程的集合，每一个线程是进程中的一条执行路径。  问题2：什么是多线程？  回答：及时提高程序的效率（同时执行，互不影响）。  问题2：多线程应用场景？  回答：主要能体现到多线程提高程序效率。  举例: 迅雷多线程下载、分批发送短信、爬虫等。  问题3：创建多线程有哪些方式？  回答： 1、使用继承Therad类方式 继承Thread类重写run方法 2、使用实现runlabe接口方式3、使用匿名内部类方式 4、callable 5、使用线程池创建线程。  问题4：redis的持久化方式？  回答：RDB快照方式和AOF日志模式  RDB特点：容易丢失数据，效率高，存数据格式：dump.rdb  AOF的点：增删改操作记录下来，不会造成数据丢失，效率相对RDB低  问题5：你们在项目中是怎么持久化的呢？  回答：使用的时候是两点结合使用，服务器运行的时候使用AOF方式记录，一定时间点用RDB快照的方式持久化，并把之前AOF记录的方式删除，如果服务器down机了，先根据RDB快照做恢复，再根据AOF方式做恢复，就不会造成数据丢失，效率也不会降低。  问题6：redis有哪些数据类型？  回答：String hash list set zset  问题7：如何用redis实现一个消息队列？  回答：利用redis中的list数据类型，因为list是有序的，然后利用lpush放入消息等待处理和rpop取出待处理消息进行处理  问题8：什么是堆栈？  回答：栈的空间比较小，存放函数的参数值，局部变量的值等。  堆的空间比较大，用于存放new的对象。  与C++不同，Java自动管理栈和堆，程序员不能直接地设置栈或堆。  堆（数据结构）：堆可以被看成是一棵树，如：二叉树。  栈（数据结构）：一种先进后出的数据结构。  问题9：.Java中的方法覆盖(Overriding)和方法重载(Overloading)是什么意思？  回答：Java中的方法重载发生在同一个类里面两个或者是多个方法的方法名相同但是参数不同的情况。  方法覆盖是说子类重新定义了父类的方法。方法覆盖必须有相同的方法名，参数列表和返回类型。与此相对，覆盖者可能不会限制它所覆盖的方法的访问。  问题10：接口和抽象类的区别是什么？  回答：接口中所有的方法隐含的都是抽象的。而抽象类则可以同时包含抽象和非抽象的方法。类可以实现很多个接口，但是只能继承一个抽象类  类如果要实现一个接口，它必须要实现接口声明的所有方法。但是，类可以不实现抽象类声明的所有方法，当然，在这种情况下，类也必须得声明成是抽象的。  抽象类可以在不提供接口方法实现的情况下实现接口。  Java接口中声明的变量默认都是final的。抽象类可以包含非final的变量。  Java接口中的成员函数默认是public的。抽象类的成员函数可以是private，protected或者是public。  接口是绝对抽象的，不可以被实例化。抽象类也不可以被实例化，但是，如果它包含main方法的话是可以被调用的。  问题11：什么是值传递和引用传递？  回答：对象被值传递，意味着传递了对象的一个副本。因此，就算是改变了对象副本，也不会影响源对象的值。  对象被引用传递，意味着传递的并不是实际的对象，而是对象的引用。因此，外部对引用对象所做的改变会反映到所有的对象上。  问题12：Java反射机制？  回答：通过类(Class对象)，可以得出当前类的fields、method、construtor、interface、superClass、modified等，同是可以通过类实例化一个实例、设置属性、唤醒方法。Spring中一切都是返射、struts、hibernate都是通过类的返射进行开发的。我们代码中一般也就用他写一些工具类。比如jdbc工具类等  Java反射机制主要提供了以下功能：  在运行时判断任意一个对象所属的类；  在运行时构造任意一个类的对象；  在运行时判断任意一个类所具有的[成员变量](https://baike.baidu.com/item/%E6%88%90%E5%91%98%E5%8F%98%E9%87%8F" \t "_blank)和方法；  在运行时调用任意一个对象的方法；生成动态代理。  问题13：算法复杂度  回答：[算法复杂度](https://baike.baidu.com/item/%E7%AE%97%E6%B3%95%E5%A4%8D%E6%9D%82%E5%BA%A6" \t "_blank)分为时间复杂度和[空间复杂度](https://baike.baidu.com/item/%E7%A9%BA%E9%97%B4%E5%A4%8D%E6%9D%82%E5%BA%A6" \t "_blank)。其作用： 时间复杂度是指执行算法所需要的计算工作量；而空间复杂度是指执行这个算法所需要的内存空间。  时间复杂度常用[大O符号](https://baike.baidu.com/item/%E5%A4%A7O%E7%AC%A6%E5%8F%B7" \t "_blank)表述  问题14：&和&&的区别  回答：&和&&都可以用作逻辑与的运算符，&&为短路与，&不是短路与。  另外&可以做为整数的位运算符  例1：对于if(str != null&& !str.equals(“”))表达式，当str为null时，后面的表达式不会执行，所以不会出现NullPointerException如果将&&改为&，则会抛出NullPointerException异常。  例2：If(x==33 &++y>0) y会增长，if(x==33 && ++y>0)不会增长  问题15：字节流与字符流的区别  字节流是按字节读取或写入设备，但字符流是以字符为单位读取或写入设备。  如果是二进制文件，需要用字节流读取。一般来说，字符流只处理文本文件（txt、word、excel等）。在设备中，大多数情况是以字节形式存储数据的，因此字符流通过需要传入字节流当参数。  问题16：XML和Json的特点  回答：xml有且只有一个根节点，所有的标签都需要自定义，是纯文本格式；  json：json对象(就是在{}中存储键值对，键和值之间用冒号分隔，键 值 对之间用逗号分隔);  json数组(就是[]中存储多个json对象，json对象之间用逗号分隔)  （两者间可以进行相互嵌套）数据传输的载体之一  问题17：内部类  回答：一个文件中，可以有多个类，但只能有一个public的类，并且public的类名必须与文件名相一致。  