|  |
| --- |
| 技术问题(至少50个) |
| 事务篇：  问题1：事务的4大传播特性？  回答：原子性，隔离性，一致性，持久性。  问题2：对Spring事务的理解  回答：在我平常的工作过程中对spring的应用还是比较多的，整体感觉也没有什么，主要有两个核心AOP和IOC/DI，分别为面向切面编程思想和控制反转/依赖f注入，工作过程中主要用到IOC注入这块比较多，通过spring的注入能够更加方便维护bean之间的关系，大多数的bean实例都是单例的，很好的节省的对内存的消耗。Spring的注入方式主要用到过构造函数注入，属性注入，注入数据的类型比较多，常用对象注入，也使用到过list集合注入，使用比较灵活。  ioc是控制反转，主要用来维护bean之间的注入关系，使用工厂模式创建bean的实例，然后再根据xml中配置的bean的注入关系为创建好的实例注入bean对象。  Spring的aop面向切面编程是基于代理模式实现的，通过我的工作经验来看，它其实就是把一部分公用的代码段提取到一个实现类中，让程序在运行过程中把代码再拼接成完整代码执行，增强代码的可读性和高复用性，尽可能的避免重复代码的出现，但是并不是所有的业务都适合使用spring的aop功能。  Spring的aop由切点和通知组成，通知分为前置通知、后置通知和环绕通知，我这块主要用过前置和后置通知。环绕通知这块大概了解过，没有深入的理解，主要把公用代码提取到前置通知和后置通知中，在代码运行过程中先运行前置通知方法，再运行切点方法，最后运行后置通知方法。  注释:java反射技术可以通过类的全限定名创建对应对象，创建对象的方法是newInstance创建对应对象。  JAVA反射机制是在运行状态中，对于任意一个类，都能够知道这个类的所有属性和方法；对于任意一个对象，都能够调用它的任意方法和属性；这种动态获取信息以及动态调用对象方法的功能称为java语言的反射机制。  spring原理这块我也有一定的了解，使用spring这么长时间，我认为spring其实就是项目启动时加载spring的监听器，通过监听器和dom4j等解析xml技术读取spring的xml文件，把spring.xml文件中的所有bean标签读取到，拿到标签中的class属性，通过java的反射技术创建bean的实例，把所有创建好的bean实例放入一个beanMap中，beanMap的key为bean标签的ID值，value为java反射技术创建的具体实例对象，后续需要把创建好的bean注入给别的类使用时，其实就是通过bean的id属性从beanMap中获取对应bean实例，调用set方法把获取出来的bean注入到相关使用的实例中。  问题3：Spring通知的五种类型  回答：Before-在方法被调用之前调用通知。  After-在方法被调用之后调用通知，无论方法是否执行成功。  After-returning-在方法成功执行之后调用通知。  After-throwing-在方法抛出异常后调用通知。  Around-方法被调用之前和调用之后各执行一次自定义的行为，通知把被通知方法“包裹”起来了。  问题4：事务常见类？  回答：TransactionDefinition给定的事务规则，定义如隔离级别、传播行为等  PlatformTransactionManager事务管理器  TransactionStatus用于表示一个运行着的事务的状态  问题5：对分布式事务了解吗？  回答：简单的说，就是一次大的操作由不同的小操作组成，这些小的操作分布在不同的服务器上，而且属于不同的应用，分布式事务需要保证这些小的操作要么全部成功，要么全部失败，分布式事务就是为了保证不同的数据库一致性。  问题6：你是如何解决分布式事务的？  回答：  1.两阶段提交方案/XA方案： 举个例子：公司每隔一段时间都会搞一次团建，团建就相当于是一次事件，组织者也就是负责人，第一个阶段：他回询问各自的意见（比如果去哪，都有啥活动），如果都ok就去做这件事，如果其中一个人不ok就取消。第二个阶段：  前提是第二个都ok，大家就可以一起完成了。 这里的负责人就是个事务管理器，主要负责协调多个数据库的事务，保证了数据的一致性，首先去执行各地的本地事务，然后就可以正式提交了整个事务了，如果有一个数据库执行失败了，就需要回滚。可以使用spring+JTA来实现。  缺点：这种方案适用于单体应用，（一个应用连接多个数据源，会跨库操作，严重依赖于数据库操作事务，操作效率低，不适合用于高并发场景，现在都用微服务了，主张的是一个应用连接一个数据库，不允许跨库操作）   1. TCC方案（强一致性）补偿性事务:属于两阶段提交方式的一个变种。T（Try）：对各个服务的资源做确认，及对资源锁定和保留。C（Confirm）：各个服务实际的操作。C（Cancel）：只要有其中一个服务的业务方法出现错误，就需要补偿（即已经执行成功的操作进行回滚）   举个例子：跨行转账：T：先将两个银行账户的资金冻结；C：实际进行转账，A减钱，B加钱；C：如果有任何银行的操作失败了，就需要回滚补偿。比如：A已经扣款成功，B在增加资金的时候失败了，此时就需要把A的钱加回去。  缺点：回滚操作是严重依赖于开发人员的代码进行回滚或者补偿的，代价比较大，所以适应场景是有限的。一般是要求数据一致性要求特别高的时候才用到TCC方案，比如说银行关于钱的一些业务。  