项目总结

1.天机学堂项目说明

项目名：天机学堂

项目说明：

> 天机学堂是一个在线的非学历职业技能培训平台，核心业务是以售卖各种技能培训的在线课程，并提供丰富的学习辅助功能、交互功能，以提升用户学习时的氛围感和学习的积极性。

技术栈：

> SpringBoot、SpringCloud、Mybatis、MySQL、Redis、Redisson、Caffeine、RabbitMQ、XXL-JOB、腾讯云VOD（视频点播）、Nginx等

工作职责：

> - 负责部分业务的需求分析、数据库表设计，以及通用工具的封装设计。例如开发了基于Redisson的分布式锁组件，零代码侵入，基于注解实现加锁、锁类型切换、加锁策略切换、SPEL的动态锁名、限流等功能，大大提高了开发效率。
> - 参与设计和开发了学习计划和学习进度统计功能。设计了一套**高性能**的视频播放进度记录系统，在不增加数据库压力的情况下，使视频播放进度回放精度达到**秒级**。
> - 对评价系统中的点赞功能进行了系统重构：重新设计了点赞数据结构，解除了与业务的耦合，成为了一个**通用**的点赞系统。基于Redis实现点赞记录和点赞数缓存，同时基于定时任务做数据持久化，大大提高了点赞系统的**并发能力**，减轻了数据库压力。
> - 参与了积分排行榜功能的设计和开发。对用户的各种学习和交互行为做积分统计，例如签到积分、学习积分、问答积分等；并且按月累计积分形成赛季排行榜。做到了当前赛季排行榜的**实时更新**、历史赛季排行榜的**分表**持久保存。
> - 参与了优惠券系统的设计与开发。优惠券系统支持基于兑换码兑换优惠券，我设计了一套可以支持20亿量级的并且可以脱离数据库做高效校验的兑换码生成算法。同时也对领券功能进行了并发安全、并发性能的优化设计。另外还设计实现了优惠券叠加方案推荐算法，基于用户购物车中商品推荐最佳的优惠券叠加方案。

工作职责部分选择其中的2~3个填即可，不要全写。

2.可迁移的技术方案

天机学堂中包含的技术和解决方案有：

> 基于自定义注解和Redisson的分布式锁工具
>
> XXL-JOB分布式任务调度工具
>
> Caffeine本地缓存工具
>
> 支持可靠消息、延迟消息的RabbitMQ工具
>
> 延迟队列DelayQueue
>
> 基于CompletableFuture和CountDownLatch的并发任务处理方案
>
> 高并发高精度的视频进度记录和回放解决方案
>
> 学习计划和学习进度统计的学习监督方案
>
> 通用的问答（评论）功能实现方案
>
> 通用、高性能的点赞系统解决方案
>
> 高性能、低存储成本的签到解决方案
>
> 实时性强、通用性好的积分排行榜、历史排行榜解决方案
>
> 支持大数据量、高性能校验的优惠券兑换码算法
>
> 基于LUA脚本的高性能、并发安全的优惠券领取解决方案（秒杀解决方案）
>
> 优惠券叠加的智能推荐算法（MapReduce的思想）
>
> 基于Redis合并写请求并基于定时任务异步持久化的并发优化方案
>
> 基于Redis和MQ的异步写优化方案
>
> 基于腾讯VOD的视频加密、视频点播、视频审核、视频雪碧图功能（已实现未讲解）
>
> 包含支付宝支付、微信支付的多平台支付系统（已实现未讲解）
>
> 订单退款拆单处理方案（已实现未讲解）

