1. 解决哈希冲突方法？（4种）

答案：1、开放地址法：当哈希冲突时，就去寻找下一个空的散列地址，只要散列表足够大，空的散列地址总能找到，并将记录存入   
公式为：fi(key) = (f(key)+di) MOD m (di=1,2,3,……,m-1) 的位置。2、再哈希法：当哈希冲突时，用第二个哈希算法、第三个哈希算法……知道找到一个不冲突的地址。3、链地址法：数组加链表，当哈希冲突时，用链表的next指针链接起来就可以了。 4、创建共享溢出区，将哈希表分为基本表和溢出表，当发生冲突时，直接存到溢出表里面就行。

1. 读写分离常用代理方式来实现，代理服务器接收应用层传来的读写请求，然后决定转发到哪个服务器。
2. 可以使用 Redis 自带的 SETNX 命令实现分布式锁，也可采用zookeeper临时有序节点做分布式锁。使用 SETNX（set if not exist）指令插入一个键值对，如果 Key 已经存在，那么会返回 False，否则插入成功并返回 True。SETNX 指令和数据库的唯一索引类似，保证了只存在一个 Key 的键值对，那么可以用一个 Key 的键值对是否存在来判断是否存于锁定状态。EXPIRE 指令可以为一个键值对设置一个过期时间，从而避免了数据库唯一索引实现方式中释放锁失败的问题。数据库唯一索引：获得锁时向表中插入一条记录，释放锁时删除这条记录。唯一索引可以保证该记录只被插入一次，那么就可以用这个记录是否存在来判断是否处于锁定状态。

系统运维：

1. Jvm调优原则？

答案：多数的java应用不需要在服务器上进行GC调优，多数导致GC问题的java应用，都不是因为我们参数设置错误，而是代码问题，在应用上线之前，先考虑将机器的JVM参数设置到最优（最合适），减少创建对象的数量，减少使用全局变量和大对象，GC优化是到最后不得已才采用的手段，在实际应用中，分析GC情况优化代码比优化GC参数要多得多。

1. Jvm调优目的？

答案：将转移到老年代的对象数量降低到最小，减少GC的执行时间。

1. java常用排查工具？

答案：top:linux自带的命令，乐意查看系统资源消耗情况，可以看看CPU，内存，SWAP，IO的消耗情况，需要特别注意的有几个值：ni：这个值如果特别高，表示线程上下文切换开销比较大，看是不是开了太多的线程导致的。Res:表示了进程实际占用的内存，swap:内存不足就会占用swap空间，这个时候一般应用的性能会急剧降低，需要特别关注。jstack,jmap,jvisualvm：这三个是jvm 自带的一些工具，必须熟练掌握。

**Jstack:jstack命令主要用于生成虚拟机当前时刻的线程快照**。线程快照就是当前虚拟机内每一条线程正在执行的方法堆栈的集合，生成快照的目的主要是定位线程长时间出现停顿的原因，如线程死锁，死循环，请求外部资源导致的长时间等待等都是导致线程长时间停顿的原因。线程出现停顿的时候通过jstack来查看各个线程的调用堆栈，就可以知道没有响应的线程到底在后台做些什么事情，或者在等待些什么资源。通过jps得到代码的id之后可以用jstack id(刚刚得到的)。（2）**jmap:命令用于生成堆转储快照，**jmap的作用并不仅仅是为了获取dump文件，他还可以查询finalizer执行队列，java堆和永久代的详细信息，如空间使用率，当前使用的是哪种收集器等，将指定的应用程序的快照输出到桌面，然后，可以通过jhat，visual VM等工具分析该堆文件，看看有哪些大对象，或者哪些类的实例特别多。**Jhat分析**headdump文件：他会建立一个HTTP/HTML服务器，让用户可以在浏览器上查看分析结果。访问<http://localhost:7000/> （3）**jvisualvm:提**供在java虚拟机上运行的java应用程序的详细信息，在visualvm的图形用户界面中，您可以很方便、快捷地查看多个java应用程序的相关信息。功能有：1.显示虚拟机进程以及进程的配置、环境信息（类似jps、jinfo）。2.监视应用程序的CPU、GC、堆、方法区及线程信息（jstat、jstack）。3.dump生成以及分析堆转储快照（jmap、jhat）。4.方法级的程序运行性能分析，找到被调用最多，运行时间最长的方法。5.离线程序快照：收集程序的运行时配置，线程dump、内存dump等信息建立一个快照，可以将快照发送开发者出进行bug反馈。