内部类就是在一个类的内部定义的类。内部可以定义在除参数位置上的任意位置。  问题18：sql锁：  悲观锁：就是考虑问题很悲观，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；  实现：sql语句后边加上for update  例子：Select id,nam from biao for update  乐观锁:就是考虑问题很乐观，每次去拿数据的时候都认为别人不会修改，所以不会上锁  实现：在表里边加一个vesion  例子：  Select max(nub) ,version from biao  Update biao set nub=nub+1,version=vsersion+1 where id=id and version =version  问题19：方法锁  回答：chronized锁和lock锁  区别：  1.Lock是一个接口，而synchronized是Java中的关键字;  2.synchronized当一个线程获取了对应的锁，并执行该代码块时，其他线程便只能一直等待，等待获取锁的线程释放锁,不能够响应中断,（释放：1执行完线程自动释放2发生异常jvm让线程释放）（（比如调用sleep方法）），这样的好处是不会导致死锁现象发生。  Lock锁，可以不让等待的线程一直无期限地等待下去，比如只等待一定的时间或者响应中断。  但Lock在发生异常时，如果没有主动通过unLock()去释放锁，则很可能造成死锁现象，因此使用Lock时必须在try{}catch{}块中进行，需要在finally块中释放锁；  通过Lock可以知道有没有成功获取锁，而synchronized却无法办到。  在性能上来说，如果竞争资源不激烈，两者的性能是差不多的，而当竞争资源非常激烈时（即有大量线程同时竞争），此时Lock的性能要远远优于synchronized。所以说，在具体使用时要根据适当情况选择。  在并发量比较小的情况下，使用synchronized是个不错的选择，但是在并发量比较高的情况下，其性能下降很严重，此时ReentrantLock（可重入锁，唯一实现了Lock接口的类）是个不错的方案。  问题20：Jvm调优  回答：因为我们的项目，web服务器中间件都用的是Tomcat，所以jvm的调优一般都是针对Tomcat的调优。调优工具可以可以借用jdk自带的**VisualVM工具，到**JDK安装目录/bin目录下，双击jvisualvm.exe文件，直接启动，就可以查看到当前的堆空间大小分配情况、线程监控情况和垃圾回收监控等。根据这些信息，再进行相应的调整。  我觉的Jvm调优的重点就是垃圾回收（gc，garbage collection）和内存管理。垃圾回收的时候会导致，整个虚拟机暂停服务。因此，应该尽可能地缩短垃圾回收的处理时间。  Java中垃圾回收机制是jvm自动完成的，我们不用操作，但可以通过修改一些配置对他进行影响；  首先可以开启server模式，这样虽  然启动慢点，但是长期运行效率高；（修改%JAVA\_HOME%/jre/lib/i386/jvm.cfg文件开启）  2、针对JVM堆的设置，JVM初始堆内存分配有-Xms指定，默认是物理内存的1/64；  最大分配堆内存有-Xmx指定，默认是物理内存的1/4；当堆内存小于40%时，JVM就会自动增加，直到最大值。当空余堆内存大于70%时，JVM就会自动减少，直到最小值。因此，为了防止垃圾收集器在最小、最大之间收缩堆而产生额外的时间，我们通常把最大、最小设置为相同的值；  3、配置年轻代(Xmn)的值，持久代一般固定大小为64m，所以增大年轻代后，将会减小年老代大小。此值对系统性能影响较大，Sun官方推荐配置为整个堆的3/8。  4、设置个线程的堆栈大小Xss，每个线程默认堆栈大小为1M，可根据应用的线程所需内存大小进行调整。在相同物理内存下，减小这个值能生成更多的线程。但是操作系统对一个进程内的线程数也是有限制的，不能无限生成，经验值在3000~5000左右。  5、回收器的选择，JVM给了三种选择：串行收集器、并行收集器、并发收集器，JVM会根据当前系统配置进行判断，自动选择。但我们可以对一些参数进行设置。串行收集器只适用于小数据量的情况，一般不用处理；可以配置年轻代使用并发收集器，老年代使用并行收集器；如果响应时间有限，就选择并发收集器，尽可能设大年轻代；  如果吞吐量优先就选择并行收集器，也尽可能设大年轻代；  6、禁用Tomcat的DNS查询。（当 Web 应用程序记录客户端的信息时，它也会记录客户端的 IP 地址或者通过域名服务器查找机器名转换为 IP 地址。 DNS 查询需要占用网络，并且可能从很多很远的服务器或者不起作用的服务器上去获取对应的 IP， 这样会消耗一定的时间。为了消除 DNS 查询对性能的影响，可以关闭 DNS 查询。方法是修改 server.xml 文件中的 enableLookups 参数值。）  7、线程数配置：Tomcat连接数过大可能引起的死机。所以，可以根据并发量在Tomcat的server.xml中修改他的最大线程数、初始化线程数等参数。（一般也就估计这配置，小了就配大点。也没有具体评估过）  我们在项目中，一般也就是项目出现问题以后，再去优化。Tomcat在年老代溢出（java.lang.OutOfMemoryError: Java heap space）、持久代溢出（java.lang.OutOfMemoryError: PermGen space）、堆栈溢出（java.lang.StackOverflowError）、线程溢出（Fatal: Stack size too smal）、内存溢出（java.lang.OutOfMemoryError: unable to create new native thread）是会抛出不同溢出。根据溢出去进行修改。  