|  |
| --- |
| 技术问题(至少50个) |
| **问题1：对Spring事务的理解**  回答：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会失败，以后操作就会回滚到操作前状态，或者是上一个节点.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。  **问题2：Spring概述**  回答：Spring是完全面向接口的设计,降低程序耦合性,主要是事务控制并创建bean实例对象.  在SSM整合时,充当黏合剂的作用(把它当做一个大管家，来帮我们管理对象，控制事务等等).Spring框架有两个核心的特性:  1.IOC(Inversion of Control)控制反转/依赖注入,又称DI(Dependency Injection)(依赖注入)  2.AOP 面向切面编程是对OOP(面向对象编程)的延续和补充,我们在项目中主要使用AOP应用于日志记录,性能统计,安全控制,事务处理等方面.  **问题3: IOC & DI**  回答：IOC不是一种技术，只是一种思想，一个重要的面向对象编程的法则，它能指导我们如何设计出松耦合、更优良的程序。传统应用程序都是由我们在类内部主动创建依赖对象，从而导致类与类之间高耦合，难于测试；有了IOC容器后，把创建和查找依赖对象的控制权交给了容器，由容器进行注入组合对象，所以对象与对象之间是 松散耦合，这样也方便测试，利于功能复用，更重要的是使得程序的整个体系结构变得非常灵活。  DI—Dependency Injection，即“依赖注入”：组件之间依赖关系由容器在运行期决定，形象的说，即由容器动态的将某个依赖关系注入到组件之中。依赖注入的目的并非为软件系统带来更多功能，而是为了提升组件重用的频率，并为系统搭建一个灵活、可扩展的平台。通过依赖注入机制，我们只需要通过简单的配置，而无需任何代码就可指定目标需要的资源，完成自身的业务逻辑，而不需要关心具体的资源来自何处，由谁实现。  IOC和DI由什么关系呢？其实它们是同一个概念的不同角度描述，“依赖注入”明确描述了“被注入对象依赖IOC容器配置依赖对象”。  **问题4:** **AOP的理解**  回答：面向切面的编程AOP，是一种编程技术，允许程序模块化横向切割关注点，或横切典型的责任划分，如日志和事务管理。  AOP的核心是切面，它将多个类的通用行为封装成可重用的模块，该模块含有一组API提供横切功能。  在SpringAOP中，切面通过带有@Aspect注解的类实现。 关注点是应用中的一个模块的行为，一个关注点可能会被定义成一个我们想实现的一个功能。 横切关注点一个关注点，此关注点是整个应用都会使用的功能，并影响整个应用，比如日志，安全和数据传输，几乎应用的每个模块都需要的功能。因此这些都属于横切关注点。 连接点代表一个应用程序的某个位置，在这个位置我们可以插入一个AOP切面，它实际上是个应用程序执行Spring AOP的位置。 切点是一个或一组连接点，通知将在这些位置执行。可以通过表达式或匹配的方式指明切入点。 引入运行我们在已存在的类中添加新的方法和属性。  **问题5:** **Spring的优点**  回答：降低了组件之间的耦合性 ，实现了软件各层之间的解耦  可以使用容易提供的众多服务，如事务管理，消息服务等  容器提供单例模式支持  容器提供了AOP技术，利用它很容易实现如权限拦截，运行期监控等功能  容器提供了众多的辅助类，能加快应用的开发  Spring对于主流的应用框架提供了集成支持，如hibernate，JPA，Struts等  Spring属于低侵入式设计，代码的污染极低  独立于各种应用服务器  Spring的DI机制降低了业务对象替换的复杂性  Spring的高度开放性，并不强制应用完全依赖于Spring，开发者可以自由选择Spring的部分或全部.  **问题6:** **请解释Spring Bean的生命周期**  回答：在一个bean实例被初始化时，需要执行一系列的初始化操作以达到可用的状态。同样的，当一个bean不在被调用时需要进行相关的析构操作，并从bean容器中移除。Spring bean factory 负责管理在spring容器中被创建的bean的生命周期。  Bean的生命周期由两组回调（callback）方法组成。  初始化之后调用的回调方法。  销毁之前调用的回调方法。  **问题7:** **Spring框架的优点**  回答：  •轻量：Spring是轻量级的，基本的版本大小为2MB。  •控制反转：Spring通过控制反转实现了松散耦合，对象们给出它们的依赖，而不是创建或查找依赖的对象们。  •面向切面的编程AOP:Spring支持面向切面的编程，并且把应用业务逻辑和系统服务分开。  •容器：Spring包含并管理应用中对象的生命周期和配置  •MVC框架： Spring-MVC。  •事务管理：Spring提供一个持续的事务管理接口，可以扩展到上至本地事务下至全局事务JTA。  •异常处理：Spring提供方便的API把具体技术相关的异常。  **问题8: Spring中自动装配的方式有哪些？**  回答:  - no：不进行自动装配，手动设置Bean的依赖关系.  - byName：根据Bean的名字进行自动装配。  - byType：根据Bean的类型进行自动装配。  - constructor：类似于byType，不过是应用于构造器的参数，如果正好有一个Bean与构造器的参数类型相同则可以自动装配，否则会导致错误。  - autodetect：如果有默认的构造器，则通过constructor的方式进行自动装配，否则使用byType的方式进行自动装配。  **问题9: Spring框架中的单例Beans是线程安全的么**  回答: Spring框架并没有对单例bean进行任何多线程的封装处理。关于单例bean的线程安全和并发问题需要开发者自行去搞定。但实际上，大部分的Spring bean并没有可变的状态(比如Serview类和DAO类)，所以在某种程度上说Spring的单例bean是线程安全的。如果你的bean有多种状态的话（比如ViewModel对象），就需要自行保证线程安全。最浅显的解决办法就是将多态bean的作用域由"singleton"变更为"prototype".  **问题10:** **@Value**  回答: @Value是一个新的装配注解，可以让我们使用注解装配String类型的值和基本类型的值，如int, boolean。我们可以通过@Value直接标注某个属性，方法或者方法参数，并传入一个String类型的表达式来装配属性.  **问题11:Eureka**  回答: Eureka是Netflix开发的服务发现框架，主要用于定位运行在AWS(亚马逊公司旗下云计算服务平台)域中的中间层服务，以达到负载均衡和中间层服务故障转移的目的。  Eureka包含两个组件：Eureka Server和Eureka Client。  Eureka Server提供服务注册服务，各个节点启动后，会在Eureka Server中进行注册，这样EurekaServer中的服务注册表中将会存储所有可用服务节点的信息，服务节点的信息可以在界面中直观的看到。  Eureka Client是一个java客户端，用于简化与Eureka Server的交互，客户端同时也就是一个内置的、使用轮询(round-robin)负载算法的负载均衡器。  **问题12:** **HashMap源码浅析**  回答: 首先，HashMap 是 Map 的一个实现类，它代表的是一种键值对的数据存储形式。Key 不允许重复出现，Value 随意。jdk 8 之前，其内部是由数组+链表来实现的，而 jdk 8 对于链表长度超过 8 的链表将转储为红黑树。  什么时候扩容：当向容器添加元素的时候，会判断当前容器的元素个数，如果大于等于阈值---即当前数组的长度乘以加载因子的值的时候，就要自动扩容啦。  扩容(resize)就是重新计算容量，向HashMap对象里不停的添加元素，而HashMap对象内部的数组无法装载更多的元素时，对象就需要扩大数组的长度，以便能装入更多的元素。当然Java里的数组是无法自动扩容的，方法是使用一个新的数组代替已有的容量小的数组，就像我们用一个小桶装水，如果想装更多的水，就得换大水桶。  数组的长度是2次幂的扩展(指长度扩为原来2倍)，  经过rehash之后，元素的位置要么是在原位置，要么是在原位置再移动2次幂的位置。**问题13:** **hashmap为什么线程不安全**  回答: hashmap的实现没有加上锁，初步判断不安全我们首先要知道线程操作数据的时候一般是从主存拷贝一个变量副本进行操作。  在多线程下，同时put数据时，两个线程如果读到的副本数据都是1，加完数据后变成2，再把2写入主存中，正常情况下应该变成3，但是还是2，这样就是线程不安全的一个体现  同时还有在多线程情况下，不同的线程同时对hashmap进行扩容机制，进行rehash（）会导致hashmap出现链表闭环，一旦进入了闭环get数据，程序就会进入死循环，所以导致HashMap是非线程安全的。  **问题14:** **LinkedList源码浅析**  回答: LinkedList 基于双向链表实现的，  LinkedList中 维护了两个节点 一个first节点，一个last节点  当我们调用LinkedList的add方法时， 会将当前的last节点 指向给新添加的对象，原来的last节点 变为新加节点的上一个节点  当调用remove(元素)方法时，会从 first节点开始匹配是否是要删除的元素，如果不是找下一个，如果是就直接调用unlink方法，将要删除元素的上一个节点和下一个节点的引用改变，完成删除操作。  **问题15:** **Java对象的生命周期**  回答: 在Java中，对象的生命周期包括以下几个阶段： 1. 创建阶段(Created) 2. 应用阶段(In Use) 3. 不可见阶段(Invisible) 4. 不可达阶段(Unreachable) 5. 收集阶段(Collected) 6. 终结阶段(Finalized) 7. 