还有jps，jinfo、jstat，jps：类似于Unix的ps命令，用户查看所有的java进程的启动类，传入参数和java虚拟机参数等信息。显示虚拟机执行主类名称以及这些进程的本地虚拟机唯一ID。Jps –q:只输出进程的本地虚拟机唯一ID，jps –l:输出主类的全名，如果进程执行的是jar包，输出jar路径。Jps –v:输出虚拟机进程启动时JVM参数。 Jps –m:输出传递给java进程的main()函数的参数。**Jinfo:**实时查看和调整虚拟机各项参数。**Jstat:**监视虚拟机各种运行状态信息，他可以显示本地或者远程虚拟机的类信息，内存、垃圾收集、JIT编译等运行数据，在没有GUI，只是纯文本控制台环境的服务器上，**他将是运行期间定位虚拟机性能问题的首选工具。**

1. 系统日志中找出error日志

答案：grep命令，grep命令是一款强大的文本搜索工具，他能使用特定模式匹配（包括正则表达式）搜索文本，并默认输出匹配行，命令格式：grep string filename是从filename中找string，这里从系统日志文件中找error日志：grep error \*\*\*.log，例：find . –name “.log” | grep –i error | grep –vi “info” 是使用find –name来列出所有log文件，然后重定向给grep，然后使用grep –i error来查找所有包含error的行，使用grep –vi来查找不包含info的行。grep过滤多个条件是使用grep –E命令，如grep –E ‘113|abc’ filename，找出文件filename中包含123或者abc的行，中间用“|”管道符分开。

1. **自旋锁**的思想是让一个线程在请求一个共享数据的锁时执行忙循环（自旋）一段时间，如果在这段时间内能获得锁，就可以避免进入阻塞状态。

自旋锁虽然能避免进入阻塞状态从而减少开销，但是它需要进行忙循环操作占用 CPU 时间，它只适用于共享数据的锁定状态很短的场景，在 JDK 1.6 中引入了自适应的自旋锁。自适应意味着自旋的次数不再固定了，而是由前一次在同一个锁上的自旋次数及锁的拥有者的状态来决定。

**锁消除**是指对于被检测出不可能存在竞争的共享数据的锁进行消除。

锁消除主要是通过逃逸分析来支持，如果堆上的共享数据不可能逃逸出去被其它线程访问到，那么就可以把它们当成私有数据对待，也就可以将它们的锁进行消除。

如果一系列的连续操作都对同一个对象反复加锁和解锁，频繁的加锁操作就会导致性能损耗。**锁粗化**将会把加锁的范围扩展（粗化）到整个操作序列的外部。

偏向锁：只有一个线程的情况下，如果一个线程反复访问共享资源，就把线程id记录下来，以后再访问的时候不用加锁，直接访问，但是一旦出现第二个线程偏向锁就不行了，退化为CAS，CAS的例子就举自旋锁就行。

AQS：请求共享资源state（是volatile的），有虚拟队列存储暂时得不到CAS锁的线程，AQS使用CAS对该同步状态进行原子操作实现对其值的修改。AQS对资源的共享有两种方式：共享（semaphore,countdownlatch，CyclicBarrier等），和独占（synchronized，reentrantlock）。

AtomicInteger 类主要利用 CAS (compare and swap) + volatile 和 native 方法来保证原子操作，从而避免 synchronized 的高开销，执行效率大为提升。

