|  |
| --- |
| 技术问题(至少50个) |
| 问题1：对springcloud的理解  回答：springcloud是一系列框架的有序集合，他利用springboot的开发便捷，简化分布式系统的开发，可以说springcloud提供了构建分布式系统所需的全家桶，而且springcloud的组件大多数比较轻量级  问题2：springcloud和dubbo对比  回答： dubbo实现的是服务治理，但是Springcloud实现的是整个微服务架构  问题3：eureka  回答： eureka就是springcloud中提供服注册与发现，他提供了注册中心，服务发现客户端，  还有方便查看所有服务状态的界面，各个服务利用eureka的服务发现功能，将自己  注册到eureka的服务器上  问题4：zuul  回答: 所有的客户端请求通过网关访问后台的服务，网关可以使用一定的路由配置来判断  某一个url由哪一个请求来处理，并从eureka中获取相应的服务来处理请求。  问题5：ribbon  回答：当zuul网关将一个请求发送给某一个服务应用的时候，如果一个服务启动了多个实例，就通过ribbon的负载均衡策略计算出要发送给某一个服务实例  问题5：feign  回答：在ribbon的基础上进行了一次改进，是一个使用起来更加方便的http客户端，feign客户端会默认使用ribbon来实现负载均衡  问题6：Hystrix  回答：在接口上加上Hystrix标签就可以实现对这个接口的监控和断路器功能，能够在一个或者多个依赖出现问题的时候，保证系统依然可用  问题7：config  回答：是一个解决分布式事务的配置管理方案，有Client和server两部分，server提供配置文件，的储存，并且以接口的形式将配置文件的内容提供出去，Client则通过接口获取相应的配置文件并且初始化应用。默认使用git存放配置文件  问题8：Springboot的理解  回答：简化编码，简化配置，简化部署，简化监控，快速搭建项目，Springboot之前的框架，比如ssm都需要引入许多依赖，但是用boot项目只需要一个starter-web依赖，这个依赖包括了许多依赖，比如spring-mvc等。而且他的配置文件也做了优化， 以前的配置文件形式，都是以xml格式配置的，现在则使用一个properties格式或者yml格式即可，ssm框架中，项目启动需要部署tomcat，然后吧项目打成war包，扔到tomcat中，但是在boot项目中，就不需要这么麻烦，因为boot项目直接内嵌了tomcat，所以我们只需要将项目打成jar包，使用命令，java-jar XXX.jar直接一键启动  问题9：hashmap的底层原理  回答：hashmap是一个线程不安全的集合，是以键值对进行存储，采用了数组加链表的结构，  他的原码中，定义了默认容量，为1<<4位，就是16，最大容量1<<30位，加载因子是  0.75f，意思就是，数组的长度达到现在最大长度的0.75倍之后，会触发扩容机制，就  是将数组长度扩大到原数组的二倍，这个扩容实在put的时候触发的，首先这个数组是  Entry类型的数组，每个数据都是一个entry对象，entry对象则有四个属性，为 hash，  Key，value，next，hash就是数组的下标，就是他存在数组的某个地方，key和value就  是键值对，next就是链表的体现，他的每一个next都是一个和他结构相同的entry对象  我们在执行get方法的时候，传过来一个参数就是key，我们会首先判断这个key，是否  为空，如果为空，则直接返回键为null的值，如果不为空，则开始查找，先用他的  Key计算出他的hashcode值，再用他的hashcode值通过hash算法，计算出他的hash  也就是下标，然后根据下标找到数组相应的位置，开始循环遍历链表，查找和自己key  一样的entry对象，得到值后返回，如果没有相应的值则返回null。这就是get的基本流程，还有另一个比较重要的方法就是put，通常在put时候会传来两个参数，就是key和value，他会首先判断这个key是否为null，如果为null则将key为空的键值对存储的  数据返回， 如果不是null，则开始计算hash，通过键计算出他的hash值，找到相应的位置存进去，如果相应的位置已经有值，则存放在这个entry对象的next属性里面，  如果当前位置没有数据，则直接存储进去，存的时候会先判断，当前位置是否已经达到阈值，如果达到，触发扩容函数。这就是hashmap常用的两个方法  问题10：hibernate为什么使用缓存  回答：使用缓存的目的是为了减少数据库的访问次数，提升hibernate的执行效率  问题11：hibernate的一级缓存  回答：一级缓存也叫session缓存，他可以在session的范围内减少数据库的访问次数，只在session的范围内有效，session关闭，一级缓存失效，session缓存是由hibernate提供的用户不能直接操控缓存内容，如果想操控，就通过hibernate提供的evict/clear方法操作，不同的session不会共享数据  问题12：hibernate的二级缓存  回答：sessionfactory级别的缓存，可以跨多个session。