面试模拟问题

1. **面向对象设计的原则？开闭原则在JDK或你常用的框架中是怎样体现的？你用过哪些设计模式(不要说个单例或代理)，可以详细介绍一下吗？**

   面向对象的设计原则有迪米特法则，里式替换原则，依赖倒置原则，开闭原则，单一职责原则

   开闭原则是对扩展开放对修改关闭。对扩展开放意味着模块的行为是可扩展的，对于修改的封闭则是对模块的行为进行扩展时，不必改动模块的源代码。

   JDK中提供接口或抽象类，我们可以通过一个类实现多个接口来实现类方法的扩展；~~对修改关闭意味着要封装变化，如SpringMVC中，每个模块都有对应的Service接口和其实现类，将相同的变化封装到一个接口或者抽象类中，不同的变化封装到不同的接口或抽象类中~~。（其实想说不同模块通过依赖注入实现service类的调用，这样代码改变的时候无须改变依赖他的类）

   Spring 框架的依赖注入体现了开闭原则。spring 通过依赖注入，将当前类所需要的服务注入到使用他的地方，由Spring容器去负责对象的创建和装配。这样想对某个类进行改变的时候，就不用改变所有依赖的他的类。

   我使用过策略模式和装饰者模式。

   策略模式：策略模式的主要目的是将算法的定义与使用分开，将算法的定义放在专门的策略类中，每一个策略类封装了一种实现算法，由环境类（content）负责决定使用不同的策略。

   在开发过程中Controller层和Service层实现的策略模式。~~使用Controller层来控制不同请求URL使用不同的Service来完成不同的功能，并返回相应的结果。Controller层Service功能是如何实现的，实现了Controller和Model之间的解耦~~。Service类提供了不同的策略，Controller 类根据不同的 url 来调用不同的 Service 完相应的功能。举个例子：如 /item/getList 的 url请求返回商品列表，Controller层会根据这个url 调用 itemService 完成相应的功能。

   装饰者模式：在不改变原有对象的基础之上，将功能附加到对象上。他通过组合被装饰者对象，可以对被装饰者对象的“行为”进行扩展。

   在项目中我主要是使用的是包装器业务异常类来统一管理在项目运行时抛出的异常。业务首先定义一个异常类接口，包括异常的错误码和错误信息等。然后自定义一个业务异常类BusinessException和异常枚举EmBusinessError实现异常类接口。EmBusinessError中定义项目规定好的错误码和描述信息，BusinessException实现对枚举异常类EmBusinessError的包装：可以使用定义好的枚举类当做构造函数的参数来构造自定义业务异常类，也可以自定义错误码来构建业务异常。最终可以在业务代码中抛出使用自定义的业务异常类。

2. **SpringCloud Eureka了解吗？对比一下ZK eureka和 console？用的时候有没有遇到什么坑，怎么解决的？**

   Eureka是SpringCloud的服务中心，实现了微服务部署之后的注册和发现。Eureka专门用于给其他服务注册的称为Eureka Server(服务注册中心)，其余注册到Eureka Server的服务称为Eureka Client。Eureka Client分为服务提供者和服务消费者。Eureka 提供了以下治理机制：对于服务提供者：有服务注册（启动的时候会通发送REST请求的方式将自己注册到 Eureka Server 上）、服务续约（注册完后定期发心跳）、服务下线（关闭时发送服务下线的REST请求给 Eureka Server）对于服务消费者：获取服务（启动服务消费者的时候，发送REST请求给服务注册中心获取服务清单）和服务调用（通过服务名可以获得具体提供服务的实例名和该实例的元数据信息）

   对比ZK： 二者的实现是针对 CAP 原则的实现不同

   C: 数据一致性(consistency)：节点拥有数据的最新版本；A：可用性(availability)：数据具备高可用性；P：分区容错性(partition-tolerance)：容忍网络出现分区，分区之间网络不可达。CAP理论指出，C 和 A 是 无法兼得的

SpringCloud Eureka基于AP，能保证高可用，即使所有机器都挂了，也能拿到本地缓存的数据。他是针对于服务发现场景来设计的，对于服务消费者来说，最重要的是服务能够消费。拿到可能不正确的服务实例信息后尝试消费一下，也好过因为无法获取实例信息而不去消费。对于服务发现而言，可用性比数据一致性更加重要。在遇到问题时牺牲一致性来保证可用性，既返回旧数据，缓存新数据。

Zookeeper则是基于CP来设计的，即任何时刻对Zookeeper的访问请求能得到一致的数据结果，同时系统对网络分割具备容错性，但是它不能保证每次服务请求的可用性。从实际情况来分析，在使用Zookeeper获取服务列表时，如果zookeeper正在选主，或者Zookeeper集群中半数以上机器不可用，那么将无法获得数据。所以说，Zookeeper不能保证服务可用性。