1. **MySQL集群高可用？**

**答案：在服务器架构时，为了保证服务器不宕机在线状态，需要为每台单点服务器提供冗余机，对于写服务器来说，需要提供一台同样的写-冗余服务器，当写服务器健康时（写-冗余通过心跳检测），写-冗余作为一个从机的角色复制写服务器的内容与其做一个同步，当写服务器宕机时，写-冗余服务器便顶上来作为写服务器继续提供服务，对外界来说这个处理过程是透明的，即外界仅通过一个IP访问。**

1. 接口和抽象类的区别是什么？

接口的方法默认是 public，所有方法在接口中不能有实现(Java 8 开始接口方法可以有默认实现），而抽象类可以有非抽象的方法。

接口中除了 static、final 变量，不能有其他变量，而抽象类中则不一定。

一个类可以实现多个接口，但只能实现一个抽象类。接口自己本身可以通过 extends 关键字扩展多个接口。

接口方法默认修饰符是 public，抽象方法可以有 public、protected 和 default 这些修饰符（抽象方法就是为了被重写所以不能使用 private 关键字修饰！）。

从设计层面来说，抽象是对类的抽象，是一种模板设计，而接口是对行为的抽象，是一种行为的规范。

1. @autowire和@resource的区别：

答案：@Autowired与@Resource都可以用来装配bean，但@autowire默认按类型（byType）装配,如果想按名称（byName）导入就得配合使用**@Qualifier注解**。@resource默认按名称装配，名称可以通过name属性进行指定，如果没有指定name属性，当注解写在字段上时，默认取字段名进行名称查找。如果注解写在setter方法上默认取属性名进行装配。当找不到与名称匹配的bean时才按照类型进行装配。但是需要注意的是，如果name属性一旦指定，就只会按照名称进行装配。推荐使用@Resource注解在字段上，这样就不用写setter方法了，

1. Redis 主从复制的过程：（1）主服务器创建快照文件，发送给从服务器，并在发送期间使用缓冲区记录执行的写命令。快照文件发送完毕之后，开始向从服务器发送存储在缓冲区中的写命令；（2）从服务器丢弃所有旧数据，载入主服务器发来的快照文件，之后从服务器开始接受主服务器发来的写命令；（3）主服务器每执行一次写命令，就向从服务器发送相同的写命令。

随着负载的不断上升，主服务器可能无法很快的实现所有从服务器的复制过程，为了解决这个问题可以采用一个中间层来分担主服务器的复制工作，中间层的服务器是最上层服务器的从服务器，又是最下层服务器的主服务器。

redis有三种集群方式：主从复制，哨兵模式和集群。

**1.主从复制**

**主从复制原理：**

* 从服务器连接主服务器，发送SYNC命令；
* 主服务器接收到SYNC命名后，开始执行BGSAVE命令生成RDB文件并使用缓冲区记录此后执行的所有写命令；
* 主服务器BGSAVE执行完后，向所有从服务器发送快照文件，并在发送期间继续记录被执行的写命令；
* 从服务器收到快照文件后丢弃所有旧数据，载入收到的快照；
* 主服务器快照发送完毕后开始向从服务器发送缓冲区中的写命令；
* 从服务器完成对快照的载入，开始接收命令请求，并执行来自主服务器缓冲区的写命令；（**从服务器初始化完成**）
* 主服务器每执行一个写命令就会向从服务器发送相同的写命令，从服务器接收并执行收到的写命令（**从服务器初始化完成后的操作**）

1. Sentinel（哨兵模式）可以监听集群中的服务器，并在主服务器进入下线状态时，自动从从服务器中选举出新的主服务器（投票原则，大于一半以上就可以）。
2. Redis所有缓存数据的存储受可用内存大小的限制，形成木桶效应，所以可以采用客户端分片、代理分片、服务器分片：

* 客户端分片：客户端使用一致性哈希等算法决定键应当分布到哪个节点。
* 代理分片：将客户端请求发送到代理上，由代理转发请求到正确的节点上。
* 服务器分片：Redis Cluster