对象空间重分配阶段(De-allocated)  **问题16:** **简单说说Java中的异常处理机制的简单原理和应用**  回答: 异常是指java程序运行时（非编译）所发生的非正常情况或错误，与现实生活中的事件很相似，现 实生活 中的事件可以包含事件发生的时间、地点、人物、情节等信息，可以用一个对象来表示，Java使用面向 对象的 方式来处理异常，它把程序中发生的每个异常也都分别封装到一个对象来表示的，该 对象中包含有异常的信息。 Java对异常进行了分类，不同类型的异常分别用不同的Java类表示，所有异常的根类为 java.Iang.Throwable, Throwable下面又派生了两个子类： Error和Exception，Error表示应用程序本身无法克服和恢复的一种严重问题，程序只有 奔溃了， 例如，说内存溢出和线程死锁等系统问题。 Exc印tion表示程序还能够克服和恢复的问题，其中又分为系统异常和普通异常： 系统异常是软件本身缺陷所导致的问题，也就是软件开发人员考虑不周所导致的问题，软件使 用者无 法克服和恢复这种问题，但在这种问题下还可以让软件系统继续运行或者让软件挂掉，例如，数组脚本越 界 (ArrayIndexOutOfBoundsException)，空指针异常（NullPointerException)、类转换异常 (ClassCastException); 普通异常是运行环境的变化或异常所导致的问题，是用户能够克服的问题，例如，网络断线， 硬盘空 间不够，发生这样的异常后，程序不应该死掉。 java为系统异常和普通异常提供了不同的解决方案，编译器强制普通异常必须try.. catch处理或用th rows 声 明继续抛给上层调用方法处理，所以普通异常也称为checked异常，而系统异常可以处理也可以不处理 ，所以， 编译器不强制用try catch处理或用throws声明，所以系统异常也称为unchecked异常  **问题17:** **JDK各版本特性**  jdk1.5的新特性：  1. 泛型  2 自动装箱/拆箱  3 for-each  4 static import  Math.sqrt();--------------->sqrt();  5 变长参数  int sum(int ...intlist)有任意个参数,把他看作数组  jdk1.6的新特性  1.增强的for循环语句  Integer[] numbers = computeNumbers();  for (int i=0; i < numbers.length ; i++)  sum += numbers[i];  ||  int sum = 0;  for ( int number: computeNumbers() )  sum += number;  2.监视和管理  Java SE 6中对内存泄漏增强了分析以及诊断能力。当遇到java.lang.OutOfMemory异常的时候，可以得到一个完整的堆栈信息，  并且当堆已经满了的时候，会产生一个Log文件来记录这个致命错误。另外，JVM还添加了一个选项，允许你在堆满的时候运行脚本。  3.插入式注解处理  插入式注解处理API(JSR 269)提供一套标准API来处理Annotations  4.安全性  jdk1.7的新特性  1.模块化特性  Java7也是采用了模块的划分方式来提速，一些不是必须的模块并没有下载和安装，当虚拟机需要的时候，再下载相应的模块，  同时对启动速度也有了很大的改善。  2.多语言支持  Java7的虚拟机对多种动态程序语言增加了支持，比如：Rubby、 Python等等。  3.开发者的开发效率得到了改善  switch中可以使用字符串  在多线程并发与控制方面：轻量级的分离与合并框架，一个支持并发访问的HashMap等等。  通过注解增强程序的静态检查。  提供了一些新的API用于文件系统的访问、异步的输入输出操作、Socket通道的配置与绑定、多点数据包的传送等等。  4.执行效率的提高  对对象指针由64位压缩到与32位指针相匹配的技术使得内存和内存带块的消耗得到了很大的降低因而提高了执行效率。  提供了新的垃圾回收机制（G1）来降低垃圾回收的负载和增强垃圾回收的效果。    JDK1.8的新特性  一、接口的默认方法  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，这个特征又叫做扩展方法。  二、Lambda 表达式  在Java 8 中你就没必要使用这种传统的匿名对象的方式了，Java 8提供了更简洁的语法，lambda表达式：  Collections.sort(names, (String a, String b) -> {  return b.compareTo(a);  });  **问题18:** **HashMap 与 HashTable 的区别**  回答: HashMap是Hashtable的轻量级实现（非线程安全的实现），他们都完成了 Map接口，主要区别在于:  HashMap允许空（nuU)键值（key),由于非线程安全，在只有\_个线程访问的情况下，效率要高 于Hashtable。  