3. **SpringCloud里你还对哪些组件比较熟悉? 看过哪个的源码, 能简单介绍下他的原理吗?**

   客服端负载均衡：Netflix Ribbon；

   断路器：Netflix Hystrix：在高并发的状态下由于某个单个服务的延迟，可能导致所有的请求都处于延迟状态，服务处于负载饱和状态最终导致分布式系统都不可用，发生雪崩。Spring Cloud Hystrix实现了断路器、线程隔离等一系列服务保护功能。

   - Fallback(失败快速返回)：当某个服务单元发生故障，通过断路器的故障监控， 向调用方返回一个错误响应， 而不是长时间的等待。这样就不会使得线程因调用故障服务被长时间占用不释放，避免了故障在分布式系统中的蔓延。
   - 依赖隔离(线程池隔离)：它会为每一个依赖服务创建一个独立的线程池，这样就算某个依赖服务出现延迟过高的情况，也只是对该依赖服务的调用产生影响， 而不会拖慢其他的依赖服务。

   服务网关：Netflix Zuul；

   分布式配置：Spring Cloud Config

   介绍一下Hystrix断路到恢复的过程

   Hystrix提供几个熔断关键参数：滑动窗口大小、 熔断器开关间隔、错误率

   - 如果请求数达到了设置的请求阈值或者请求失败的比例超过了设置的比例，熔断器就会打开，此时再调用此服务，将会直接返回失败，不再调远程服务。
   - 当开关时间间隔过后，下一个请求将被放过，为了验证下一后面的路是否通畅，此时断路器处于半开半闭状态。请求失败则熔断器继续打开，请求成功则熔断器关闭，返回相应结果

   Hystrix/Feign/Ribbon三者是怎么协作的(代码上是怎么设计的)

   代码设计是指源码上是怎么设计的吗?

   流程解释：一个服务调用接口，首先Ribbon会从 Eureka Client里获取到对应的服务注册表，Ribbon 做负载均衡，可以使用 Round Robin 轮询算法，从多台服务器取选择其中的一台机器；Feign 会针对这台服务器，基于动态代理机制，根据注解和选择的机器，拼接请求URL地址，构造发起请求。发起的请求是通过Hystrix的线程池来走的，不同的服务走不同的线程池，实现了不同服务调用的隔离，避免了服务雪崩的问题。

   线程池隔离和信号量隔离有什么区别, 各自的优势和劣势是什么

   使用信号量，创建多少线程实际就会有多少线程执行了，同时执行的线程数量会有限制。使用线程池，创建的线程只是作为任务提交给线程池执行，实际工作的线程由线程池创建和管理。

   线程池隔离是把每个服务单独用一个线程池去隔离，当线程池内最大线程超过指定值时会进行熔断；

   信号量隔离是每次调用线程，通过信号量的计数器进行限制。当信号超过最大请求数时进行熔断；

   线程隔离优势：支持超时和异步，劣势：上下文切换需要较大的资源消耗

   信号量隔离优势：资源消耗小，劣势：不支持超时和异步调用

4. **SpringAOP如何实现的, 动态代理的实现方法, 对比JDK和cglib动态代理**

SpringAOP是通过代理实现的，其中静态代理是在编译阶段就可以生成代理类，而动态代理是在运行时动态在内存中临时生成AOP动态代理。

spring的动态的代理模式有两种：JDK动态代理,基于接口(默认代理模式)，CGLIB动态代理（若要使用需要进行配置）

动态代理：如果要代理的对象实现了某个接口，那么Spring AOP会使用JDK Proxy，去创建代理对象，而对于没有实现接口的对象，使用Cglib 生成一个被代理对象的子类来作为代理

5. **Redis用来做什么了? 有没有遇到缓存和数据库数据不一致的情况,怎么解决? Redis支持事务吗? Redis支持的数据结构知道/用过哪些? Redis为什么快? Redis为什么用跳表 而不用B+树?Redis持久化?**

   Redis主要是缓存功能，在项目中主要用来存储常访问的数据或者经常更新的数据，如常用的字典字段（包括订单状态、商品状态等）。支持事务。读缓存和数据库不一致的情况：一般数据读的时候先读缓存，如果没有数据就去数据库中读取数据，将读取的数据放回缓存；更新的时候先更新数据库，再删除缓存。

   数据结构：string、list、set、hash、zset。其中用string当做key 用hash存value的情况比较多。

   ~~Redis 当做用户和数据库之间的缓存，用户可以直接从内存中读取而不是从磁盘中读数据因此更快；而且修改数据时，避免了数据库并发修改数据加锁带来的性能和时间损耗~~

   Redis为什么快：redis是基于内存的数据库，请求都是基于内存的操作；数据结构简单，没有像B树那么复杂的数据结构；单线程，避免了上下文切换、多线程竞争、不用考虑锁的问题；采用多路I/O 复用技术可以让单个线程高效的处理多个连接请求，尽量减少网络 IO 的时间消耗。

   Redis为何使用跳表?