即不同的session都可以访问缓存数据，这就叫二级缓存  二级缓存的配置，  cfg.xml中配置属性开启二级缓存  问题13：java内存结构，  回答：我们写完的程序在运行的之前会编译成class文件，然后有我们的运行时数据区分配存储内存，由执行引擎进行执行程序，内存结构主要是在我们的运行时数据区，运行时数据区分为，堆，栈，方法区，本地方发栈，程序计数器，程序执行的时候new出来的对象实例会被存储在堆中，而我们的栈就是存放我们线程运行中的变量以及操作数，  栈中会有栈帧，栈帧就是我们程序中的方法，我们没运行一个方法就会在栈里压入一个栈帧，方法运行完就出栈，栈帧消失，栈帧里有四个结构，就是局部变量表，操作数栈，动态链接，方法出口，局部变量表就是我们方法中的局部变量，操作数栈就是方法中所操作的数，是一个操作数的中转的地方，动态链接不清楚，方法出口就是方法返回地址，  问题14：二叉树  回答：二叉树是链表的加强，在极端情况下，查找的时间复杂度为o(n)  问题15：avl树  回答；也叫平衡二叉树，有自平衡的特性，再插入数据的时候，调整数的结构，每个节点对应的左子树和右子树的高度差不超过1，如果超过就会进行调整，就是左旋和右旋的操作  问题16：红黑树  回答：也有自平衡，但不追求绝对的平衡，允许局部很少的不完全平衡，不会经常调整树的结构，对性能影响不大，规定就是最长路径不能超过最短路径的二倍，超过的话也会进行调整  问题17：红黑树的颜色分配  回答：如果节点是红色，则子节点必须输黑色  根节点是黑色，  空节点也是黑色  一个节点到他的子孙节点路径上的黑节点相同  问题18：linux的常用命令  回答： cd ps mv mkdir scp clear exit chmod vi cat vim rm kill find ln -s slink source ln link source  问题19：说一下常见的锁  回答： 乐观锁，悲观锁，读锁，写锁，无锁，偏向锁，轻量级锁，重量级锁，自旋锁，非自旋锁，分布式锁，区间锁，重入锁，非重入锁，公平所，非公平锁  问题20：介绍一下悲观锁  回答：认为自己在操作数据的时候，一定会有其他线程进入，导致数据的不安全，所以每次操作都要加锁，避免了被其他线程修改数据，悲观锁的主要使用场景是写的操作较多，悲观锁能保证写入的数据的安全  问题21：介绍一下乐观锁  回答：认为自己在操作数据的时候，一定不会有其他线程进入，认为数据一定是安全的，所以，都不加锁，只是加了个cas算法，实现变量的同步，  问题22：那什么是cas算法呢  回答：cas是一个无锁算法，基于硬件原语实现，在不实用锁的情况下，实现多线程之间的变量同步，是一个原子操作  问题23：cas的ABA问题是什么  回答：就是要修改数据的线程没有感知到内存的数据已经经过变化，只是又被改回了原值  问题24：ABA问题怎么解决  回答：ABA问题已经在jdk内部解决，不需要操作，就是加个版本号，每一次修改数据都会版本号+1,让线程感知到数据的变动，进行自旋  问题25：自旋锁是什么  回答：自旋锁就是cas计算中，感知到数据变化后，进行重新从主内存中读取值，再进行比较与交换，知道更新成功  问题26：自旋的时间和次数问题  回答：自旋的时间是由次数决定的，自旋的次数可以自己设置（jdk1.6之前），但是在jdk1.7之后，自旋锁的参数被取消，用户不能配置自旋锁，自旋锁总是会执行，自旋的次数也会由虚拟机自动调整  如果自旋达到一定的次数之后，还没有更新成功，就会返回，或者报错  问题27：线程的上下文切换问题  回答：负责执行某一个线程的时间片结束后，线程的逻辑还没有执行完，就会切换到另一个时间片继续执行，时间片的切换就是线程上下文切换，因为我们的代码不能直接操作底层的硬件，所以我们需要先将切换的请求发送到操作系统，操作系统才能操作底层硬件，  所以比较消耗资源。  另一方面是线程执行过程中，遇到其他线程的抢夺cpu调度权，就需要先将线程中执行到的指令指针保存到内存中，因为我们切换回来之后，要接着执行之前的线程，这种切换也是线程的上下文切换，而且需要将之前线程的中间变量也存到内存中，下次切换回来之后保证之前的变量的值还存在，这种存取是非常消耗资源的，所以说，线程的上下文切换消耗资源。  问题28：synchronized 的理解  回答：是一个悲观锁的体现，是基于 监视器锁实现的（Monitor）  用Monitor做锁的进入和退出操作  Monitor又基于操作系统的互斥锁实现  问题29：分布式事务锁的实现方式  回答：redis database zookeeper  问题30：微服务架构有哪些  回答：之前用过dubbo+zookeeper。现在用比较流行的SpringCloud  ....... |
| **人资问题(至少30个)** |
| 问题1：你得离职原因是啥  回答：  回答2：为什么的年龄和身份证不一致呢？  回答：  ........ |