HashMap允许将null作为一个entry的key或者value，而Hashtable不允许。HashMap把 Hashtable 的 contains 方法去掉了，改成 containsvalue 和 containsKey。因为 contains 方法 容易让人引起误解。  Hashtable 继承自 Dictionary 类，而 HashMap 是 Java1.2 引进的 Map interface 的一个实现。 最大的不同是，Hashtable的方法是Synchronize的，而HashMap不是，在多个线程访问Hashtable 时，不 需要自己为它的方法实现同步，而HashMap就必须为之提供外同步。  Hashtable和HashMap采用的hash/rehash算法都大概一样，所以性能不会有很大的差异。  **问题19:接口和抽象类有什么区别**  回答:抽象类（abstract class）：  1：abstract 关键字修饰，并且没有方法体  2：抽象类不能直接创建实例  3：抽象类只能被继承，一个具体类继承一个抽象类，必须实现所有抽象方法  接口（interface）：  1：实现接口的一定要实现接口里定义的所有方法  2：接口可以实现多重继承  区别：  1：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  2：一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  3：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  4：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  **问题20:** **什么是java序列化，如何实现java序列化**  回答: 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（将对象转换成二进制）。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。序列化是为了解决在对对象流进行读写操作时所引发的问题 序列化：把Java对象转换为字节序列的过程。 反序列化：把字节序列恢复为Java对象的过程。 实现： 将需要被序列化的类实现Serializable接口 注意： 被关键字static、transient修饰的变量不能被序列化。在被序列化后，transient修饰的变量会被设为初始值。如int型的是0、对象型的是null.  **问题21:** **线程和进程有什么区别**  回答: 线程是进程的子集，一个进程可以有很多线程,每条线程并行执行不同的任务。不同的进程使用不同的内存空间，而所有的线程共享一片相同的内存空间。每个线程都拥有单独的栈内存用来存储本地数据。  **问题22:线程的生命周期(几种状态)**  回答: 线程的状态以及状态之间的相互转换：  1、新建状态(New)：新创建了一个线程对象。  2、就绪状态(Runnable)：线程对象创建后，其他线程调用了该对象的start()方法。该状态的线程位于可运行线程池中，变得可运行，等待获取CPU的使用权。  3、运行状态(Running)：就绪状态的线程获取了CPU，执行程序代码。  4、阻塞状态(Blocked)：阻塞状态是线程因为某种原因放弃CPU使用权，暂时停止运行。直到线程进入就绪状态，才有机会转到运行状态。  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。等待被销毁。  **问题23:** **阻塞的情况**  回答:  (一)、等待阻塞：运行的线程执行wait()方法，JVM会把该线程放入等待池中。  (二)、同步阻塞：运行的线程在获取对象的同步锁时，若该同步锁被别的线程占用，则JVM会把该线程放入锁池中。  (三)、其他阻塞：运行的线程执行sleep()或join()方法，或者发出了I/O请求时，JVM会把该线程置为阻塞状态。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。  **问题24:** **线程池概述**  回答:java线程池的工作原理和数据库连接池的差不多，因为每次重新创建线程 都是很耗资源的操作，所以我们可以建立一个线程池，这样当需要用到线程 进行某些操作时，就可以直接去线程池里面找到空闲的线程，这样就可以直接 使用，而不用等到用到的时候再去创建，用完之后可以把该线程重新放入线程池 供其他请求使用从而提高应用程序的性能。  **问题25:** **线程池的优点**  回答:  重用线程池中的线程,减少因对象创建,销毁所带来的性能开销;  能有效的控制线程的最大并发数,提高系统资源利用率,同时避免过多的资源竞争,避免堵塞;  能够多线程进行简单的管理,使线程的使用简单、高效  **问题26:** **Java关键字volatile与synchronized作用与区别**  回答: 1,volatile 它所修饰的变量不保留拷贝，直接访问主内存中的。 