3.本地消息表：1）A系统在自己的本地事务操作的同时，会插入一条数据到消息表  2）接着A系统会把找个数据发送到MQ，A系统就是消息的生产者  3）B系统接收到消息（B系统是个消息的消费者），在自己的本地事务中，往自己的本地消息插入一个数据，同时执行自己的业务，如果消息被处理过了，那么这个事务就需要回滚。  4）B系统执行成功后，会更新自己的本地消息表及A系统的本地消息表的状态  5）如果B系统处理失败了，首先不会更新自己的本地消息表，A系统会自己定时扫描自己的消息表，如果发现数据没有改变成成功的状态，会往B继续发送消息。  6）这个方案保证的是消息的最终一致性，即使B失败无数次，A会一直发送消息，直到B成功。A和B并不是严格一致的，而是最终一致的，我们在这里牺牲了一致性，换来了性能的大幅度提升。当然，这种玩法也是有风险的，如果B一直执行不成功，那么一致性会被破坏。  缺点：这个方案严重依赖数据库的消息表来管理事务，不适合于高并发的场景，实际使用少。  4.可靠消息的最终一致性：如果没有本地消息表，直接使用MQ来完成整个事务的操作，RocketMQ  1）A系统先发送一条prepared消息到MQ，如果消息发送失败，就取消操作别执行  2）如果发送成功，MQ会给A发送回调消息，那么A系统就可以处理自己的本地事务，如果失败了，告诉MQ进行回滚。  3）A系统执行成功给MQ发送确认消息，此时B系统就会接收到这个消息然后执行本地事务  注：一般一个项目用到分布式事务也就两三个场景，全用分布式事务的话成本太大，会导致项目特别复杂。  多线程篇：  问题7：进程和线程的区别  回答：进程是正在运行的应用程序，进程是线程的集合。  线程就是一条执行路径，一个独立的执行单元。  多线程就是为了提高程序的效率，多线程之间每个线程互不影响。  问题8：对线程的应用场景：多线程下载，QQ，爬虫，前端开发ajax（异步上传），分布式job（需要同时一个执行多个任务调度）、使用多线程体现程序的效率。  问题9：线程的5种状态：   1. 创建（new Thread()）、 2. 就绪（调用线程start()方法）、 3. 运行（当线程获得CPU事件后，才进入运行状态，真正开始执行run()方法）、 4. 阻塞（由于各种原因进入阻塞状态：1>线程通过调用sleep()方法进入睡眠状态；   2>线程调用一个在I/O上被阻塞的操作，即该操作在输入输出操作完成之前不会返回到它的调用者；3>线程试图得到一个锁，而该锁正被其他线程持有；4>线程在等待某个触发条件。）   1. 死亡（两个原因会导致线程死亡：1）run方法正常退出而自然死亡；2）出现异常而未被捕获，从而终止了run方法而使线程猝死。   为了确定线程在当前是否存活（要么是可运行的，要么是被阻塞了），需要使用isAlive()方法。如果是可运行或被阻塞，这个方法返回true；如果线程仍旧是new状态且不是是可运行的，或者线程死亡了，则返回false；  问题10：线程池  回答：线程池是为突然大量爆发的线程设计的，通过有限的几个固定线程为大量的操作服务，减少了创建和销毁线程所需的时间，从而提高效率。  如果一个线程的时间非常长，就没必要用线程池了(不是不能作长时间操作，而是不宜。)，况且我们还不能控制线程池中线程的开始、挂起、和中止。  问题11：线程池四种创建方式  回答：  newCachedThreadPool创建一个可缓存线程池，如果线程池长度超过处理需要，可灵活回收空闲线程，若无可回收，则新建线程。  newFixedThreadPool 创建一个定长线程池，可控制线程最大并发数，超出的线程会在队列中等待。  newScheduledThreadPool 创建一个定长线程池，支持定时及周期性任务执行。newSingleThreadExecutor 创建一个单线程化的线程池，它只会用唯一的工作线程来执行任务，保证所有任务按照指定顺序(FIFO, LIFO, 优先级)执行。  问题12：线程池的优点  回答：第一：降低资源消耗。通过重复利用已创建的线程降低线程创建和销毁造成的消耗。 第二：提高响应速度。当任务到达时，任务可以不需要等到线程创建就能立即执行。 第三：提高线程的可管理性。线程是稀缺资源，如果无限制地创建，不仅会消耗系统资源， 还会降低系统的稳定性，使用线程池可以进行统一分配、调优和监控。  问题13：线程池原理  回答：1、判断线程池里的核心线程是否都在执行任务，如果不是（核心线程空闲或者还有核心线程没有被创建）则创建一个新的工作线程来执行任务。如果核心线程都在执行任务，则进入下个流程。  2、线程池判断工作队列是否已满，如果工作队列没有满，则将新提交的任务存储在这个工作队列里。如果工作队列满了，则进入下个流程。  3、判断线程池里的线程是否都处于工作状态，如果没有，则创建一个新的工作线程来执行任务。如果已经满了，则交给饱和策略来处理这个任务。  Mybiatis篇：  问题14：什么是Mybatis一级缓存？  回答：是指在内存中开启一块区域，用来保存用户对数据库的操作信息和数据库返回的数据，如果下一次用户再执行相同的请求，那么直接从内存中读取数据而不是从数据库中读取。  问题15：一级缓存中数据生命周期的影响因素（）  回答：1.