1. **MySQL与Redis数据不一致怎么处理？不管是先写MySQL数据库，再删除redis缓存，还是先删除缓存，再写库，都有可能出现数据不一致的情况。举例子，如果删除了缓存redis，还没有来得及写库MySQL，另一个线程就来读取，发现缓存为空，则去数据库中读取数据写入缓存，此时缓存中为脏数据。或者先写了库，在删除缓存前，写库的线程宕机了，没有删除掉缓存，则也会出现数据不一致的情况。因为读和写是并发的，没办法保证顺序，就会出现缓存和数据库的数据不一致的情况。**

答案：第一种方案：采用延时双删策略，在写库前都设置redis.del(key)操作，并且设置合理的超时时间。具体步骤：1.先删除缓存，2.再写数据库 3.休眠500毫秒 4.再次删除缓存。具体需要休眠多久呢？需要评估自己的项目的读数据业务逻辑的耗时，这么做的目的，就是确保读请求结束，写请求可以删除读请求造成的缓存脏数据。当然，这种策略还要考虑redis和数据库主从同步的耗时，最后的写数据的休眠时间，则在读数据业务逻辑的耗时基础上，加几百ms即可。设置合理的缓存过期时间：从理论上来说，给缓存设置过期时间，是保证最终一致性的解决方案。所有的写操作以数据库为准，只要到达缓存过期时间，则后面的读请求自然会从数据库中读取新值然后回填缓存。

该方法的弊端，结合双删策略+缓存过期时间，这样最差的情况就是在超时时间内数据存在不一致，而且又增加了写请求的耗时。

**第二种方案：异步更新缓存（基于订阅binlog的同步机制），技术整体思路：MySQL binlog增量订阅信息+消息队列+增量数据更新到redis。**具体的步骤：1.读redis：热数据基本都在redis中 2.写MySQL：增删改都是操作MySQL 3.更新redis数据：MySQL的数据操作binlog 来更新到redis（MySQL数据的增删改命令都会被存在binlog,且通过二进制的形式存的）然后通过binlog来更新redis。

Redis更新：数据操作主要分为两大块，一个是全量（将全部数据一次写入到redis）另一个是增量（实时更新），这里说的增量，指的是MySQL的update、insert、delete变更数据。读取binlog后分析，利用消息队列，推送更新各台的redis缓存数据，这样一旦MySQL中产生了新的写入、更新、删除等操作，就可以把binlog相关的消息推送到redis、redis再根据binlog中的记录，对redis进行更新，其实这种机制**，很类似MySQL的主从备份机制，因为MySQL的主备也是通过binlog来实现的数据一致性。这里可以结合使用canal（阿里的一款开源框架），通过该框架可以对MySQL的binlog进行订阅，而canal正是模仿了MySQL的slave数据库的备份请求，使得redis的数据更新更新达到了相同。当然，也可以使用第三方，kafka,rabbitMQ等来实现推送更新redis。**

1. **如何实现MySQL读写分离？**

答案：读写分离，把读操作分散到从服务器中，并且对多个从服务器能实现负载均衡，1. MaxScale可以帮我们实现读写分离的同时负载均衡功能。MaxScale目前提供的插件功能分为5类，1.认证插件：提供了登陆认证功能，Maxscale会读取并缓存数据库中user表中的信息，当有连接进来时，先从缓存信息中进行验证，如果没有此用户，会从后端数据库中更新消息，再次进行验证。 2.协议插件：包括客户端连接协议，和连接数据库的协议。 3.路由插件：决定如何把客户端的请求转发给后端数据库服务器，读写分离和负载均衡的功能就是由这个模块实现的。 4.监控功能：对各个数据库服务器进行监控，例如发现某个数据库服务器响应很慢，那么就不向其转发请求了。 5.日志和过滤插件：提供简单的数据库防火墙功能，可以对SQL进行过滤和容错。