在Java内存模型中，有main memory，每个线程也有自己的memory (例如寄存器）。为 了性 能，一个线程会在自己的memory中保持要访问的变量的副本。这样就会出现同一个变量在某个瞬间， 在一个线 程的memory中的值可能与另外一个线程memory中的值，或者main memory中的值不一致的情况。一 个变量 声明为volatile，就意味着这个变量是随时会被其他线程修改的，因此不能将它cache在线程memory 中。  2,synchronized 当它用来修饰一个方法或者一个代码块的时候，能够保证在同一时刻最多只有一个线程执行该段代 码。 -、当两个并发线程访问同一个对象object中的这个synchronized( this)同步代码块时 ，一个 时间内只能有一个线程得到执行。另一个线程必须等待当前线程执行完这个代码块以后才能执行该代码块 二、 然而，当一个线程访问object的一个synchronized(this)同步代码块时，另一个线 程仍然 可以访问该object中的非synchronized (this)同步代码块。  3,尤其关键的是，当一个线程访问object的一个synchronized(this)同步代码块时， 其他线 程对object中所有其它synchronized (this)同步代码块的访问将被阻塞。  4、当一个线程访问object的一个synchronized (this)同步代码块时，它就获得了这个 object的对象锁。结果，其它线程对该object对象所有同步代码部分的访问都被暂时阻塞。  **问题27:** **Lock和synchronized**  回答:  1）Lock是一个接口，而synchronized是Java中的关键字，synchronized是内置的语言实现；  2）synchronized在发生异常时，会自动释放线程占有的锁，因此不会导致死锁现象发生；而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时需要在finally块中释放锁；  3）Lock可以让等待锁的线程响应中断，而synchronized却不行，使用synchronized时，等待的线程会一直等待下去，不能够响应中断；  4）通过Lock可以知道有没有成功获取锁，而synchronized却无法办到。  5）Lock可以提高多个线程进行读操作的效率。  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。  **问题28:** **多线程高并发业务问题**  回答: 处理高并发一般有分流和队列两种分式。  分流可以用集群实现。  队列可以用activeMQ消息队列或者线程锁。  消息队列就是把所有的线程消息集中管理,排成对一个一个的进行。  线程锁的方式在生成订单编号时用到过，就是在生成编号的方法上加上syn开头的同步锁。对了我们做项目中有个同事提到了用redis可以实现，说redis是单线程的。如果把库存放到redis里边，就可以防止负库存的实现。还有就是我们项目中有时候会用到线程池的技术，比如做数据的批量导入时，如果使用单线程方式，速度会很慢，如果引入线程池，同时开启n个线程，同时进行就可以增加很大的效率。 问题：java开启线程的几种方式：我了解的有继承Thread类，实现runable接口。但是我们现在基本上这样用的比较少，大部分时间都是开启一个spring的线程池ThreadPoolTaskExecutor。  **问题29:** **用户线程和守护线程有什么区别**  回答: 当我们在Java程序中创建一个线程，它就被称为用户线程。  一个守护线程是在后台执行并且不会阻止JVM终止的线程。  当没有用户线程在运行的时候，JVM关闭程序并且退出。  一个守护线程创建的子线程依然是守护线程,GC回收 就是一个守护线程  **问题30:** **JVM的内存区域**  回答: JVM内存主要由 方法区、虚拟机栈、本地方法栈、堆 和 程序计数器组成，  其中方法区和堆是所有线程共享的区域，  **方法区**呢主要存放的是类的信息，常量，静态变量等信息，关于常量JDK1.7以后应该是放到堆中了，  **虚拟机栈**呢主要是存储的是栈帧 栈帧说白了就是描述我们所要调用方法的上下文 比如传递的参数啊，局部变量啊，或者是调用对象的引用等等，我们每调用一个方法就会创建一个栈帧加入到虚拟机栈中，方法执行完会出栈。  这里面有可能抛两种异常 一个栈溢出异常 一个内存溢出异常。  **本地方法栈**和虚拟机栈很像不同的是虚拟机栈是执行java方法 而本地方法栈是为执行native方法而服务。（native方法就是一些其它语言的方法在java中使用native做标识）,  **堆**呢  则是内存当中最大的一块区域了，我们创建的对象基本上都在这里存放，这里也是GC垃圾回收的主要区域，因为现在的很多收集器都是采用分代算法，所以堆又分为 新生代 和 老年代 如果在细致点分 新生代又可以分为Eden （伊甸园）空间、From Survivor （幸存者）空间、To Survivor 空间，如果内存不够也会报内存溢出异常。  **程序计数器** 主要是记录当前线程执行到哪了，如果执行的是native方法 那计数器的值就为undefined  **问题31:** **java的垃圾回收机制**  回答: 这个应该算我们JAVA语言当中很重要的一个特性了。  我们在开发中内存区域的资源很宝贵，有很多对象使用过后是需要清理的，java引用了GC垃圾收集器，  当Java虚拟机发觉内存资源紧张的时候，就会自动地去清理无用对象所占用的内存空间。如果需要，可以在程序中显式地使用System.gc()来强制进行一次立即的内存清理。  Java提供的GC功能可以自动监测对象是否超过了作用域，从而达到自动回收内存的目的，Java的GC会自动进行管理，调用方法：System.gc() 或者Runtime.getRuntime().gc();  **问题32:** **java垃圾回收算法**  回答: ***1.引用计数法 （早期算法）***  引用计数是垃圾收集器中的早期策略。在这种方法中，堆中每个对象实例都有一个引用计数。当一个对象被创建时，且将该对象实例分配给一个变量，该变量计数设置为1。当任何其它变量被赋值为这个对象的引用时，计数加1（a = b,则b引用的对象实例的计数器+1），但当一个对象实例的某个引用超过了生命周期或者被设置为一个新值时，对象实例的引用计数器减1。任何引用计数器为0的对象实例可以被当作垃圾收集。当一个对象实例被垃圾收集时，它引用的任何对象实例的引用计数器减1。  ***2.标记-清除算法分析***  标记-清除算法采用从根集合进行扫描，对存活的对象对象标记，标记完毕后，再扫描整个空间中未被标记的对象，进行回收，如上图所示。标记-清除算法不需要进行对象的移动，并且仅对不存活的对象进行处理，在存活对象比较多的情况下极为高效，但由于标记-清除算法直接回收不存活的对象，因此会造成内存碎片。  ***3. 标记-整理算法***  标记-整理算法采用标记-清除算法一样的方式进行对象的标记，但在清除时不同，在回收不存活的对象占用的空间后，会将所有的存活对象往左端空闲空间移动，并更新对应的指针。标记-整理算法是在标记-清除算法的基础上，又进行了对象的移动，因此成本更高，但是却解决了内存碎片的问题。在基于Compacting算法的收集器的实现中，一般增加句柄和句柄表。  ***4.copying算法(Compacting Collector) 复制算法***  该算法的提出是为了克服句柄的开销和解决堆碎片的垃圾回收。它开始时把堆分成 一个对象 面和多个空闲面， 程序从对象面为对象分配空间，当对象满了，基于copying算法的垃圾 收集就从根集中扫描活动对象，并将每个 活动对象复制到空闲面(使得活动对象所占的内存之间没有空闲洞)，这样空闲面变成了对象面，原来的对象面变成了空闲面，程序会在新的对象面中分配内存。一种典型的基于coping算法的垃圾回收是stop-and-copy算法，它将堆分成对象面和空闲区域面，在对象面与空闲区域面的切换过程中，程序暂停执行。  ***5.generation算法(Generational Collector) 分代算法***  分代的垃圾回收策略，是基于这样一个事实：不同的对象的生命周期是不一样的。因此，不同生命周期的对象可以采取不同的回收算法，以便提高回收效率。  **年轻代（Young Generation）**  1.所有新生成的对象首先都是放在年轻代的。年轻代的目标就是尽可能快速的收集掉那些生命周期短的对象。  2.新生代内存按照8:1:1的比例分为一个eden区和两个survivor(survivor0,survivor1)区。一个Eden区，两个 Survivor区(一般而言)。大部分对象在Eden区中生成。回收时先将eden区存活对象复制到一个survivor0区，然后清空eden区，当这个survivor0区也存放满了时，则将eden区和survivor0区存活对象复制到另一个survivor1区，然后清空eden和这个survivor0区，此时survivor0区是空的，然后将survivor0区和survivor1区交换，即保持survivor1区为空， 如此往复。  3.当survivor1区不足以存放 eden和survivor0的存活对象时，就将存活对象直接存放到老年代。若是老年代也满了就会触发一次Full GC，也就是新生代、老年代都进行回收  4.新生代发生的GC也叫做Minor GC，MinorGC发生频率比较高(不一定等Eden区满了才触发)  **年老代（Old Generation）**  1.在年轻代中经历了N次垃圾回收后仍然存活的对象，就会被放到年老代中。因此，可以认为年老代中存放的都是一些生命周期较长的对象。  2.内存比新生代也大很多(大概比例是1:2)，当老年代内存满时触发Major GC即Full GC，Full GC发生频率比较低，老年代对象存活时间比较长，存活率标记高。  **持久代（Permanent Generation）**  用于存放静态文件，如Java类、方法等。