问题21：Java集合  Java里边主要有两种集合，collection接口和map接口，其中collection下又包含list和set两个子接口；  List子接口：有序，可以有重复元素。和数组类似，List可以动态增长，查找元素效率高，相对的插入删除元素效率低，因为会引起其他元素位置改变。  Set子接口:无序，不允许重复。检索元素效率低下，删除和插入效率高，插入和删除不会引起元素位置改变。set集合中的元素不按特定方式排序，只是简单的把对象加入集合中，就像往口袋里放东西。  List接口下有三个实现类ArrayList 、LinkedList和Vector  Vector是一种老的动态数组，是线程同步的，效率很低，一般不赞成使用  ArrayList  实现一个动态数组，它的规模可变并且能像链表一样被访问。它提供的功能类似Vector类但不同步，它是以Array方式实现的List，允许快速随机存取。特点是读快改慢；  LinkedList实现一个链表，提供最佳顺序存取，适合插入和移除元素。由这个类定义的链表也可以像栈或队列一样被使用。提供最佳顺序存取，适合插入和移除元素。特点是改快读慢  Set接口有HashSet 和TreeSet 两个实现类  HashSet 能够快速定位一个元素，  要注意的是：存入HashSet中的对象必须实现HashCode()方法；  TreeSet 将放入其中的元素按序存放。它实现的是SortedSet接口，也就是加入了对象比较的方法。通过对集中的对象迭代，我们可以得到一个升序的对象集合。  Map接口的实现类主要有HashMap 、HashTable和TreeMap；  当元素的顺序很重要时选用TreeMap，当元素不必以特定的顺序进行存储时，使用HashMap。HashMap不是同步的，Hashtable是同步的，但Hashtable不推荐使用，因为HashMap提供了所有类似的功能，并且速度更快。当需要在多线程环境下使用时，HashMap也可以转换为同步的。HashMap没法保证映射的顺序一直不变，但是作为HashMap的子类LinkedHashMap可以。  HashMap可以通过Map m = Collections.synchronizedMap（hashMap）来达到同步的效果。  问题22：Jdk1.8新特新  回答：  1、Java 8允许我们给接口添加一个非抽象的方法实现，只需要使用 default关键字即可；  2、Lambda 表达式（collection排序时使用）；  3、Person::new来调用Person类的构造函数，创建对象；  问题23：数据库比较  回答：首先主要是关系型数据库、nosql数据库和缓存数据库的区别；我觉的：  关系型数据库（mysql、Oracle）就想当于一个储物柜，里边分了各种小格子，把各种数据有序的保存在里边，查询时非常方便；有比较完善的sql语句；  mysql免费开源，自动提交事务，是基于表级别的一个事务，而Oracle不自动提交，是基于行级别的事务；mysql安装包小，使用简单，Oracle安装包较大，还会收费，更适合于大数据的处理、并发量高；  mysql用limit进行分页，第一个参数表示启始数，第二个参数表示要查询的条数。  oracle用伪列rownum来进行分页，因为伪列只能用小于，不能用大于，所以需要一个两层的嵌套查询。  mysql主键可以自增。Oracle必须用序列（sequence）；  字段类型有些区别；  mysql一般都是先有用户，然后创建库、创建表。而Oracle一般是，先创建数据库，然后创建临时表空间、物理表空间，再创建用户，创建表。  nosql数据库（mongdb）就相当于一个仓库，比如把衣服放在一堆，书本放在一堆，这样他的存储效率肯定很高，但是查找时，不方便精确查找。所以nosql更适合于数据量比较大，但是数据之间关系不是特别强，或者不是特别重要的数据；比如：电商里边的评论数据；  他的collection相当于数据库，document相当于表，document中数据主要以json格式存储；  缓存数据库（redis、memcached）：缓存数据库由于是保存在内存中的，所以他的存储效率非常高；但是他不适合持久化存储数据，尽管redis支持持久化，但是他也仅仅是在一定程度上保证了数据的安全性，还是有很多情况会造成数据丢失的，比如redis穿透、雪崩等。  所以他仅仅适合做为数据库和应用程序直接的中间层使用，不能作为真正的存储数据库使用。即使使用，也只能是一些不是特别重要的数据，比如购物车，在B2C的项目中，用户购物车一般都是临时存储，如果放入购物车长时间不提交，那么用户再购买的可能性很小了，所以即使丢了，也不会造成太多的损失。以这个，来换取项目的运行效率非常划算；  问题24：垃圾回收时的停顿现象  回答：垃圾回收的任务是识别和回收垃圾对象进行内存清理，为了让垃圾回收器可以更高效的执行，大部分情况下，会要求系统进入一个停顿的状态。停顿的目的是为了终止所有的应用线程，只有这样的系统才不会有新垃圾的产生。同时停顿保证了系统状态在某一个瞬间的一致性，也有利于更好的标记垃圾对象。因此在垃圾回收时，都会产生应用程序的停顿。  问题25：Tomcat配置调优  回答：tomcat优化我们主要从三方面优化，一是内存优化，tomcat内存优化主要是对tomcat启动参数优化，我们可以在 tomcat 的启动脚本catalina.sh 中设置 java\_OPTS 参数，通过Xms来初始化虚拟机最小内存Xmx来初始化虚拟机可使用的最大内存，  二是并发优化，通过配置server.xml里的Connector标签，在标签里配置maxThreads客户请求最大线程数、minSpareThreads 初始化时创建的socket线程数maxSpareThreads连接器的最大空闲socket线程数，connectionTimeout 连接的超时时间，minProcessors服务器创建时的最小处理线程数，maxProcessors服务器同时最大处理线程数，  三是缓存优化，配置compression打开压缩功能  问题26：直接缓冲区与非直接缓冲区别  回答：非直接缓冲区：通过 allocate() 方法分配缓冲区，将缓冲区建立在 JVM 的内存中  直接缓冲区：通过allocateDirect() 方法分配直接缓冲区，将缓冲区建立在物理内存中。