   B+树叶子节点存储数据，非叶子节点存储索引，查询数据时首先要根据索引查询到叶子节点，再到叶子节点所指向的地址去磁盘读取数据；这涉及到 I /O 操作。

   而Redis是基于内存的不需要I/O操作，并且跳表与B+树相比有很少的内存占用，B+树有2个以上的指针，而跳表的指针数平均为1/(1-p)(p为结点具有的指针概率，如Redis为 1/4)；在删除和修改操作上跳表只需修改相邻结点的指针，而B树要分裂或合并结点来调整树；且跳表具有较简单的实现（基于链表）

   Redis持久化有两种：RDB（设置快照保存某个时间点上的数据副本，使用快照备份) AOF(数据命令改变后，将命令写入磁盘中的AOF文件，服务器重启后加装文件来实现数据的恢复)

   Redis事务是可以一次执行多个命令，本质是一组命令的集合。一个事务中的所有命令都会序列化，按顺序串行化的执行而不会被其他命令插入。他没有隔离级别的概念，队列的命令提交前都不会被执行。不保证原子性，若一个事务里有一条命令执行失败，其他的命令仍然会执行。不支持传统关系数据库中的事务ACID特点

6. **GitFlow了解吗?**

   Git Flow定义了一个项目发布的分支模型，提供了一个清晰的分支规范，如项目中的master分支存放的是正是发布的版本，还会有对应版本的tag、dev分支主要是开发分支，基于master分支创建，feature分支为新功能分支，feature分支都是基于develop创建的，开发一个新功能时会新建feature分支，开发完成后合并到develop分支上，release是基于最新develop创建，作为一个发布新版本的分支，hotfix基于master分支创建主要用于修复线上bug。我在实习过程中，要实现商品的重构商品的的开发工作时，就新建了一个开发分支，当功能完毕后，合并到主开发分支；同时在开发kafka、redis相关功能的同事也有自己的开发分支；在封版的时候，测试的时候会创建一个创建relase分支，所有的bug会再此分支上改，改完再合并到master和dev分支上。

7. **项目中"重构查询方式，将多表连接查询分解 成单表查询，在业务代码中关联，使缓存更高效，提高了数据的可分离性", 能介绍一下吗? 如果不关联, 如何处理Join查询分页问题呢? 为什么不用ES, 直接从ES查? ES用来干什么了, 怎么往ES同步的数据**?

   比如查询订单详情，以前的查询时基于订单表基本表、订单商品表、订单收货地址表三个表连接查询而来，现在把这一个SQL语句改成三个SQL语句，在Service分别调用这三种查询并合成一个结果，这样拆分后的SQL语句还可以被其他的功能使用。拆分的时候没有遇到分页的问题。

这样有助做的分库和分表做，分库分表使用join链接多表会降低性能。分解查询时也可以减少锁的竞争。这样也使得单个的sql更好的进行维护。还有一个出发点就是如过某个表很少改变，那么查询的时候直接走缓存就可以，使缓存的利用率更高。
当时我做的时候没有遇到需要分页的Sql语句，像大的分页问题如商品页搜索和商家订单的搜索还是从ES查的。具体的分页问题如果在业务代码中合并数据不复杂的话在应用层分页，数据量不是很大的话，可以一次返回全部数据在前端分页。（如何分页问题这里也想请教一下学长）

ES主要用到的是商铺的订单和商品查询、用户的商品查询。

ES数据同步：(参考网上答案，在实习的项目中如何进行的ES同步不太了解）（1）同步双写：数据修改到数据库中，同时写到ES中。（可能有双写失败风险，ES和mysql业务逻辑强耦合）（2）异步双写：从数据写到数据库中后，再使用MQ （3）使用多线程异步调用es的保存操作。

8. **SpringBoot的优势是什么, 自动配置和自动依赖是怎样实现的?**

   主要是配置简单，尽可能的提供项目依赖来自动配置Spring框架、内嵌Tomcat框架。

   自动配置：Spring Boot启动的时候会通过@EnableAutoConfiguration注解找到META-INF/spring.factories配置文件中的所有自动配置类，并对其进行加载，而这些自动配置类都是以AutoConfiguration结尾来命名的，它实际上就是一个JavaConfig形式的Spring容器配置类，它能通过以Properties结尾命名的类中取得在全局配置文件中配置的属性如：server.port，而XxxxProperties类是通过@ConfigurationProperties注解与全局配置文件中对应的属性进行绑定的。

   （自动依赖？）起步依赖：本质是一个maven项目对象模型（project object model，pom），定义了对其他库的传递依赖，这些依赖整合在一起可支持某项功能。如果记性WEB项目开发，引入引入spring-boot-starter-web依赖就可以。