3.可能碰到的面试题：

问题1：分布式锁

面试官：能详细聊聊你的分布式锁设计吗，你是如何实现锁类型切换、锁策略切换基于限流的？

> 好的。

> 首先我的分布式锁是基于自定义注解结合AOP来实现的。在自定义注解中可以指定锁名称、锁重试等待时长、锁超时释放时长等属性。当然最重要的，在注解中也支持锁类型属性、加锁策略属性。
>
> 我们先说锁类型切换，Redisson支持的分布式锁类型是固定的，比如普通的可重入锁Lock、公平锁FairLock、读锁、写锁等。因此我设计了一个枚举，与Redisson锁的类型一一对应，然后我还写了一个简单工厂，提供一个方法，可以便捷的根据枚举来创建锁对象。这样用户就可以在自定义注解中通过设置锁类型枚举来选择想要使用的锁类型。而我的AOP切面代码就可以根据用户设置的锁类型来创建对应锁对象了。
>
> 然后再说加锁策略切换，线程获取锁时如果成功没什么好说的，但如果失败则可以选择多种策略：例如获取锁失败不重试，直接结束；获取锁失败不重试直接抛异常；获取锁失败重试一段时间，依然失败则结束；获取锁失败重试一段时间，依然失败则抛异常；获取锁失败一直重试等。每种策略的代码逻辑不同，因此我就基于策略模式，先定义了加锁策略接口，然后提供了5种不同的策略实现，然后为各种策略定义了枚举。接下来就与锁类型切换类似了，在自定义注解中允许用户选择锁策略枚举，在AOP切面中根据用户选择的策略选择不同的策略实现类，尝试加锁。
>
> 至于限流功能，这里实现的就比较简单，就是在自定义注解中加了一个autoLock的标识，默认是true，在AOP切面中会在释放锁之前对这个autoLock做判断，如果为true才会执行unlock释放锁的动作，如果为false则不会执行；所不释放就只能等待Redis自动释放，假如锁自动释放时长设置为1秒，那就类似于限流QPS为1

面试官追问：那你的设计中是否支持Redisson的连锁（MultiLock）机制呢？

> 这个锁我知道，它需要利用多个独立Redis节点来分别获取锁，主要解决的是Redis节点故障问题，提高分布式锁的可用性。但是性能损耗比较大，因此我们的设计中并没有支持MultiLock。

面试官追问：那你知道Redisson分布式锁原理吗？

> 分布式锁主要是满足多进程的互斥性，如果是简单分布式锁只需要利用redis的setnx即可实现。但是Redisson的分布式锁有更多高级特性，例如：可重入、自动续期、阻塞重试等，因此就没有选择使用setnx来实现。
>
> Redisson底层是基于Redis的hash结构来记录获取锁的线程信息，结构是这样的：key是锁名称，hasKey是线程标示，hashValue是锁重入次数。这样就可以实现锁的可重入性。
>
> 然后Redisson的分布式锁允许自定义锁的超时自动释放时间，如果没有设置或者设置的值为-1，则自动释放时间为30秒，并且会开启一个WatchDog机制。WatchDog就是一个定时任务，每隔（leaseTime/3）秒就会执行一次，会重置锁的expire时间为30秒，从而实现所的自动续期
>
> 至于阻塞重试机制，则是基于Redis的发布订阅机制。如果设置了waitTime大于0，则获取锁失败的线程会订阅一个当前锁的频道，然后等待。获取锁成功的线程在执行完业务释放锁后会向频道内发送通知，收到通知的线程会再次尝试获取锁，重复这个过程直到获取锁成功或者重试时长超过waitTime

面试官追问：那基于Hash结构如此复杂的业务逻辑来实现，代码肯定不止一行，如何保证获取锁逻辑的原子性？

> 答：这个问题也很好解决，Redisson底层是基于LUA脚本实现的，在Redis中，LUA脚本中的多行代码逻辑执行是天然具备原子性的。

问题2：视频点播

面试官：你们是在线教育，那视频在线点播、视频加密等功能是如何实现的，你们是如何避免视频被盗播的?

> 视频上传、播放等功能并不是我负责的，不过我也有一定的了解。我们的视频处理都是基于腾讯云VOD的视频点播服务。其中有非常全面的视频任务流，只要配置好任务流的工作，例如视频加密、视频封面生成、视频雪碧图生成、视频内容审核等，在视频上传后这些工作都会自动完成，无需我们自己处理。
>
> 我们只需要实现视频上传即可。而且视频上传也无需在服务端上传，服务端之只是提供了一个授权签名，腾讯云提供了前端JS的SDK工具，前端拿到我们的授权签名以后，可以利用工具直接从客户端上传，不会增加服务端的压力。
>
> 另外腾讯云VOD还提供了一个视频播放器，播放视频时无需告知其视频地址，而是在服务端给一个授权码和文件id，剩下的视频就由视频播放器与腾讯云服务交互。整个视频数据传输的过程都是加密进行，视频内容也无法下载。
>
> 不过如果用户自己录制电脑屏幕那就没办法控制了。

问题3：视频进度统计

面试官：能不能介绍一下你说的视频播放进度统计功能，你是如何保证进度回放的高精度的?