对sqlsession执行commit操作，也就意味着用户执行了update，delete等相关操作，那么数据库中的数据势必会发生变化，如果用户请求数据仍使用之前的内存中的数据，那么将读到脏数据，所以在执行sqlsession操作后，会清除保存数据的hashMap，用户再发起查询请求时就会重新读取数据并放入一级缓存中了。  2.关闭sqlsession，一般在mybatis集成spring时，会把SqlSessionFactory设置为单例注入到IOC容器中，不把sqlsession也设置为单例的原因是sqlsession是线程不安全的，所有不能为单例。那也就意味着其实是有关闭sqlsession的过程的，其实，对应每一个service中的sqlsession是不同的，这是通过mybatis-spring中的org.mybatis.spring.mapper.MapperScannerConfigurer创建sqlsession自动注入到service中的。  而每个sqlsession单独使用一个缓存空间，不同的sqlsession是不能互相访问数据的，在sqlsession关闭后，其中数据自然被清空。  问题16：为什么要使用二级缓存？  回答：一级缓存中，Sqlsession被关闭后再执行同样的查询操作也会再访问一遍数据库，为了解决这个问题，需要使用二级缓存，一级缓存的作用域仅限于一个sqlsession，但是二级缓存的作用域是一个namespace。如果开启了二级缓存，那么在关闭sqlsession后，会把该sqlsession一级缓存中的数据添加到namespace的二级缓存中。  问题17：如何开启二级缓存？  回答：1.打开总开关，只需要mybatis总配置文件中加入一行设置  <settings>  <!--开启二级缓存-->  <setting name=”cacheEnabled” value=”true”/>  </settings>  2.打开需要使用二级缓存的mapper的开关，在需要开启二级缓存的mapper.xml中加入cache标签<cache/>  3.pojo序列化：让需要使用二级缓存的pojo类实现Serializable接口。通过之前的三步操作就可以使用二级缓存了。注意：只有关闭了sqlsession之后，才会把其中一级缓存数据写入二级缓存。  Redis篇：  问题18：Redis缓存穿透？  回答：指正查询一个数据库一定不存在的数据，首先会从redis缓存中进行查询，如果key不存在或者key已经过期，会进数据库中进行查询，查不到数据则不写入缓存，这将导致这个不存在的数据每次请求都要到数据库去查询，当在大流量流入时，频繁的访问存储层导致数据库直接宕机，造成缓存穿透。  问题19：redis缓存穿透的解决方案？  回答：常见的两种，第一种采用布隆过滤器：提前对所有可能查询的参数以hash形式存储，在控制层先进行校验，不存在则直接被拦截，放弃下面的操作。从而避免了对底层存储系统的查询压力。第二种采用缓存空对象，如果从数据库查询不存在，那么会直接在redis存储一个null或者别的特别的字段，这样下次再访问会从redis中查询，防止穿透的发生。  第三种使用互斥锁排队：就是根据key获取value为空的时候，上锁，从数据库总load数据后再释放锁，若其他线程获取锁失败，则等待一段时间后重试，但是这种方法使得逻辑变得更复杂了，治标不治本。  问题20：什么是redis缓存雪崩？如何解决？  回答：指在某一个时间，缓存集中过期失效，产生的原因之一是在对数据进行缓存的时候是指的过期时间相同导致的，解决的办法1.就是在缓存数据的时候尽可能的分散设置过期时间的范围，热数据设置缓存时间长一点；2.加锁排队；3.建立备份缓存；加机器；  问题21：缓存的击穿：是指一个key非常热，在不停的抗着大并发，当这个key在失效的瞬间，持续的大并发就穿破缓存，直接请求数据库。解决办法：直接将数据生命周期设置成不过期。（*实际项目中除了中真正能碰见如此大并发的项目比较少，如果能碰见如此大的并发，那一定是积累了大量的客户才会这样，比如像淘宝天猫拼多多这种平台，一般的平台没有这么大的并发*）  问题22：redis持久化问题？  回答：redis提供了RDB持久化和AOF持久化，RDB是指在制定的时间间隔内将内存中的数据集快照写入磁盘，这也是默认的持久化方式，这种方式就是将内存中数据以快照的方式写入到二进制文件中，默认文件名称为dump.rdb，我们可以配置在n秒内，如果超过m个key被修改就自动做快照，下面是默认的快照保存配置。  save 900 1 #900秒内如果超过1个key被修改，则发起快照保存  save 300 10 #300秒内容如超过10个key被修改，则发起快照保存  save 60 10000  问题23：Redis的常用数据类型  String：最常见，最简单的key-value类型，二进制安全，可以包含任何数据。比如图片或者序列化对象，一个键最大能存储512MB。  Hash：是一个键值对集合，hash适合用于存储对象，每个hash何以存储40多亿的键值对。  List：简单的字符串列表，按照插入的顺序排序  Set：集合，无序，通过hash表来实现的，不允许重复，如果添加了重复数据，那么会被忽略。  ZSet：和set一样也是String类型元素的集合，且值不允许重复；每个元素都会关联一个double类型的分数，redis正式通过分数为集合中的成员进行从小到大的排序。