可以提高效率  问题27：分散读取与聚集写入  回答：分散读取(scattering Reads)：将通道中的数据分散到多个缓冲区中，  聚集写入(gathering Writes)：将多个缓冲区的数据聚集到通道中  问题28：MySQL如何优化  回答：表的设计合理化(符合3NF)  添加适当索引(index) [四种: 普通索引、主键索引、唯一索引unique、全文索引  SQL语句优化  分表技术(水平分割、垂直分割)  读写分离  存储过程 [模块化编程，可以提高速度]  对mysql配置优化 [配置最大并发数my.ini, 调整缓存大小 ]  mysql服务器硬件升级  定时的去清除不需要的数据,定时进行碎片整理(MyISAM)  问题29：数据库三大范式  回答：第一范式：是对属性的原子性约束，要求属性(列)具有原子性，不可再分解；(只要是关系型数据库都满足  第二范式：是对记录的惟一性约束，表中的记录是唯一的, 就满足2NF, 通常我们设计一个主键来实现，主键不能包含业务逻辑。  第三范式：3NF是对字段冗余性的约束，它要求字段没有冗余。 没有冗余的数据库设计可以做到。  但是，没有冗余的数据库未必是最好的数据库，有时为了提高运行效率，就必须降低范式标准，适当保留冗余数据。具体做法是： 在概念数据模型设计时遵守第三范式，降低范式标准的工作放到物理数据模型设计时考虑。降低范式就是增加字段，允许冗余。  问题30：分表分库  回答：垂直拆分  垂直拆分就是要把表按模块划分到不同数据库表中（当然原则还是不破坏第三范式），这种拆分在大型网站的演变过程中是很常见的。当一个网站还在很小的时候，只有少量的人来开发和维护，各模块和表都在一起，当网站不断丰富和壮大的时候，也会变成多个子系统来支撑，这时就有按模块和功能把表划分出来的需求。其实，相对于垂直切分更进一步的是服务化改造，说得简单就是要把原来强耦合的系统拆分成多个弱耦合的服务，通过服务间的调用来满足业务需求看，因此表拆出来后要通过服务的形式暴露出去，而不是直接调用不同模块的表，淘宝在架构不断演变过程，最重要的一环就是服务化改造，把用户、交易、店铺、宝贝这些核心的概念抽取成独立的服务，也非常有利于进行局部的优化和治理，保障核心模块的稳定性  垂直拆分用于分布式场景。  水平拆分  上面谈到垂直切分只是把表按模块划分到不同数据库，但没有解决单表大数据量的问题，而水平切分就是要把一个表按照某种规则把数据划分到不同表或数据库里。例如像计费系统，通过按时间来划分表就比较合适，因为系统都是处理某一时间段的数据。而像SaaS应用，通过按用户维度来划分数据比较合适，因为用户与用户之间的隔离的，一般不存在处理多个用户数据的情况，简单的按user\_id范围来水平切分  通俗理解：水平拆分行，行数据拆分到不同表中， 垂直拆分列，表数据拆分到不同表中  问题31：水平分割案例  回答：思路:在大型电商系统中，每天的会员人数不断的增加。达到一定瓶颈后如何优化查询。  可能大家会想到索引，万一用户量达到上亿级别，如何进行优化呢？  使用水平分割拆分数据库表。  问题32：如何使用水平拆分数据库  回答：使用水平分割拆分表，具体根据业务需求，有的按照注册时间、取摸、账号规则、年份等  问题33：什么是慢查询  回答： MySQL默认10秒内没有响应SQL结果,则为慢查询  可以去修改MySQL慢查询默认时间  show variables like 'long\_query\_time';  问题34：什么是索引  回答：索引用来快速地寻找那些具有特定值的记录，所有MySQL索引都以B-树的形式保存。如果没有索引，执行查询时MySQL必须从第一个记录开始扫描整个表的所有记录，直至找到符合要求的记录。表里面的记录数量越多，这个操作的代价就越高。如果作为搜索条件的列上已经创建了索引，MySQL无需扫描任何记录即可迅速得到目标记录所在的位置。如果表有1000个记录，通过索引查找记录至少要比顺序扫描记录快100倍。  问题35：索引的分类  回答：主键索引、全文索引、唯一索引（不能出现重复的空字符串）、普通索引  问题63：索引的实现原理  回答：索引底层采用b-tree 折半查找，增加索引的目的是减少全表扫描。先取一个中间数，不如11个，中间数为6，比6小的放左边，必6大的放右边。  优点：增加程序效率  缺点：增加删除索引文件也需要更新。需要维护索引，增加内存。  问题36：那些列上适合添加索引  回答：  在where后面查询的  查询次数比较多  该字段的内容不是唯一的几个值  字段内容不是频繁变化。  执行计划查看;explain select \* from student where id=1; 可以查看是否有索引，如果type为all表示全表扫描查询，type为ref为普通索引，type为fulltext为全文索引  问题37：SQL优化技巧  回答：  如果使用组合索引，第一个不用和第二个一起作为条件查询，第二个需要和第一个一起查询索引才会生效  条件中加like，会造成全表扫描。不能加%%，第一个可以去掉。  使用or，条件必须都要加索引，否则索引会失效  使用group by 分组查询是，默认分组后，还会排序，可能会降低速度，  在group by 后面增加 order by null 就可以防止排序.  有些情况下，可以使用连接来替代子查询。因为使用join，MySQL不需要在内存中创建临时表  对查询进行优化，要尽量避免全表扫描，首先应考虑在 where 及 order by 涉及的列上建立索引  应尽量避免在 where 子句中对字段进行 null 值判断，不能写==null，可以写is null。否则将导致引擎放弃使用索引而进行全表扫描  最好不要给数据库留 NULL，尽可能的使用 NOT NULL 填充数据库.  备注、描述、评论之类的可以设置为 NULL，其他的，最好不要使用 NULL  可以在 num 上设置默认值 0，确保表中 num 列没有 null 值  不要用in和not in，也会全表扫描  对于数据量表较大的时候，使用缓存、分表和分页  问题38：MySQL数据引擎  回答：myisam 存储: 如果表对事务要求不高，同时是以查询和添加为主的，我们考虑使用myisam存储引擎. ,比如 bbs 中的 发帖表，回复表.  