面试模拟问题 V2

问题补充：用过哪些设计模式？如何使用的？

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。策略模式将要实现的策略定义成接口，具体的策略类实现策略接口；Content类与具体的策略类组合，来实现根据情况下使用不同策略

我使用基于SpringIOC 实现策略模式：Spring提供了根据类型注入bean的特性：如果依赖注入的是集合类或者Map类型的，容器将自动装配所有与声明值类型匹配的bean，对于Map ， key将解析为相应的bean名称，value为对象实现bean实例。

1. 定义一个商品的活动类接口当做策略接口；定义两个接口继承商品活动接口当做具体的策略（如商品折扣活动，商品抢购活动，商品组团活动等）；这几个具体的策略有各自的实现类 Service。
2. 具体的实现类Service在@Service注解上指定value值 如 `@Service("discountActivityService")`
3. 定义活动类型枚举，封装各种类型的活动，并设置活动key、活动描述、活动value值，即对应的Service实现类的value（@service注解上的指定的value值）
4. 在 Controller 里根据不同的情况调用不同的Service类，也就是不同的策略。此时Controller相当于 Content 类。
5. 在Controller 类里注入 Map，key 代表具体service实现类的value值，value 代表活动接口
6. 这样我们可以根据不同的活动类型，去活动枚举类中得到对应活动的具体 service 的value值
7. 将活动的value值当做map的key，去 map 中根据key 就可以获得具体的活动实例

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。

装饰者模式定义了一个组件（被装饰者）作为接口，装饰者类继承组件接口，具体的装饰者类实现装饰者类的接口；装饰者通过组合组件类，来动态的扩展组件的行为。

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EmBusinessException实现异常类接口。EmBusinessException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EmBusinessException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。

Eureka保护机制

Euraka 的自我保护模式正是一种针对网络异常波动的安全保护措施，使用自我保护模式能使Eureka集群更加的健壮、稳定的运行。

默认情况下，如果Eureka Server在一定时间内（默认90秒）没有接收到某个微服务实例的心跳，Eureka Server将会移除该实例。但是当网络分区故障发生时，微服务与Eureka Server之间无法正常通信，而微服务本身是正常运行的，此时不应该移除这个微服务，所以引入了自我保护机制。

自我保护机制的工作机制是如果在15分钟内超过85%的客户端节点都没有正常的心跳，那么Eureka就认为客户端与注册中心出现了网络故障，Eureka Server自动进入自我保护机制。Eureka Server会将当前的实例注册信息保护起来，同时提示一个警告，一旦进入保护模式，Eureka Server将会尝试保护其服务注册表中的信息，不再删除服务注册表中的数据。也就是不会注销任何微服务。

Eureka Server间如何做数据同步

Eureka 是基于AP的，保证具有高可用性，但他是弱数据一致性的：主要采用异步的方式进行同步，不能保证结点的状态一定是一直的，能保证最终状态是一定的

1. Eureka 是采用 peer to peer 对等复制，副本间不分主从，任何副本都可以接受写操作，每个副本之间互相进行数据更新同步；Eureka Server 是作为其他 Eureka Server 的 Client。
2. Eureka Server 启动后，会通过 Eureka Client 请求其他 Eureka Server 节点中的一个节点，获取注册的服务信息，然后复制到其他 peer 节点。
3. Eureka Server 每当自己的信息变更后（例如 Client 向自己发起注册、续约、注销请求），就会把自己的最新信息通知给其他 Eureka Server，保持数据同步。
4. Eureka Server在执行同步操作的时候，使用HEADER_REPLICATION的http header来将这个复制请求操作与普通应用实例的正常请求操作区分来。通过HEADER_REPLICATION来标志复制请求。这样peer节点收到请求的时候，就不会对其进行复制操作，避免同步死循环（一个自己的信息变更是由另外一个Euraka Server 同步过来的，再同步回去会出现数据同步死循环）
5. 通过版本号来机制来判断请求复制的数据的版本号和本地数据的版本号的高低。lastDirtyTimestamp是注册中心里面服务实例的一个属性，表示此服务实例最近一次的变更时间。假设 ureka Server A 向 Eureka Server B 请求复制数据：
    1. A 的数据比B的数据新，B返回 404 说明自己没有这个数据，则A重新把这个应用实例注册到B，注册完成后完成数据同步
    2. A 的数据比B的数据旧，B返回 409 说明自己的数据新，存在数据冲突，要求A同步B的数据
6. 使用心跳机制来实现数据修复

ZK选举过程/zab协议

1. 每个Server启动后，发出一个投票投给自己。例如初始情况，Server1和Server2都会将自己作为Leader服务器来进行投票，每次投票会包含所推举的服务器的 Servierid 和 ZXID（服务器中存放的最大数据ID），使用(Servierid, ZXID)来表示，此时Server1的投票为(1, 0)，Server2的投票为(2, 0)，然后各自将这个投票发给集群中其他机器。
2. 接受来自各个服务器的投票。集群的每个服务器收到投票后，首先判断该投票的有效性，如检查是否是本轮投票、是否来自LOOKING状态的服务器。
3. 处理投票。针对每一个投票，服务器都需要将别人的投票和自己的投票进行PK，PK规则如下