自动重排。  Redis集群模式：哨兵系统：用于管理多个redis服务器，执行以下三个任务：  监控：哨兵会不断地检查你的Master和Slave是否运行正常。  提醒：当被监控的某个redis出现问题时，哨兵可以通过API向管理员或者其他应用程序发送通知。  自动故障迁移：当一个Master不能正常工作时，哨兵会开始一次自动故障迁移操作，他将会失效Master的其中一个Slave升级为新的master，并让失效Master的其他Slave改为复制新的Master，当客户端试图链接失效的Master时，集群也会向客户端返回新Master的地址，使得集群可以使用Master代替失效Master。  数据库篇：  问题：MyISAM和INNODB的区别是什么？  回答：1.事务安全：（MyISAM不支持事务，INNODB支持事务）  2.外键：MyISAM不支持外键，INNODB支持外键  3.锁机制：MyISAM粒度表锁，INNODB；；粒度行锁  4.查询和添加速度（MyISAM批量插入速度快）  5.全文索引：MyISAM支持全文索引，INNODB不支持全文索引  6.MyISAM内存空间使用率比InnoDB低  问题：默认端口号：  回答：Oracle：1521，redis：3306，SQL Server：1433，MongoDB：27017，Redis：6379  问题：拦截器和过滤器有哪些区别？  回答：  1.拦截器是基于java反射机制的，过滤器是基于函数回调  2.拦截器只对action请求起作用，而过滤器可以对几乎所有请求起作用  3.拦截器不依赖于servlet容器，过滤器依赖于servlet容器  4.在action生命周期中，拦截器可以多次被调用，而过滤器只能在容器初始化的时候被调用一次。  各种锁篇：  问题：乐观锁和悲观锁？  回答：简单的说乐观锁就是指总是假设最好的情况，每次去拿数据的时候都认为别人不会修改，所以不会上锁，但是在更新的时候会判断一下在此期间别人有没有去更新这个数据，可以使用版本号机制和CAS算法实现。  悲观锁：就是每次获取数据的时候都会上锁，这样别人想拿这个数据就会阻塞直到它拿到锁（共享资源每次只给一个线程使用，其他线程阻塞，用完后再把资源转让给其他线程），MySQL里面的表锁，行锁都是操作之前先上锁。Java中synchronized和ReentrantLock等独占锁就是悲观锁思想的实现。  问题：产生死锁的条件？  回答：1.互斥条件：一个资源每次只能被一个进程使用。  2.请求与保持条件：一个进程因请求资源而阻塞时，对已获得的资源保持不放。  3. 不剥夺条件：进程已获得的资源，在未使用完之前，不能强行剥夺。  4.循环等待条件：若干进程之间形成一种头尾相接的循环等待资源关系。    问题：CAS无锁机制  回答：（1）与锁相比，使用比较交换（下文简称CAS）会使程序看起来更加复杂一些。但由于其非阻塞性，它对死锁问题天生免疫，并且，线程间的相互影响也远远比基于锁的方式要小。更为重要的是，使用无锁的方式完全没有锁竞争带来的系统开销，也没有线程间频繁调度带来的开销，因此，它要比基于锁的方式拥有更优越的性能。  （2）无锁的好处：  第一，在高并发的情况下，它比有锁的程序拥有更好的性能；  第二，它天生就是死锁免疫的。  就凭借这两个优势，就值得我们冒险尝试使用无锁的并发。  （3）CAS算法的过程是这样：它包含三个参数CAS(V,E,N): V表示要更新的变量，E表示预期值，N表示新值。仅当V值等于E值时，才会将V的值设为N，如果V值和E值不同，则说明已经有其他线程做了更新，则当前线程什么都不做。最后，CAS返回当前V的真实值。  （4）CAS操作是抱着乐观的态度进行的，它总是认为自己可以成功完成操作。当多个线程同时使用CAS操作一个变量时，只有一个会胜出，并成功更新，其余均会失败。失败的线程不会被挂起，仅是被告知失败，并且允许再次尝试，当然也允许失败的线程放弃操作。基于这样的原理，CAS操作即使没有锁，也可以发现其他线程对当前线程的干扰，并进行恰当的处理。  （5）简单地说，CAS需要你额外给出一个期望值，也就是你认为这个变量现在应该是什么样子的。如果变量不是你想象的那样，那说明它已经被别人修改过了。你就重新读取，再次尝试修改就好了。  （6）在硬件层面，大部分的现代处理器都已经支持原子化的CAS指令。在JDK 5.0以后，虚拟机便可以使用这个指令来实现并发操作和并发数据结构，并且，这种操作在虚拟机中可以说是无处不在。  问题：wait()、notify、notifyAll()方法  回答：wait()、notify()、notifyAll()是三个定义在Object类里的方法，可以用来控制线程的状态。这三个方法最终调用的都是jvm级的native方法。随着jvm运行平台的不同可能有些许差异。如果对象调用了wait方法就会使持有该对象的线程把该对象的控制权交出去，然后处于等待状态。如果对象调用了notify方法就会通知某个正在等待这个对象的控制权的线程可以继续运行。如果对象调用了notifyAll方法就会通知所有等待这个对象控制权的线程继续运行。  问题：wait与sleep区别  回答：对于sleep()方法，我们首先要知道该方法是属于Thread类中的。而wait()方法，则是属于Object类中的。