配置方式：配置前需要在master中为Maxscale创建两个用户，用于监控模块和路由模块。4006是read-write listener使用的端口，用于连接Maxscale，6603是maxadmin listener使用的端口，用于Maxscale管理器。

2. 使用MySQL的com.mysql.jdbc.ReplicationDriver驱动

3. 构造不同的数据库连接池，在service层通过判断方法的读写类型，读方法使用从数据库连接池，写方法使用主数据库连接池。

4. 使用阿里的mycat或者360的Atlas来实现分库分表，读写分离和负载均衡等。

**5. 使用Spring AOP解决数据源切换的问题，那么就可以和mybatis，Druid整合到一起了，我们在整合spring和mybatis时，我们只需写dao接口和对应的SQL语句，Spring创建DAO实例，他通过我们注入的数据源，帮我们完成从中获取数据库连接、使用连接执行SQL语句的过程以及最后归还连接给数据源的过程。如果我们能在调用dao接口时根据接口方法动态地选择数据源，就可以做到读写分离了。实现动态切换数据源主要依赖spring-aop和spring aspects,其中RoutingDataSourceImpl是实现动态切换功能的核心类。**

**3．如果发现主库宕机了，那么让备库先执行完所有其从主库上获取到的中继日志，然后再执行stop slave，change master to master\_host=’ ’，然后再执行reset\_slave，使其断开与老主库的连接，并丢弃master.info里面记录的信息，然后执行show master status记录新主库的二进制日志坐标，将客户端重新连接到新主库上，在每台备库上执行change master t0语句，使用之前通过show master status获取到的二进制日志坐标，指向新的主库。**

1. **使用Nginx做负载均衡？**

**答案：将多台服务器组成一个集群，当用户访问到一个转发服务器，再由转发服务器将访问分发到压力更小的服务器。Upstream定义负载节点池，使用Nginx做负载均衡的两大模块，location模块进行URI匹配，proxy模块发送请求给upstream定义的节点池。Nginx实现负载均衡的策略有五种：1.轮询：每个请求按时间顺序逐一分配到不同的后端服务器，如果后端某个服务器宕机，能自动剔除故障系统。 2.权重weight：weight的值越大分配到的访问概率越高，主要用于后端每台服务器性能不均衡的情况下，其次是在主从的情况下设置不同的权值，达到合理有效的利用主机资源。 3.ip\_hash(ip绑定)：每个请求按访问IP的哈希结果被分配，使来自同一个IP的访客固定访问一台后端服务器，并且可以有效解决动态网页存在的session共享问题 4.fair（第三方插件）必须安装upstream\_fair模块，对比weight、ip\_hash更加智能的负载均衡算法，fair算法可以根据页面大小和加载时间长短智能地进行负载均衡，响应时间短的优先分配。哪个服务器的响应速度快，就将请求分配到那个服务器上。 5.url\_hash(第三方插件)：必须安装Nginx的hash软件包，按访问URL的hash结果来分配请求，使每个URL定向到同一个后端服务器，可以进一步提高后端缓存服务器的效率。**

1. **慢查询分析?**

**答案：1.开启慢查询日志，设置阈值，比如超过5分钟的就是慢SQL，并将它抓取出来。2.explain+慢SQL分析 3.show profile查询SQL在MySQL服务器里面的执行细节和生命周期情况。 4. SQL数据库服务器的参数调优。原则是小表驱动大表，group by关键字优化：group by实质上是先排序后进行分组，遵照索引创建的最佳左前缀；当无法使用索引列，增大max\_length\_for\_sort\_data参数的设置+增大sort\_buffer\_size参数的设置；where高于having，能写在where限定的条件就不要用having限定了。**