    1、优先检查ZXID。ZXID比较大的服务器优先作为Leader。

    2、如果ZXID相同，那么就比较myid。myid较大的服务器作为Leader服务器。

    对于Server1而言，它的投票是(1, 0)，接收Server2的投票为(2, 0)，首先会比较两者的ZXID，均为0，再比较Servierid ，此时Server2的Servierid 最大，于是更新自己的投票为(2, 0)，然后重新投票，对于Server2而言，其无须更新自己的投票，只是再次向集群中所有机器发出上一次投票信息即可。
4. 统计投票。每次投票后，服务器都会统计投票信息，判断是否已经有过半机器接受到相同的投票信息，对于Server1、Server2而言，都统计出集群中已经有两台机器接受了(2, 0)的投票信息，此时便认为已经选出了Leader。
5. 改变服务器状态。一旦确定了Leader，每个服务器就会更新自己的状态，如果是Follower，那么就变更为FOLLOWING，如果是Leader，就变更为LEADING。

    Zab协议 是Zookeeper原子广播。Zookeeper 是通过 Zab 协议来保证分布式事务的最终一致性。他有三个阶段：选举、恢复、广播。其中ZK的选举就是Zab个阶段之一。

什么是服务雪崩

若果存在一条服务调用链，如果一个服务失败，会导致整条链路的服务都失败的情形，称之为服务雪崩

比如存在一条服务调用：服务A调用服务B，服务B调用服务C。如果当大量用户请求、程序bug、硬件故障等原因造成服务的提供者C不可用。那么服务B的请求就会阻塞，慢慢耗尽服务B的请求资源，B变得不可用。以此类推可能会导致整个系统服务都的不可用，这就造成了服务雪崩。

Kafka在项目中的应用

kafka 在项目中主要用于消息队列，用于异步处理消息。我在做订单提交功能的时候，就是将生产订单创建成功消息并发送 Kafka 集群，创建订单消费者完成商品库存销量、购物车记录、优惠券使用状态的更新操作。

Kafka基本架构

的 Kafka 集群中包含若干Producer，若干broker，若干Consumer Group，以及一个Zookeeper集群。Kafka通过Zookeeper管理集群配置，选举leader，以及在Consumer Group发生变化时进行rebalance。Producer使用push模式将消息发布到broker，Consumer使用pull模式从broker订阅并消费消息。

Kafka 如何存储消息

存储文件

1. kafka 将消息的基本存储单位是分区，分区内部的信息是有序的。以个topic 分为多个 partition 。Kafka将来自Producer 的数据顺序追加到 Partition 里去，这种顺序存储也避免的随机读取带来的性能损耗。
2. kafka 是使用日志文件的方式来保存Producer 发送的消息，每条消息都有一个 offset 值来表示它在分区中的偏移量。为了维护 partition 文件，通过一个 LogSegment 对 partition 的文件内容进行管理，kafka 是通过分段的方式将 Log 分为多个 LogSegment，LogSegment 是一个逻辑上的概念，一个 LogSegment 对应磁盘上的一个日志文件(.log) 和一个索引文件(.index)
3. 其中日志文件是用来记录消息。索引文件是用来保存消息的索引。索引文件中的元数据存储对应数据文件的物理偏移地址。

存储策略

4. kafka设置了两个日志清理策略。 删除修改时间在N天之前的日志（按时间删除），当 topic 所占的日志文件大于一定的阈值，则开始删除最旧的消息（按大小删除）
5. 日志压缩策略：定期将相同的 key 进行合并，只保留 key 最新的 value 值
6. Producer 生产的数据持久化到 broker 采用mmap文件映射，实现顺序的快速写入。内存映射文件将磁盘文件映射到物理内存，用户通过修改内存就能修改磁盘文件。他的工作原理就是直接利用操作系统的Page来实现文件到物理内存的直接映射。操作系统程序会主动调用flush 来把数据写入硬盘。

Kafka如何消费

1. Kafka是使用消费者来实现消费分区的信息，创建好消费者后订阅主题，消费者通过持续的轮询向Kafka请求数据，在轮询中调用 pull() 方法，返回由生产者写入Kafka 但是还没被读取过的记录，如果请求到数据则做进一步处理。
2. Kafka定义的消费者组，用于对消息进行分流，每个消费者接受主题一部分的的分区的消息
3. Topic中的消息根据一定规则将消息推送给具体消费者，主要原则：

- 若消费者数小于partition数，且消费者数为一个，那么它就消费所有消息；
- 若消费者数小于partition数，假设消费者数为N，partition数为M，那么每个消费者能消费的分区数为M/N或M/N+1；
- 若消费者数等于partition数，那么每个消费者都会均等分配到一个分区的消息；
- 若消费者数大于partition数，则将会出现部分消费者得不到消息分区，出现空闲的情况；

3. Kafka提供再均衡机制来允许分区的所有权从一个消费者转移到另一个消费者。

Kafka如何保证信息不丢失

1. 采用复制机制，分区有多个副本，Kafka会把消息写入多个副本，这样在系统崩溃时也能保证消息的持久性
2. 只有当消息被写入分区的所有同步副本的时候，他才被认为是已经提交的；只要有一个副本是活跃的那么已经提交的信息就不会丢失。消费者只是读取已经提交的信息
3. kafka也可以通过设置生产者的发送确认 ， acks = all 来保证消息在集群 leader 返回确认或者错误之前，会等待所有的同步的副本都收到消息。生产者会一直重试提交，等待所有的副本都收到消息

Exception/Error/RuntimeException/Throwable的区别与联系

Throwable 是 Java 中所有错误或者异常的超类，他的两个子类是 Error 和 Exception；
RuntimeException 它是Exception的子类。

Exception 表示一个程序员导致的错误，应该在应用程序被处理，是可以控制或者不可控制的

Error 表示系统或者底层资源的错误，可能的或应该在系统级被捕捉，是不可控制的

RuntimeException 表示虚拟机正常运行期间抛出的异常的超类。可能在执行方法期间抛出但未被捕获，属于不可控制的