sleep()方法导致了程序暂停执行指定的时间，让出cpu给其他线程，但是他的监控状态依然保持者，当指定的时间到了又会自动恢复运行状态。在调用sleep()方法的过程中，线程不会释放对象锁。而当调用wait()方法的时候，线程会放弃对象锁，进入等待此对象的等待锁定池，只有针对此对象调用notify()方法后本线程才进入对象锁定池准备  获取对象锁进入运行状态  问题：Lock 接口与 synchronized 关键字的区别  回答：Lock 接口可以尝试非阻塞地获取锁 当前线程尝试获取锁。如果这一时刻锁没有被其他线程获取到，则成功获取并持有锁。 Lock 接口能被中断地获取锁 与 synchronized 不同，获取到锁的线程能够响应中断，当获取到的锁的线程被中断时，中断异常将会被抛出，同时锁会被释放。  Lock 接口在指定的截止时间之前获取锁，如果截止时间到了依旧无法获取锁，则返回  集合篇：  问题：Vector与ArrayList区别  回答：.ArrayList是最常用的List实现类，内部是通过数组实现的，它允许对元素进行快速随机访问。数组的缺点是每个元素之间不能有间隔，当数组大小不满足时需要增加存储能力，就要将已经有数组的数据复制到新的存储空间中。当从ArrayList的中间位置插入或者删除元素时，需要对数组进行复制、移动、代价比较高。因此，它适合随机查找和遍历，不适合插入和删除。  2.Vector与ArrayList一样，也是通过数组实现的，不同的是它支持线程的同步，即某一时刻只有一个线程能够写Vector，避免多线程同时写而引起的不一致性，但实现同步需要很高的花费，因此，访问它比访问ArrayList慢  问题24：HasTable与HasMap  回答：1.HashMap不是线程安全的  HastMap是map接口的子接口，是将键映射到值的对象，其中键和值都是对象，并且不能包含重复键，但可以包含重复值。HashMap允许null key和null value，而hashtable不允许。  2.HashTable是线程安全的一个Collection。  3.HashMap是Hashtable的轻量级实现（非线程安全的实现），他们都实现了Map接口，主要区别在于HashMap允许空（null）键值（key）,由于非线程安全，效率上可能高于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。  问题25：ConcurrentHashMap  回答：ConcurrentHashMap内部使用段(Segment)来表示这些不同的部分，每个段其实就是一个小的HashTable,它们有自己的锁。只要多个修改操作发生在不同的段上，它们就可以并 发进行。把一个整体分成了16个段(Segment.也就是最高支持16个线程的并发修改操作。 这也是在多线程场景时减小锁的粒度从而降低锁竞争的一种方案。并且代码中大多共享变 量使用volatile关键字声明，目的是第一时间获取修改的内容，性能非常好。  问题：ArrayList和LinkedList的区别？  回答：前者底层是数组，会在内存中开辟一整快空间，存在下标索引，所以查询快，增删慢，后者是双相链表，在内存中是零散的内存碎片，通过建立指针来指向前后的内存碎片，增删快，查询慢。  Springboot和Spring cloud篇：  问题：zuul 重定向和Cookie传递问题。  回答：通过在zuul网关配置文件中配置add-host-header: true来处理重定向host不正确的问题。  问题：如何防止外网请求跳过网关直接对项目内的接口进行访问：  回答：通过在zuul网关配置文件中配置ignored-patterns:/\*-服务名/\*\*来屏蔽任何以-服务名结尾的服务请求。  问题：作为服务注册中心，Eureka比Zookeeper好在哪里  回答：著名的CAP理论指出，一个分布式系统不可能同时满足C(一致性)、A(可用性)和P(分区容错性)。由于分区容错性在是分布式系统中必须要保证的，因此我们只能在A和C之间进行权衡。在此Zookeeper保证的是CP, 而Eureka则是AP。  当Zookeeper master节点因为网络故障与其他节点失去联系时，剩余节点会重新进行leader选举，问题在于选举leader的时间太长，30-120s，且选举期间整个zk集群都处于不可用状态，这就导致在选举期间 注册服务瘫痪，虽然最终服务能够恢复，但是漫长的选举时间导致的注册长期不可用时不能容忍的；  而eureka首先保证的是可用性，eureka每个节点都是平等的，几个节点down掉不会影响正常节点的工作，剩余的正常节点依旧可以提供服务和注册，但是eureka不能保证服务信息的强一致性，eureka还有一种保护机制，如果在15分钟内85%的节点都没有正常心跳（不可用）  那么Eureka就认为客户端与注册中心之间出现了网络故障，此时会出现以下几种情况： 1、Eureka不再从注册列表中移除因为长时间没收到心跳而应该过期的服务 2、Eureka仍然能够接收新服务的注册和查询请求，但是不会被同步到其他节点上(保证当前节点的可用性) 3、当网络稳定后，当前实例新注册的服务会被同步到其他节点  因此,Eureka能够保证注册中心的高可用性，而不会像zookeeper一样直接集群瘫痪  其他篇：  问题：get和load区别？429  回答：get是立即加载，load是延迟加载。