> 好的。
>
> 视频的播放进度记录分为登录用户和未登录用户两种情况，对于未登录用户，其视频播放进度只能是在客户端保存，有前端同时来实现。我重点说说登录用户的视频播放进度统计功能。
>
> 首先，视频播放进度的记录要考虑的用户异常退出的场景，因此我们不能依赖视频停止播放、浏览器退出等前端事件来提交播放进度，而是应该由前端定期的提交播放进度，类似一种心跳机制。
>
> 同时，由于要实现视频播放进度的秒级精度，因此这种心跳必然要维持一个较高的频率，频率越高则回放的时间精度越高。不过还要考虑到服务器压力问题，因此频率也不能太高，例如我们项目中设置的是15秒一次心跳。服务端在接收到前端提交的播放进度信息后，将其持久化保存即可。
>
> 当然，这些播放进度不能直接写入数据库，因为在用户访问的高峰期，对数据库来说压力还是太大了。所以我们采用的策略是将播放进度信息先写入缓存当中，再异步的持久化到数据库中。这是很常用的一种并发优化思路：先写缓存，再异步持久化到数据库。而异步持久化通常会采用定时任务来实现，但在这里却存在一些问题：第一，我们要保证播放进度的高精度，定时任务如果频率较低，则精度不足。定时任务频率较高则会增加数据库压力。第二，这里的播放进度信息有一些特殊性，因为播

> 放进度不需要每一次的都记录到数据库，而是只需记录用户离开某个视频时最后一次提交的播放进度即可。如果定时任务频繁执行，用户离开视频前持久化到数据库的操作属于无效操作，增加了数据库负担。
>
> 综上，用户每次提交的视频播放进度先缓存到Redis，但是却不应该立刻持久化到数据库。而是应该在用户离开视频时最后一次提交播放进度再持久化到数据库。但问题是该如何判断用户是否是最后一次提交呢？
>
> 这里有两种方案：
>
> 方案一，基于时间做判断。只要用户依然在播放视频，那么Redis中的进度就会每隔15秒变化一次。因此我们只需要在缓存播放进度的同时，记录本次更新的时间戳。定时任务每隔20秒执行一次，但不是立刻更新数据库。而是要先判断Redis缓存中的时间戳与当前时间相差是否超过15秒，超过了则说明用户已经停止提交播放进度，说明当前是最后一次提交，我们写入数据库。否则放弃本次任务即可。
>
> 方案二，基于进度做判断。只要用户依然在播放视频，那么Redis中的进度就会每隔15秒变化一次。因此只要Redis中数据不变了，就说明用户停止播放视频了。因此，前端提交播放进度的时候，我们除了要缓存播放进度到Redis以外，还需要提交一个20秒后执行的延迟任务，任务中记录本次播放进度。将来任务执行的时候与缓存中的进度做比较，如果发生了变化，则证明用户依然在播放视频，放弃本次任务。如果没有变化，则证明用户停止播放视频了，则持久化到数据库。
>
> 考虑到方案一除了定时任务以外，需要一个额外的任务队列来记录发生变更的视频信息，增加了实现复杂度。所以最终我采用了第二种方案。

问题4：点赞系统

面试官：能讲讲你的点赞系统设计吗？

> 答：

面试官追问：能详细说说你的Redis数据结构吗？

> 答：

问题5：积分排行榜

面试官：你们是如何保证积分排行榜的实时性的？

> 答：

面试官追问：那如果数据量很大，所有排行榜数据都放入Redis的一个key，是不是太大了？

> 答：

面试官追问：那历史排行榜数据量越来越多，全都放入一张表吗？如何处理海量数据存储和查询的高效性？

> 答：

面试官追问：那为什么你们没有选择基于开源框架，例如ShardingJDBC来实现分表呢？

> 答：

问题6：优惠券系统

面试官：能不能讲一讲你说的优惠券兑换算法？

> 答：

面试官：你是如何避免优惠券的超发的（你是如何避免商品库存超卖的？）？

> 答：超发与商品库存超卖类似，解决方案也基本一致，...
>
> 不过券超发或特殊商品还有一个限领或限买数量的问题，也就是说某张券每人限领一张，或者某个商品每人限买一个。这里就不能采用乐观锁机制了，...

面试官追问：加了锁以后性能肯定会有下降，如果无法满足当前业务的并发需求，你有什么改进的思路？

> 答：提高并发的手段有很多，例如：...

面试官：能不能聊一聊你们如何实现优惠券的叠加推荐的？

> 答：

问题7：通用问题

面试官：你在项目中有用过线程池吗？

> 答：

面试官：你在项目开发中有没有遇到过什么疑难问题，最后是怎么解决的？

> 答：

面试官：你有没有过高并发性能优化的经验？

> 答：

面试官：你有没有自己设计过一些组件或者工具？

> 答：