INNODB 存储: 对事务要求高，保存的数据都是重要数据，我们建议使用INNODB,比如订单表，账号表.  问题39：MyISAM 和 INNODB的区别  回答：1. 事务安全（MyISAM不支持事务，INNODB支持事务）  2. 查询和添加速度（MyISAM批量插入速度快）  3. 支持全文索引（MyISAM支持全文索引，INNODB不支持全文索引）  4. 锁机制（MyISAM时表锁，innodb是行锁）  5. 外键 MyISAM 不支持外键， INNODB支持外键. (在PHP开发中，通常不设置外键，通常是在程序中保证数据的一致)    问题40：Myisam注意事项  回答：如果你的数据库的存储引擎是myisam,请一定记住要定时进行碎片整理  问题41：ZooKeeper的Master选举  回答：在分布式环境中，相同的业务应用分布在不同的机器上，有些业务逻辑（例如一些耗时的计算，网络I/O处理），往往只需要让整个集群中的某一台机器进行执行，其余机器可以共享这个结果，这样可以大大减少重复劳动，提高性能，于是这个master选举便是这种场景下的碰到的主要问题。  利用ZooKeeper的强一致性，能够保证在分布式高并发情况下节点创建的全局唯一性，即：同时有多个客户端请求创建 /currentMaster 节点，最终一定只有一个客户端请求能够创建成功。利用这个特性，就能很轻易的在分布式环境中进行集群选取了。  另外，这种场景演化一下，就是动态Master选举。这就要用到EPHEMERAL\_SEQUENTIAL类型节点的特性了。上文中提到，所有客户端创建请求，最终只有一个能够创建成功。在这里稍微变化下，就是允许所有请求都能够创建成功，但是得有个创建顺序，于是所有的请求最终在ZK上创建结果的一种可能情况是这样： /currentMaster/{sessionId}-1 ,/currentMaster/{sessionId}-2,/currentMaster/{sessionId}-3 ….. 每次选取序列号最小的那个机器作为Master，如果这个机器挂了，由于他创建的节点会马上消失，那么之后最小的那个机器就是Master了。  1. 在搜索系统中，如果集群中每个机器都生成一份全量索引，不仅耗时，而且不能保证彼此之间索引数据一致。因此让集群中的Master来进行全量索引的生成，然后同步到集群中其它机器。另外，Master选举的容灾措施是，可以随时进行手动指定master，就是说应用在zk在无法获取master信息时，可以通过比如http方式，向一个地方获取master。  2. 在Hbase中，也是使用ZooKeeper来实现动态HMaster的选举。在Hbase实现中，会在ZK上存储一些ROOT表的地址和HMaster的地址，HRegionServer也会把自己以临时节点（Ephemeral）的方式注册到Zookeeper中，使得HMaster可以随时感知到各个HRegionServer的存活状态，同时，一旦HMaster出现问题，会重新选举出一个HMaster来运行，从而避免了HMaster的单点问题  问题42：分布式锁  回答：分布式锁，这个主要得益于 ZooKeeper 为我们保证了数据的强一致性。锁服务可以分为两类，一个是 保持独占，另一个是 控制时序。  1. 所谓保持独占，就是所有试图来获取这个锁的客户端，最终只有一个可以成功获得这把锁。通常的做法是把 zk 上的一个 znode 看作是一把锁，通过 create znode 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。  2. 控制时序，就是所有视图来获取这个锁的客户端，最终都是会被安排执行，只是有个全局时序了。做法和上面基本类似，只是这里 /distributelock 已经预先存在，客户端在它下面创建临时有序节点（这个可以通过节点的属性控制：CreateMode.EPHEMERALSEQUENTIAL 来指定）。Zk 的父节点（/distribute\_lock）维持一份 sequence, 保证子节点创建的时序性，从而也形成了每个客户端的全局时序。  问题43：生成订单号，保证幂等性  回答：  使用时间戳  使用UUID  推特 (Twitter) 的 Snowflake 算法——用于生成唯一 ID  问题44：数据库引擎特点  回答：InnoDB 支持事务，支持行级别锁定，支持 B-tree、Full-text 等索引，不支持 Hash 索引； MyISAM 不支持事务，支持表级别锁定，支持 B-tree、Full-text 等索引，不支持 Hash 索引； Memory 不支持事务，支持表级别锁定，支持 B-tree、Hash 等索引，不支持 Full-text 索引； NDB 支持事务，支持行级别锁定，支持 Hash 索引，不支持 B-tree、Full-text 等索引； Archive 不支持事务，支持表级别锁定，不支持 B-tree、Hash、Full-text 等索引；  问题45：zookepper应用场景  回答：  rpc远程调用框架dubbo+zookepper注册中心。  发布订阅，服务的生产者在zookepper上面注册，类似于mq里面的消息队列，服务消费者也是订阅zookepper上面的服务  负载均衡（权重，ip绑定和轮询（取模算法 调用次数/服务器集群数量=服务器位置））  分布式通知：应用在远程服务调用上，服务的生产者在zookepper上面注册，产生节点信息，服务消费者通过watcher事件监听到有新的接口注册到zookepper，那么消费者就可以远程调用生产者。  Master选举  Zookepper分布式锁  使用zookepper分布式配置中心  问题46：zookepper原理  Zookepper数据结构和xml类似，都是树状结构，有多个节点组成，节点名称是唯一的。  负载均衡：例如有两个用户模块注册到zookeeper里面，端口号不一致，我们的借款模块需要调用用户模块信息，就需要从注册中心中获取所有服务地址，通过本地负载均衡算法，选出一个服务地址，然后去调用。