Get查询时会先访问一级缓存，在访问二级缓存，都没有，访问数据库，load查询时会先访问一级缓存，没有则直接创建一个代理对象，需要时代理对象才会去查询二级缓存和数据库，get如果没有查询到会返回null，load没找到会直接抛异常。当我们使用session.load()方法加载对象时，此时并不会发出SQL语句，当前得到的是代理对象，里面只保存了实体对象的id，只有当我们使用这个对象，得到其他属性，这时才会发出sql语句，从数据库查询我们的对象；而当我们session.get()方法来得到一个对象时，不管我们用不用这个对象，此时都会发出sql语句去数据库查询。  问题：session的清理和清空有什么区别？433  回答：session清理缓存是指按照缓存中对象的状态的变化来同步更新数据库，清空是直接关闭session。  问题：ELastic search与Solr的区别？  回答： Solr 利用zookeeper进行分布式管理，而Elasticsearch自身带有分布式协调管理功能。  Solr支持更多格式的数据，而Electricsearch仅支持接送文件格式。  Solr传统的搜索应用中表现好于ElasticSearch，但在处理实时搜索应用时效率明显低于ElasticSearch。  Solr时传统搜索应用的有力解决方案，单ElasticSearch更适用于新兴的实时搜索应用。  问题：==和equals的区别？  回答：最大的区别就是一个是方法，一个是运算符；==：如果比较的对象是基本上是基本数据类型，则比较的是数值是否相等，；如果比较的是引用数据类型，则比较的是对象的地址值是否相等。equals（）：用来比较方法两个对象的内容是否相等，  问题：重载和重写的区别？  回答：重载和重写都是实现多态的方式，区别在于重载是编译时的多态性，重写是运行时的多态性，重载：方法名相同，参数列表（参数类型，参数个数）不同，重写是方法名相同，参数列表相同。  问题：垃圾回收？  回答：标记-清除：准备清楚前先进行标记。  标记-复制：  标记-整理：  分代回收：  问题：如何判断一个对象是否应该被回收？  回答：这就是所谓的对象存活性判断，常用的方法有两种：1.引用计数法；2对象可达性分析，由于引用计数法存在互相引用导致无法进行GC的问题，所以目前JVM虚拟机多使用对象可达性分析。  问题：https有几种请求方式？  回答：常用的几种get，post，put，delete，option，head，trace，connect  问题：常用的SpringCloud注解：  回答：@SpringBootApplication是springboot启动类，包括三个注解，他们的作用分别是：              @Configuration：表示将该类作用springboot配置文件类              @EnableAutoConfiguration：表示程序启动时，自动加载springboot默认的配置              @ComponentScan：表示程序启动是，自动扫描当前包及子包下所有类  @EnableDiscoveryClient和@EnableEurekaClient  @EnableDiscoveryClient基于spring-cloud-commons，@EnableEurekaClient基于spring-cloud-netflix，如果选用的注册中心是eureka，那么就推荐@EnableEurekaClient，如果是其他的注册中心，那么推荐使用@EnableDiscoveryClient。  @ComponentScan     &&     @Configuration    &&    @EnableAutoConfiguration  @ComponentScan 如果不设置basePackage的话 默认会扫描包的所有类，所以最好还是写上basePackage （@componentScan({" ... "}）,减少加载时间。默认扫描\*\*/\*.class路径 比如这个注解在com.wuhulala 下面 ，那么会扫描这个包下的所有类还有子包的所有类,比如com.wuhulala.service包的应用  @Configuration 表示这个类中定义了Bean，会把这个类中bean加载到spring容器中  @EnableAutoConfiguration表示 会在你开启某些功能的时候自动配置，这个注解告诉Spring Boot根据添加的jar依赖猜测你想如何配置Spring。由于spring-boot-starter-web添加了Tomcat和Spring MVC，所以auto-configuration将假定你正在开发一个web应用，并对Spring进行相应地设置。  @Mapper &&  @MapperScan          Mapper类上面添加注解@Mapper，这种方式要求每一个mapper类都需要添加此注解          使用@MapperScan可以指定要扫描的Mapper类的包的路径@MapperScan("com.demo.\*.mapper")||@MapperScan("com.test.\*.mapper","com.demo.