1. **索引优化？**

**答案：1.保持独立列，不要用函数和运算，这样会导致索引失效而进行全表扫描，2.避免隐式转换，当查询条件左右两侧类型不匹配的时候发生隐式转换，隐式转换带来的影响就是可能导致索引失效进而进行全表扫描，3. 索引不会包含有null的值，设计多列复合索引时一定要注意，所有列必须不能为null，因为含有null的列是不会被加入索引的，直接无效 4.避免使用or连接，同列or可以转换为in，innodb对or的查询条件会运行多次单独的查询然后合并结果，时间复杂度为o(n)，而对于in操作，innodb使用的是基于树的二分查找，时间复杂度是o(logn),显然log更快。 5.适当利用覆盖索引加速，如果要查询的数据都在复合索引列中，那么就可以使用覆盖索引，直接查询到结果返回，省去“回表”，但要注意别把索引搞得太多或者太长，维护索引会降低性能。 6.正确认识最左前缀匹配索引，一定要注意从最左开始匹配才能利用上索引，一旦使用前缀匹配，那么将不能使用覆盖索引，也不能利用索引来排序或者group 7.复合索引顺序很重要，复合索引第一列不需要再次建立单列索引，order by和group by后面接的列与复合索引顺序一致时可以利用复合索引避免排序和临时表 8.范围查询会导致右侧条件索引失效，查询中的某个列有范围查询，则其右边所有列都无法使用索引优化查找。**

**JVM调优步骤**

一般情况下，JVM 调优可通过以下步骤进行：

 分析 GC 日志（打开PrintGC）及 dump （Jmap命令）文件（jhat分析dump文件），判断是否需要优化，确定瓶颈问题点；

 确定 JVM 调优量化目标；

 确定 JVM 调优参数（根据历史 JVM 参数来调整）；

 依次调优内存、延迟、吞吐量等指标；

 对比观察调优前后的差异；

 不断的分析和调整，直到找到合适的 JVM 参数配置；

 找到最合适的参数，将这些参数应用到所有服务器，并进行后续跟踪。

1. JVM怎么调优？

答案：1.打开PrintGC，查看gc日志（-XX：+PrintGC -XX：+PrintGCDetails（打印gc详情） -XX：+PrintGCTimeStamps（打印gc停顿耗时） -XX：+PrintGCDateStamps（打印GC的时候添加事件标志）），输出的log里头99%都是GC（Allocation Failure）造成的young gc，allocation failure表示向young generation（Eden）给新对象申请空间，但是young generation（Eden）区剩余的空间不够所需的大小，所以导致的minor gc，再看有没有发生full/major gc 2.检查gc日志和系统日志(也可以用GC日志可视化工具：GC easy：上传gc的log文件就可以， GCViewer) 3.调整堆大小，调整GC策略（如果不会调内存的话，可直接将-xms和-xmx调成一样即可），GC调优策略有：（1）将新对象预留在新生代，由于full gc的成本远高于minor gc,因此尽可能将对象分配在新生代是明智的做法，实际项目中根据GC日志分析新生代空间大小分配是否合理，适当通过“-xmn”命令调节新生代大小，最大限度降低新对象直接进入老年代的情况。（2）大对象进入老年代，虽然大部分情况下，将对象分配在新生代是合理的，但是对于大对象这种做法却值得商榷，大对象如果首次在新生代分配可能会出现空间不足导致很多年龄不够的小对象被分配在老年代，破坏新生代的对象结构，可能会出现频繁的full gc.因此，对于大对象，可以设置直接进入老年代（当然短命的大对象对于垃圾回收来说简直就是噩梦）。-XX：PretenureSizeThreshold 可以设置直接进入老年代的对象大小。（3）合理设置进入老年代对象的年龄，-XX：MaxTenuringThreshold设置对象进入老年代的年龄大小，减少老年代的内存占用，降低full gc发生的概率。（4）设置稳定的堆大小，堆大小设置有两个参数：-xms：初始化堆大小， -xmx：最大堆大小。（5）如果满足下面的指标，则一般不需要进行GC优化：minor gc执行时间不到50ms, minor gc执行不频繁，约10秒一次， full gc执行时间不到1秒，full gc执行频率不算频繁，不低于10分钟。