如果生产者有一个服务挂了，消费者会有事件监听，  问题47：mysql读写分离，主从同步怎么配置的，如何降低延迟  回答：  读写分离配置：在项目中我们配置了两个数据源，一个主数据库的，一个是从数据库的，在事务配置的时候，我们给从的数据dataSource配置了只读事务，在项目中根据操作判断注入不同的数据源，如果是增删改操作的注入主的数据源，所有的查询操作注入从的数据源，主从同步配置：首先找到主数据库的my.ini文件，在里边配置server-id 每个数据库的server-id不能重复要保证唯一, 配置binlog-do-db指明需要同步的数据库，log-bin来指明日志文件，配置完后，重启主数据库的服务，登陆主数据库，创建一个用于从数据库访问的用户，然后通过show master status;查询出主数据库的状态，记录下它的File和Position的值，紧接着配置从数据库，找到从数据库的my.ini文件，在里边配置server-id，log-bin和replicate-do-db 指明需要同步的数据库，然后重启从数据库的服务，重启完后登陆从数据库，创建从数据库对主数据库的连接，master\_log\_file要和刚才主数据库查询的File字段内容一致，master\_log\_pos要和主数据库刚才查出的Position一致，然后通过slave start，来启动从数据库，这样主从同步就配置好了。  如何降低延迟：  1.主库和同库尽可能在同一个局域网内，交换机网卡采用千兆网卡。  2.主数据库更新完成之后产生的操作日志不是瞬间产生的，我们可以通过设置sync(sen ke)\_binlog=1, 让它瞬间产生磁盘日志，（n=1指主数据库只要操作一次，就产生一次磁盘日志，n=10，就是操作10次，产生一次），从数据库可以依赖磁盘日志，瞬间产生同步，可以达到减低延迟的效果。  3. 设置主库和从库读取日志失败之后，及时重新建立连接，延迟缩短。  问题48：sql优化、数据库优化  回答：  数据库优化：  数据库优化吧我觉应该从硬盘、内存和网络带宽考虑，提高硬盘的读写速度，增大带宽提高吞吐量，增大  服务器内存，可以采用读写分离，降低单台数据库的访问压力，查询的时候控制数据量的大小，返回更少  数据，减少交互次数，减少cpu及内存的开销，  sql优化：如果一个表中数据量过大我们可以采用横切割，如果一个表中字段过多，我们可以采用纵切割，适度冗余  减少表关联查询，避免过多的联查，设计合理的表关系，适当建立索引，但是不能每个字段都建立索引，  因为索引也要占用一定的物理存储空间，而且索引也需要动态维护，增加索引虽然可以提高查询速度，但  是如果索引过多就会降低增删改的速度，写sql的时候也要注意，尽可能不要写一些让索引失效的sql，例  如：索引的字段不能为空，不能进项模糊搜索，不能进项逻辑运算，不能使用函数，给索引查询的值应是  已知数据，不能是未知字段值。也可以使用force(fou si)强制走索引，咱们常用的索引包括，单个索引、唯一索引、  复合索引，单个索引就是创建的索引中只包含一个字段，复合索引是创建的一个索引中包含多个字段，  创建索引：create index index\_name on table\_name(column\_name)  问题49：怎么防止sql注入  永远不要信任用户的输入。对用户的输入进行校验，可以通过[正则表达式](https://baike.baidu.com/item/%E6%AD%A3%E5%88%99%E8%A1%A8%E8%BE%BE%E5%BC%8F" \t "_blank)，或限制长度；对单引号和双"-"进行转换等。  （1）.使用PreparedStatement(pu pei er de si de te men ci)  sql注入只对sql语句的准备(编译)过程有破坏作用而PreparedStatement(pu pei er de si de te men ci)已经准备好了,执行阶段只是把输入串作为数据处理, 而不再对sql语句进行解析,准备,因此也就避免了sql注入问题.  (2).使用正则表达式过滤传入的参数  (3).字符串过滤  (4).jsp中调用该函数检查是否包函非法字符  (5).JSP页面判断代码 ”‘”,”\\”,”/”  2.永远不要使用动态拼装sql，可以使用参数化的sql或者直接使用[存储过程](https://baike.baidu.com/item/%E5%AD%98%E5%82%A8%E8%BF%87%E7%A8%8B" \t "_blank)进行数据查询存取。  3.永远不要使用[管理员](https://baike.baidu.com/item/%E7%AE%A1%E7%90%86%E5%91%98" \t "_blank)权限的数据库连接，为每个应用使用单独的权限有限的数据库连接。  4.不要把机密信息直接存放，加密或者hash(ha 史)掉密码和敏感的信息。  5.应用的异常信息应该给出尽可能少的提示，最好使用自定义的[错误信息](https://baike.baidu.com/item/%E9%94%99%E8%AF%AF%E4%BF%A1%E6%81%AF" \t "_blank)对原始错误信息进行包装  6.sql注入的检测方法一般采取辅助[软件](https://baike.baidu.com/item/%E8%BD%AF%E4%BB%B6" \t "_blank)或网站平台来检测，软件一般采用sql注入检测工具jsky，网站平台就有亿思[网站安全](https://baike.baidu.com/item/%E7%BD%91%E7%AB%99%E5%AE%89%E5%85%A8" \t "_blank)平台检测工具。MDCSOFT SCAN等。采用[MDCSOFT-IPS](https://baike.baidu.com/item/MDCSOFT-IPS" \t "_blank)可以有效的防御SQL注入，XSS攻击等。  