\*.mapper")  @EnableTransactionManagement    &&    @Transactional          Spring Boot 使用事务非常简单，首先使用注解 @EnableTransactionManagement （启注解事务管理，等同于xml配置方式的 <tx:annotation-driven />）开启事务支持后，然后在访问数据库的Service方法上添加注解 @Transactional 便可。  @Async && @EnableAsync          @EnableAsync注解的意思是可以异步执行，就是开启多线程的意思。可以标注在方法、类上。          为了让@Async注解能够生效，需要在Spring Boot的主程序中配置@EnableAsync          @Async所修饰的函数不要定义为static类型，这样异步调用不会生效  @Bean && @Configuration          @Bean标注在方法上(返回某个实例的方法)，等价于spring的xml配置文件中的<bean>，作用为：注册bean对象          @Configuration标注在类上，相当于把该类作为spring的xml配置文件中的<beans>，作用为：配置spring容器(应用上下文)  @GetMapping    &&    @PostMapping           Spring4.3中引进了｛@GetMapping、@PostMapping、@PutMapping、@DeleteMapping、@PatchMapping｝，来帮助简化常用的HTTP方法的映射，并更好地表达被注解方法的语义。          @GetMapping是一个组合注解，是@RequestMapping(method = RequestMethod.GET)的缩写。该注解将HTTP Get 映射到 特定的处理方法上。          @GetMapping是一个组合注解，是@RequestMapping(method = RequestMethod.POST)的缩写。该注解将HTTP Post映射到 特定的处理方法上。  @LoadBalanced          Spring Cloud的commons模块提供了一个@LoadBalanced注解，方便我们对RestTemplate添加一个LoadBalancerClient，以实现客户端负载均衡。通过源码可以发现这是一个标记注解,我们可以通过ribbon实现客户端的负载均衡功能。  问题：Cookie 和session 的区别   1. Cookie 数据是存放在浏览器的，session 是存放在服务器 2. Cookie 不安全，可以通过解析存放在本地的cookie 进行欺骗 3. 由于session 是存放在服务器上的,当访问增多的话，会给服务器性能带来压力   问题：懒汉式和饿汉式区别？  回答：私有化构造参数，最大的区别就是懒汉式是延迟加载，是在需要的时候才会创建，而饿汉试在虚拟机启动的时候就会创建，且饿汉试无需关注多线程问题，能用则用，但他是加载类的时候创建的，所有一个工厂模式，缓存了很多实列，就需要考虑效率问题了，因为这个类加载所用，不管有没有用  两者建立单列的对象时间不同，懒汉式是等我们用到的时候才会去创建，饿汉试不管有没有都会去创建  问题：Lunix 常用命令  回答：   1. Kill -9 端口 2. Ps aux|grep tomcat 与grep联用查找某进程 3. Ps -ef|grep java 查找指定进程 4. Ps -ef 显示的是当前所有进程环境变量级进程关系 5. Ps -A 查看所有进程 6. Tar zxvf 解压 7. Find ./ -name ‘\*.log’ 在当前目录查找以.log结尾的文件 8. Mkdir 创建文件 9. Pwd 查看挡圈所在目录的绝对路径 10. Tail -f 查看日志   ....... |
| **人资问题(至少30个)** |

|  |
| --- |
| 您好，我叫刘佳鑫，  辽宁铁岭人，今年28岁，  毕业于沈阳工程学院，毕业后去外地工作，工资不高，投靠亲戚进入it行业。  第一家公司：2015.08---2016.08 重庆新科技科技有限公司  第二家公司（外包）：2016.09---2019.05，北京华路时代信息技术股份有限公司 项目结项了  北京市海淀区上地三街9号B座4层B407室。（做完了结项了）  （大概4年半工作经历）  上一个项目做的是金融，后台通过Spring Could微服务搭建开发，用Spring Could Erueka微服务来搭建分布式集群来降低模块之间的耦合度等效果，为了更加的灵活操作数据、持久化数据采用的MyBatis框架，还有防止后台（前台）对Mysql数据库数据频繁的访问，采用了Redis缓存数据库。数据展示（FreeMarker）静态模板引擎,上述是此平台的大概技术架构，还有网关、负载均衡等技术亮点项目一共分为前台后台两个系统，前台用户系统，后台管理员系统，我参与的项目有，登陆模块，开户模块，充值模块等等。  问题1：你的离职原因是啥  回答：项目结项了，外派外地出差一年。  问题2：之前是做什么的？  回答：  问题3：哪个大学毕业的？  回答：毕业于沈阳工程学院  问题4：学的什么专业？  