### 头条项目串讲

面试官你好，我叫XXX，X年本科/专科毕业，做了X年Java开发，我的性格是XXX，平时爱好是XXX，然后我最近做的一个项目是XXX，主要使用的技术是SpringBoot+MybatisPlus，然后用的关系型数据库是MySQL，非管理型数据库是Redis+MongoDB，搜索用的ES，微服务框架用的是SpringCloud Alibaba，MQ用的是Kafka，这个项目的名字是XXX，是用来做XXX功能的。

然后这个项目分成三个端，分别是APP端、自媒体端、管理员端，我自己参与了APP端和自媒体端的接口开发，然后我在自媒体做了XXX功能，在APP端做的XXX功能。

下面我先从自媒体端讲，自媒体端有登录功能，登录功能首先要对用户的密码，加密算法可以分成可逆算法和不可逆算法，可逆算法是可以根据密文解析明文，不可密算法是不可以根据密文解析明文，然后可逆算法又分为对称加密算法和非对称加密算法。然后对称加密算法是加密与解密用的同一个密钥。非对称加密算法是公钥加密，私钥解密或者相反，然后我在做登录功能的时候，用到Bcrypt算法,然后它是不可逆算法，非常安全，然后在验证成功以后就会生成token，返回给前端。这是用户模式登陆，如果是游客模式的话，就会直接返回token。

然后接下来是网关，然后网关可以解决哪些问题，第一它可以解决前后端分离式的快速对接。第二它可以进行统一的限流，鉴权、跨域和日志处理，然后我自己做了跨域和鉴权两个功能。我选了一个全局的过滤器，过滤器功能有两个，第一是对登录请求直接放行，第二是对于非登录请求，判断他们的token是否有效放行，无效返回。然后提到token， token的全称是JSON Web Token，它是一种协议，定义了token的格式，使用token有三个好处:

第一，避免了使用session导致的重复登录问题;第二解决了多服务共享数据的问题;第三可以节省大量的服务器内存。因为token，它可以存在浏览器的cookie中，不用存在服务器上。

然后在网关之中我还做了一个功能就是说为了后续的微服务提供用户的信息：  
第一，用token加拦截器加thread local做了一个存储用户信息的功能，便于后面的微服务使用用户信息,判断是否登录。第一把token的用户信息放在放行之前，存到HTTP请求头中。  
第二，然后通过在拦截器之中，把请求头中的信息存到thread local中，因为thread local和本地线程是绑定的，而拦截器、controller，service、dto在同一个线程之中。

然后我还做了一个素材管理的功能，用minio来管理图片。自媒体用户可以上传素材、收藏、删除、查看、排序等等一系列功能。然后当时技术选型的时候，参考阿里云、七牛云，FastDFS，还有minio。阿里云、七牛云本身自身开发成本低也很好用，但是收费，然后 FastDFS 可以搭集群，也可以动态扩容，但是使用比较麻烦，最近几年没有更新，之后我选择了minor可以搭集群，使用也方便，而且还免费，但是需要自己在服务器上自己安装这个服务，自己维护。然后整个功能比较简单，然后我就不讲了。

然后说自媒体的文章发布功能，发表文章那边可以填写文本和图片，这是自媒体用户自己填写的，然后我可以选择保存为草稿还是直接发布，直接发布那里还可以选择发布的时间是当前时间还是未来时间，后面在审核的时候会用到这个时间。

然后文章发布完之后就是文章的审核，然后审核整体是用异步线程池来执行的，然后使用线程池主要是为了节省发布文章的流程，节省时间，快速响应给前端。

线程池有7大参数：核心线程数、最大线程数、阻塞队列，然后非核心线程生存时间、时间单位、拒绝策略和线程工厂。然后当时我们的核心线程数是10，然后最大线程是100，阻塞队列是500，非核心线程生存时间是10秒，拒绝策略是让调用者执行任务，然后这是整体的一个线程池。然后整整个审核用到了OCR技术的一种叫Tess4j，本地的文本审核用到了敏感词过滤算法DFA，然后还用到了阿里云的文本审核和图片审核功能，然后还用minio。业务逻辑是这样的，先把文本文章中的封面图片，还有文章图片从minio上面下载下来，然后做一个去重，然后用 OCR识别技术把所有的图片里面的文本都识别出来，然后和文章的内容就一起进行一个本地的文本审核，如果通过了，然后再进行阿里云的文本审核和图片审核，如果全部通过，根据当前时间判断，是否是直接发布，还是加入定时任务以后再发布。