问题50：redis好处  回答：(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，zset，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  问题51：redis在项目中遇到的问题  回答：1).缓存穿透的问题:  一般出现这样的问题，是因为当我们查询一条肯定不存在的数据的时候，缓存中没有，就会透过缓存来查询数据库，数据库也不存在，这样就会将值保存在缓存中最后还是缓存和数据库中都没有，如果一直访问这条数据。我们就对数据库进行频繁的查询给数据库带来压力  解决办法：当查询的时候，如果缓存和数据库中都没有，我们就将这个数据以空的形式存放在缓存中，（或者是给一个false的标示）这样就不用去数据库就可以知道不存在，减少对数据库查询的次数。当我们这个值发生改变的时候，我们在重新赋值  2.并发情况:  当我们大量访问都是查询一个缓存中没有的数据时，这样就会去数据库中进行查询，可能会造成数据库宕机，  解决办法：在查询的时候，我给它添加一个同步锁，只有第一条数据去，数据库中查询并返回到redis中后才能查询，这是数据库中已近存在了值，这样可以避免  3.雪崩:  大量数据的访问缓存超时，这样用户就会访问到数据库，第一台数据库崩溃了，访问就会到第二台数据库进行查询，这样就会导致额第二台崩溃。  解决办法：就是设置失效时间，不要一起失效，或者是设置在访问少的时候，或者设置永远不失效。  问题52：redis持久化  redis的一大特点就是可以将数据进行持久化，在一定程度上确保了数据的安全性，但不是绝对的；  首先持久化分为rdb（快照持久化）和aof（精细持久化）；  快照持久化，是默认开启的；会自动保存数据，当启动时会在文件夹中生成dump(荡 pu).rdb文件；存放持久化后的数据；  当然我们也可以设置持久化的频率，在redis.conf文件中通过save进行设置，默认有三种情况，每秒超过一万数据或每5分钟有10条数据的时候再或者每15分钟有1条记录，都会执行快照持久化，  当然也可以通过bgsave的方法来手动进行一个快照持久化；（也可以通过ip和端口号就给别人进行手动持久化）；  如果频繁的快照持久化，会降低性能和效率，  但是这样也出现了一个问题，就是当一分钟内如果有一万条数据时，不会提交，但是在下一次提交之前，停电了，这样就会丢失掉这些数据；  当时想到的解决方法呢就是和（AOF）精细持久化进行一个结合，达到一个秒级的持久化；  这个持久化需要我们手动进行开启，（注意，AOF开启之后，之前的数据都会丢失，所以要在一开始使用时就要配置好）开启的方法就是在配置redis.conf，将appendOnly 改为yes；同时还可以更改文件名称；然后重新启动服务，这时精细化持久化就启动好了  appendfsync always     #每次有数据修改发生时都会写入AOF文件。  appendfsync everysec  #每秒钟同步一次，该策略为AOF的缺省策略。  appendfsync no          #从不同步。高效但是数据不会被持久化。  RDB持久化是指在指定的时间间隔内将内存中的数据集快照写入磁盘，实际操作过程是fork一个子进程，先将数据集写入临时文件，写入成功后，再替换之前的文件，用二进制压缩存储。  AOF持久化以日志的形式记录服务器所处理的每一个写、删除操作，查询操作不会记录，以文本的方式记录，可以打开文件看到详细的操作记录。  问题53：redis集群  Redis集群  首先下载linux版本的redis安装包，解压，找到redis的conf文件，配置端口号，配置要开启的数据库，看有几个库，然后再配置redis开启后日志存储的位置，数据的存储位置，支持后台运行模式，在配置下开启集群模式，完成之后，保存，复制几分，然后启动，多启动几台单台服务，一般都是一组一组，我们一般都配置六台，总共三组，所有单台服务启动完毕后，通过redis-trib(踹 bu).rb create --replicas(re pu li kai si) 1 127.0.0.1:7000 127.0.0.1:7001 127.0.0.1:7002 127.0.0.1:7003 127.0.0.1:7004 127.0.0.1:7005  Redis 集群  Redis 在3.0版本前只支持单实例模式，虽然支持主从模式、哨兵模式部署来解决单点故障  Redis 在 3.0 版本以后就推出了集群模式。  Redis 集群搭建规划，由于集群至少需要6个节点（3主3从模式）  我们计划集群中 Redis 节点的端口号为 9001-9006 ，端口号即集群下各实例文件夹。数据存放在 端口号/data 文件夹中。  2.复制执行脚本  在 /usr/local/redis-cluster 下创建 bin 文件夹，用来存放集群运行脚本，并把安装好的 Redis 的 src 路径下的运行脚本拷贝  3.复制一个新 Redis 实例  我们现在从已安装好的 Redis 中复制一个新的实例到 9001 文件夹，并修改 redis.conf 配置。修改 redis.conf 配置和单点唯一区别是下图部分，其余还是常规的这几项：  port 9001（每个节点的端口号）  daemonize yes bind 192.168.119.131（绑定当前机器 IP）  dir /usr/local/redis-cluster/9001/data/（数据文件存放位置）  pidfile /var/run/redis\_9001.pid（pid 9001和port要对应）  cluster-enabled yes（启动集群模式）  cluster-config-file nodes9001.conf（9001和port要对应）  cluster-node-timeout 15000  appendonly yes  4. 再复制出五个新 Redis 实例  我们已经完成了一个节点了，其实接下来就是机械化的再完成另外五个节点，其实可以这么做：把 9001 实例 复制到另外五个文件夹中，唯一要修改的就是 redis.conf 中的所有和端口的相关的信息即可，其实就那么四个位置。