持久代对垃圾回收没有显著影响，但是有些应用可能动态生成或者调用一些class，例如Hibernate 等，在这种时候需要设置一个比较大的持久代空间来存放这些运行过程中新增的类。  **问题33:** **classLoader类加载机制**  回答:  Class文件由类装载器装载后，在JVM中将形成一份描述Class结构的元信息对象，通过该元信息对象可以获知Class的结构信息：如构造函数，属性和方法等，Java允许用户借由这个Class相关的元信息对象间接调用Class对象的功能。  虚拟机把描述类的数据从class文件加载到内存，并对数据进行校验，转换解析和初始化，最终形成可以被虚拟机直接使用的Java类型，这就是虚拟机的类加载机制  **问题34:类加载的生命周期**  回答: 加载、验证、准备、解析、初始化、使用和卸载七个阶段。  **问题35:** **强引用，软引用和弱引用的区别**  回答:  **强引用：**  只有这个引用被释放之后，对象才会被释放掉，只要引用存在，垃圾回收器永远不会回收，这是最常见的New出来的对象。  **软引用：**  内存溢出之前通过代码回收的引用。软引用主要用户实现类似缓存的功能，在内存足够的情况下直接通过软引用取值，无需从繁忙的真实来源查询数据，提升速度；当内存不足时，自动删除这部分缓存数据，从真正的来源查询这些数据。  **弱引用：**  第二次垃圾回收时回收的引用，短时间内通过弱引用取对应的数据，可以取到，当执行过第二次垃圾回收时，将返回null。弱引用主要用于监控对象是否已经被垃圾回收器标记为即将回收的垃圾，可以通过弱引用的isEnQueued方法返回对象是否被垃圾回收器标记  **问题36:** **简述JAVA中I/O和NIO的区别？**  回答：  NIO:是一种new IO，其目的是为了实现高速IO的代码，将IO操作并转换为操作系统，属于非阻塞型，java.nio.\*，是以块（缓冲区）的形式就行数据的传输  IO：是以流的方式就行数据的传输，属于阻塞型，影响程序的性能 传统阻塞IO，如果你要read/write( byte[10M])一个10M的文件，一旦调用了read/write( byte[10M])这个方法，就得等10M全部read/write，方法底层才会返回。 非阻塞线程，调用read/write( byte[10M])方法立即返回，当然这并不能代表10M已经read/write完成，你需要检测这个byte[10M]的缓冲区  java BIO：同步并阻塞，服务器实现模式为一个连接一个线程，即客户端有连接请求时服务器端就需要启动一个线程进行处理，如果这个连接不做任何事情会造成不必要的线程开销，当然可以通过线程池机制改善；  java NIO：同步非阻塞，服务器实现模式为一个请求一个线程，NIO多线程对某资源进行IO操作时会先把资源先操作至内存缓冲区。然后询问是否IO操作就绪，是则进行IO操作，否则进行下一步操作，然后不断的轮询是否IO操作就绪，直到iIO操作就绪后进行相关操作  java AIO：异步非阻塞，异步非阻塞，服务器实现模式为一个有效请求一个线程，客户端的I/O请求都是由OS先完成了再通知服务器应用去启动线程进行处理。  BIO和NIO的异同： 共同点：两者都是同步操作。即必须先进行IO操作后才能进行下一步操作。 不同点：BIO多线程对某资源进行IO操作时会出现阻塞，即一个线程进行IO操作完才会通知另外的IO操作线程，必须等待。  **问题37:** **内存泄露和内存溢出**  回答：  内存泄露 (memory leak)，是指应用程序在申请内存后， 无法释放已经申请的内存空间.一次内存泄露危害可以忽略， 但如果任其发展最终会导致内存溢出(out of memory). 如读取文件后流要进行及时的关闭以及对数据库连接的释放。  内存溢出（out of memory），是指应用程序在申请内存时， 没有足够的内存空间供其使用。 如我们在项目中对于大批量数据的导入，采用分段批量提交的方式。 jvm内存参数是需要根据服务器的指标进行配置的。  **问题38:** **什么是ElasticSearch？**  回答：Elasticsearch是一个基于Lucene的搜索引擎。它提供了具有HTTP Web界面和无架构JSON文档的分布式，多租户能力的全文搜索引擎。Elasticsearch是用Java开发的，根据Apache许可条款作为开源发布。  **问题39:** **ElasticSearch中的集群、节点、索引、文档、类型是什么？**  回答：群集是一个或多个节点（服务器）的集合，它们共同保存您的整个数据，并提供跨所有节点的联合索引和搜索功能。群集由唯一名称标识，默认情况下为“elasticsearch”。此名称很重要，因为如果节点设置为按名称加入群集，则该节点只能是群集的一部分。  节点是属于集群一部分的单个服务器。它存储数据并参与群集索引和搜索功能。  索引就像关系数据库中的“数据库”。它有一个定义多种类型的映射。索引是逻辑名称空间，映射到一个或多个主分片，并且可以有零个或多个副本分片。 MySQL =>数据库 ElasticSearch =>索引  文档类似于关系数据库中的一行。不同之处在于索引中的每个文档可以具有不同的结构（字段），但是对于通用字段应该具有相同的数据类型。 