回答：热能与动力工程  问题5：之前在哪儿工作？  回答：四川宜宾。  问题6：学历学信网可查吗？  回答：可查  问题7：说说你的经历  回答：2014年6月毕业，之后就到重庆工作了，干了三个月，那儿的生活不是我想要的，工资低，危险。亲戚搞it而且已经毕业了  问题8：你觉得自己有哪些缺点？  回答：不会拒绝，直男，觉得谁都是好朋友  问题9：优点  回答：乐观，努力，刻苦  问题10：下班都会做什么？  回答：只要是11点之前回家，锻炼半小时，学习1个小时  问题11：上没上过五险一金？  回答：没上过。  问题12：来北京三年了没上过？  回答：因为当初来北京的时候家里做买卖赔了不少钱，还有外债，我入职之后就跟公司提交的申请，就是能不能直接套现，公司挺好同意了，去年6月份都还清了，看病啥的挺难的，就还是觉得交五险一金比较好。  问题13：为了做好你工作份外之事，你该怎样获得他人的支持和帮助？  回答：首先我们是一个开发团队，既然是团队那就要互相帮助，团队中其他人有困难积极去帮助解决，自己做好了，做到了，自然就会获得他人的认可支持和帮助。  问题14：你和别人发生过争执吗？你是怎样解决的？  回答：要理性的看待问题，找到问题的本质，积极地去解决，争执和吵闹解决不了任何问题。  问题15：如果我录用你，你将怎样开展工作？  回答：首先肯定是要适应熟悉工作环境和公司企业文化，规章制度，多学习尽快理清正在开发的项目的思路，理清业务逻辑，工作之余多跟前辈们学习交流，尽快融入到集体中去。  问题16：为什么选择我们这家公司？  回答：贵公司这次招聘的职位我觉得很适合我，而且之前了解过贵公司对待员工特别好，重视人才，我特别喜欢在这样的公司任职。  问题17：我们公司为啥要录用你呢？  回答：我优点很多啊，能吃苦，积极乐观，热爱工作，人品好，我也很符合贵公司的任职要求  问题18：休息时间和工作时间发生冲突时，怎么办？  回答：肯定不能影响公司工作进度啊，我是个热爱生活的人，同时也是个知道轻重的人，既然选择了贵公司，那就一定会以工作为主，还有就是我单身。  问题19：如果项目经理给你分配的任务过重，你怎么办?  回答：人要有最起码的判断能力，如果任务过重但是按时保质保量的做完，那就努力去做，如果做不完，那要尽快汇报上级，申请加派人手，不能隐瞒不报影响公司正常的工作进度。  问题20：什么会让你有成就感？  回答：能帮助到别人，让我最有成就感，不光是工作中，在生活上能帮助到他人我也会很开心。  问题21：怎样对待自己的失敗？  回答：失败是成功之母这句话不是没有道理的，跌倒了要有勇气再站起来，总结失败的经验，吸取教训，逐渐完善自我，迎娶白富美，走向人生巅峰。  问题22：你通常如何处理別人的批评？  回答：凡事必有因果，面对别人的批评，要理性看待，有则改之无则加勉。  问题23：最能概括你自己的三个词是什么？  回答：乐观，能吃苦，努力。  问题24：你能为我们公司带来什么呢？  回答：首先，立足本职，尽职尽责，把本职工作做好，要维护公司的形象，对公司忠诚，规范自己的言行，绝不损害公司的利益。作为公司的一员，明确[目标责任](https://www.baidu.com/s?wd=%E7%9B%AE%E6%A0%87%E8%B4%A3%E4%BB%BB&tn=SE_PcZhidaonwhc_ngpagmjz&rsv_dl=gh_pc_zhidao" \t "https://zhidao.baidu.com/question/_blank)指标，并努力去完成它。  问题25：在完成某项工作时，你认为领导要求的方式不是最好的，自己还有更好的方法，你应该怎么做？  回答：当领导在会议上说出来 的时候,我会记住,当会议结束,领导也没有什么事要忙,我会进办公室与领导私聊,本着虚心求教的态度，把自己想法说出来,与领导讨论,当然最后的决定权还是在领导手中。  问题26：面对一件事情的对错你怎么看？  回答：小孩才分对错，大人只看利弊，凡事都没有绝对的对与错，要多方面去想问题。  问题27：工作中你难以和同事、上司相处，你该怎么办？  回答：这种事情一般不会发生在我身上的，①我会服从领导的指挥，配合同事的工作。  ②我会从自身找原因，仔细分析是不是自己工作做得不好让领导不满意，同事看不惯。还要看看是不是 为人处世方面做得不好，如果是这样的话我会努力改正。  ③如果我找不到原因，我会找机会跟他们沟通，请他们指出我的不足，有问题就及时改正。  ④作为优秀的员工，应该时刻以大局为重，即使在一段时间内，领导和同事对我不理解，我也会做好本职工作，虚心向他们学习，我相信，他们会看见我在努力，总有一天会对我微笑的。  问题28：你最擅长的技术方向是什么？  回答：我最擅长的是ssm吧,虽然说我上家公司一直用的是spring全家桶，但是只就技术而言还是ssm掌握的扎实一点。spring也是一种发展趋势,当然我还是有好多地方需要学习的.  问题29：你对加班的看法？  回答：作为一个程序员,我觉得加班还是比较正常的,当我们工作没做完,或者,工期比较紧  的时候可以适当加班,这个我还是比较赞同的.  问题30：说说你的职业规划？  回答：在年龄允许的情况下，还是想在技术上走的更远更高一点，两年内成为公司开发骨干  吧，4年内争取走上项目经理的职位。  回答31：你还有什么问题要问吗？  回答：咱们这个分公司多少人，开发用什么框架？（漂亮小姐姐多不多）  ........ |