好，我先讲直接发布，直接发布就是说把文章的状态改成8（审核通过待发布），然后就调用 APP端的feign接口，然后创建 APP端的文章，这个文章中它应该是9（已发布）。然后文章表， APP端的文章表我采取了分布式的雪花算法，然后因为什么？因为文章的数据量特别大，然后将来可能要分库分表，使用数据库中间接技术进行连接。  
如果采用了 Id的自增策策略，在新增文章的时候就可能产生重复的ID，所以会要采用分布式的雪花算法。  
然后在创建APP端文章的时候，我遇到了一个很大的问题，就是说在查询首页文章的时候，数据库里边会查询在数据库会查询自媒体的文章表，因为文章表的内容字段就是长文本，在高并发的时候出现了内存溢出的问题，然后我就这么解决的：我把文章表就分成了主表和内容表，然后在展示文章的时候就不查内容，就解决了这个问题。为了提高文章的更改时候的效率，我把文章的状态也拆出来了；然后为了进一步提升查询文章的效率，我就把文章的那些内容就是长文本，用静态化模板引擎技术framework生成了一个静态网页，然后传到了minio之中，然后就把返回来的路径传到文章表之中。这个时候查询文章详细页就不用到数据库去查，就直接去minio上面查，就减轻了数据库的压力，这就是审核文章审核通过之后立即发布的逻辑。

然后现在接着讲未来发布的逻辑，我把未来发布整体做成了一个定时任务，用到了spring task，mysql加redis的技术,用到了分布式锁，然后在集群部署的时候，防止定时任务的冲突。整个功能是由4个方法构成的:第一个是添加任务，第二是同步刷新未来队列到当前队列,第三是从当前队列拉取任务,第四是用数据库同步的redis，而这一步是为了防止服务器宕机，做的数据的安全性的保存的操作。

首先了解第一步就是添加任务，添加任务就是把任务添加到数据库之中，然后还有添加到redis缓存之中，然后未来队列我用的是zset，然后用发布的时间戳当作score，用list当做当前队列，然后就是说同步刷新未来队列到当前队列，这个功能就是说去未来队列里边根据时间key和时间戳查询哪些任务过期了，然后就把它加到加入到当前队列之中，然后在 redis key的子匹配上，我就没有使用keys模糊匹配，因为这个功能虽然强大，但是在开发中会导致redis的CPU使用率很高，然后redis是单线程就会被堵塞，然后我使用了那个scan命令就是非常快。

然后我用了 spring task技术把 cron表达式设置为10秒一次，然后用了分布式锁通过互斥来保证数据的一致性。具体的来说就是 set nx的指令，然后最后我用管道技术pipeline给优化了一下，这是第二功能。

第三功能是我从当前队列拉取任务执行，就拉取任务就是说要把当前队列的任务删掉，删掉任务表的记录，然后更新任务的日志表的记录为已执行，然后在自媒体端调用定时任务拉取任务的时候，也设置成5秒一次，并且加上分布式锁然后调用APP端的发表文章任务，这个第四功能就是说用数据库同步的redis，是因为 redis的持久化机制**RDB**在当机的时候就可能会丢失数据，然后**AOF**的默认策略在宕机时会丢失一秒的数据，所以在服务器重启后就会立即重启，启动立即同步表中的在这些任务到redis当中，就是给那个方法加上一个注解叫@postconstruct（**注释用于在依赖关系注入完成之后需要执行的方法上,以执行任何初始化**），就会在应用自动执行。好，这是数据库同步的redis的功能。

然后文章同步结束之后，还有一个文章上下架的功能，然后这里我采用了kafka传递消息，可以达到异步削峰解耦的功能，其中kafka的那种高可用设计就可以保证安全性，就为什么用kafka就没有那种RabbitMQ呢？虽然RabbitMQ也可以满足需求，就是说这是为了保证以后服务的访问量可能特别大，如果你现在用 RabbitMQ可能可以，但是如果到了高并发的时候，只有kafka才能实现这种高并发高可用，所以说这是为了以后来准备的。

但是我当时还遇到一个问题就是说，由于生产者网络抖动，就没有把消息发送到kafka服务器，我是这么解决的，就说定义一个本地的消息日志表，然后在消息发送之前就把它存进去，在消息发送成功之后，再把消息表中的消息删除，然后就把这功能做成定时任务定时扫描，这个表就自动补偿就可以解决kafka消息发送失败，这个问题生成者发送失败的问题，然后计算热点文章的功能，是不是？

传统的定时任务spring task就是说存在一些问题，就说他做集群任务时候重复执行，就算你加上 redis锁也特别麻烦，每一个都要加。

第二是你那个cron表达式就是定义在代码之中，如果调度规则需要更改，就还得改代码重新上线，太麻烦，所以我选择了用xxl- job就实现在线更新。

这功能就这么完成的：  
查询了最近5天的所有文章，然后给他们计算了1个峰值，然后把每一个频道都存入30条文章然后把它传入到redis，然后对那个查询文章的那个算法做一个修改，当你查询首页的时候，查询redis时，当你查询其他样的时候就直接去查数据库。

好，最后就是评论系统的功能，但是评论系统我用了那个mongo DB，为什么用mongo DB因为那个mongo DB的内存特别大，可以放海量数据，但redis的内存特别小，而且评论功能就是对这个数据的安全性要求也没那么高，就是可以忍受丢失一部分评论啊，这个问题也不太大。然后他mongo DB还有几个特点就是说它高读取、高存储、高扩展，然后高可用，所以我选择了mongo DB。

好，大概就是这样。