MySQL => Databases => Tables => Columns / Rows ElasticSearch => Indices => Types =>具有属性的文档  类型是索引的逻辑类别/分区，其语义完全取决于用户。  **问题40:** **ElasticSearch是否有架构？**  回答：ElasticSearch可以有一个架构。架构是描述文档类型以及如何处理文档的不同字段的一个或多个字段的描述。Elasticsearch中的架构是一种映射，它描述了JSON文档中的字段及其数据类型，以及它们应该如何在Lucene索引中进行索引。因此，在Elasticsearch术语中，我们通常将此模式称为“映射”。  Elasticsearch具有架构灵活的能力，这意味着可以在不明确提供架构的情况下索引文档。如果未指定映射，则默认情况下，Elasticsearch会在索引期间检测文档中的新字段时动态生成一个映射。  **问题41:Zuul**  回答：**Z**uul 是netflix开源的一个API Gateway 服务器, 本质上是一个web servlet应用。  Zuul提供了服务网关的功能，可以实现负载均衡、反向代理、动态路由、请求转发等功能。Zuul大部分功能都是通过过滤器实现的，Zuul中定义了四种标准的过滤器类型，同时，还支持自定义过滤器。这些过滤器的类型也对应于请求的典型生命周期。  **问题42:** **Elasticsearch中的倒排索引是什么？**  回答：倒排索引是搜索引擎的核心。搜索引擎的主要目标是在查找发生搜索条件的文档时提供快速搜索。倒排索引是一种像数据结构一样的散列图，可将用户从单词导向文档或网页。它是搜索引擎的核心。其主要目标是快速搜索从数百万文件中查找数据。  **问题43:全量索引和增量索引**  回答：全量索引：检索系统再启动时一次性读取当前数据库中（不能直接从数据库中读取）的所有数据，建立索引  增量索引：系统运行过程中，监控数据库变化，即增量，实时加载更新，构建索引；  **问题44:** **什么是Spring Cloud？**  回答：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。  **问题45:** **使用Spring Cloud有什么优势？**  回答：使用Spring Boot开发分布式微服务时，我们面临以下问题  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。  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。  冗余-分布式系统中的冗余问题。  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。  性能-问题 由于各种运营开销导致的性能问题。  部署复杂性-DevOps(Development和Operations的组合词)技能的要求。  **问题46:** **负载平衡的意义什么？**  回答：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。  **问题47:** **什么是Hystrix？**  回答：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。  **问题48:** **容错概念**  回答：在服务发生不可用的时候,给出合理的处理方案:预备方案.  **问题49:** **服务雪崩效应**  回答：当一个服务突然收到高并发请,tomcat服务器如果承受不了的情况下产生服务堆积.可能导致其他服务不可用.  **问题50:** **解决雪崩效应**  回答：利用服务隔离，就是让每个接口互不影响；服务隔离，有两种方式：  **线程池方式：**每个服务都有自己独立的线程池来管理运行自己的接口。  缺点：CPU内存开销非常大。实现完全隔离。能应用高并发解决雪崩效应。  **计数器方式：**使用原子计数器，这对于每个服务设置一个阈值，当服务请求达到这个上线时直接返回一个错误提示（服务降级处理）。  **问题51:** **服务降级**  回答：当前服务不可用，导致客户端进入等待状态，服务降级就是为了不让客户端进行等待，从而返回一个错误提示给客户端。  **作用：**  目的是提高用户体验（示例错误提示：服务器忙请稍后再试），防止服务雪崩效应。  **问题51:** **什么是Netflix Feign？它的优点是什么？**  回答：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  使用功能区进行负载平衡。  获取服务实例，然后获取基本URL。  利用REST模板来使用服务 |
| **人资问题(至少30个)** |
| 问题1：你得离职原因是啥  回答：单位有工作人员是领导的亲戚或朋友,而自己则是应聘进去的,很多好的机会总轮不到自己  回答2：为什么的年龄和身份证不一致呢？  回答：  ........ |