开始操作，看图：  调用 ruby 命令来进行创建集群，--replicas 1 表示主从复制比例为 1:1，即一个主节点对应一个从节点；然后，默认给我们分配好了每个主节点和对应从节点服务，以及 solt 的大小，因为在 Redis 集群中有且仅有 16383 个 solt ，默认情况会给我们平均分配，当然你可以指定，后续的增减节点也可以重新分配。  问题54：redis缓存失效  回答：  1、设置过期时间  expire key time(以秒为单位)--这是最常用的方式  setex(String key, int seconds, String value)--字符串独有的方式  注意：  除了字符串自己独有设置过期时间的方法外，其他方法都需要依靠expire方法来设置时间  如果没有设置时间，那缓存就是永不过期  如果设置了过期时间，之后又想让缓存永不过期，使用persist key  2、三种过期策略  定时删除  含义：在设置key的过期时间的同时，为该key创建一个定时器，让定时器在key的过期时间来临时，对key进行删除  优点：保证内存被尽快释放  缺点：  若过期key很多，删除这些key会占用很多的CPU时间，在CPU时间紧张的情况下，CPU不能把所有的时间用来做要紧的事儿，还需要去花时间删除这些key  定时器的创建耗时，若为每一个设置过期时间的key创建一个定时器（将会有大量的定时器产生），性能影响严重  没人用  惰性删除  含义：key过期的时候不删除，每次从数据库获取key的时候去检查是否过期，若过期，则删除，返回null。  优点：删除操作只发生在从数据库取出key的时候发生，而且只删除当前key，所以对CPU时间的占用是比较少的，而且此时的删除是已经到了非做不可的地步（如果此时还不删除的话，我们就会获取到了已经过期的key了）  缺点：若大量的key在超出超时时间后，很久一段时间内，都没有被获取过，那么可能发生内存泄露（无用的垃圾占用了大量的内存）  定期删除  含义：每隔一段时间执行一次删除过期key操作  优点：  通过限制删除操作的时长和频率，来减少删除操作对CPU时间的占用--处理"定时删除"的缺点  定期删除过期key--处理"惰性删除"的缺点  缺点  在内存友好方面，不如"定时删除"  在CPU时间友好方面，不如"惰性删除"  难点  合理设置删除操作的执行时长（每次删除执行多长时间）和执行频率（每隔多长时间做一次删除）（这个要根据服务器运行情况来定了） |
| **人资问题(至少30个)** |
| 问题1：为什么的年龄和身份证不一致呢？  回答：爸妈在县城上班，爷爷奶奶带大的，后来上幼儿园才上的户口  问题2：你好，请先做一下自我介绍  回答：我上一份工作是在北京新网公司就职，担任JAVA工程师一职位，我们上家单位是一家外包单位，我们的客户主要有做金融理财的和企业ERP这两类，我在我们上家公司一共参与了4个项目的设计、研发和后期维护，我近期做的这个项目呢是一个金融类的APP,近期这个项目呢，我们是从去年12月份就开始开发了，今年11月份项目验收的，开发这个项目的时候我们一块投入了五个人，我在里面担任的是后台开发，这个项目采用的环境是（IDEA、JDK1.8、Tomcat、MySQL、SVN）；用到的技术有（springBoot、maven、redis、solr、spring）；主要核心模块有（介绍一些核心模块【一定要高大上】）；我在这个项目中呢担任JAVA开发职位；主要职责是对功能的设计和开发工作；开发的功能有（登陆、再介绍一下其他做的模块）还有就是完成一些领导交代的其他任务，这是我近期的这个项目的情况。  问题3：目前离职多久了？回答：  回答：上周3离职的  问题4：离职到现在这段时间是一直在找工作吗？  回答：这周一开始找的，我在家整理了一些面试的资料  问题5：目前住在哪个，离的远吗？  回答：霍家营新干线小区  问题6：学历是本科吗，学信网能查吗？  回答：是本科，民教网可查  问题7：具体工作多长时间了，几份工作经历?  回答：3年多，在太原干了一年，在北京这边待了2年多了  问题8：介绍一下智联天下公司的工作内容？  回答：就是后台开发自己的模块，完成领导交代的任务  问题9：介绍一下的情况，负责的内容？  回答：我在上家公司主要负责后端程序开发  问题10：在这家公司工作多长时间？  回答：2年多吧  问题11：离职的原因？  回答：就是项目期到了，公司没有接到新的项目，而且基本都是驻场开发  问题12：是主动离职还是被动离职？  回答：算是主动离职的  问题13：公司是做什么业务的？  回答：主要做金融，还有企业erp  问题14：在这家公司主要负责的项目是？  回答：上个项目是一个p2p的金融个人贷  问题15：这个项目具体的负责的内容  回答：满标一审，满标二审，生成还款信息  问题16：这个项目几个前端，如何分工作？  回答：2个前端，4个后端  问题17：项目已经上线了吗?  回答：还没有，还在测试  问题18：除了这个项目，还负责其它的项目吗  回答：有啊，做过一个广告项目  问题19：这家公司在什么地方？  回答：北京市海淀区农大南路1号院2号楼2层办公  问题20：未来3年的职业规划？  回答：继续专研技术  问题21：对下一份工作的内容，环境，领导有什么要求吗？  回答：最好是部分内容与我曾今用过的东西相符合，这样更容易上手，办公环境，有座位就行。领导的话，可以和我们打成一片吧  问题22：你的性格（优缺点）？  回答：优点：对待任何东西都很乐观 缺点：在工作生活中可能缺乏一定魄力，比如在使用一个新技术的时候  问题23：加班可以接受吗？  回答：加班的话，程序员很少有不加班的，在项目期比较紧的时候，肯定是需要加班的，但平时的话还是希望按时上下班的  问题24：期望的薪资是多少？  回答：13K  问题25：上份工作的薪资是多少？  回答：10.5K  问题26：上份薪资是税前还是税后？  回答：税后  问题27：有其它什么补助吗?  回答：车补，我在天津那边管住  问题28：薪资结构呢？  回答：基本工资+绩效，因为采用了合理避税  问题29：缴纳五险一金了吗？  回答：这个没有  问题30：我们会有一个复试，还有什么问题需要问吗？  回答：好，大概问哪些方面的问题了 |
|  |