自我介绍（需要可以参考我三七互娱那篇面经）
因为项目提到了多人合作，说一下怎么多人合作
由于当时是在疫情期间，没法见面，我们采用 git 进行一个项目版本控制和多人合作
我们首先采用腾讯会议的形式确定了需求及功能，之后确定了函数命名以及接口，后台传入前端的数据格式。并且根据功能分配了各自的任务。
在项目编码进行中，我们采用独立开发的方式。我建了个仓库，然后给其他成员权限，每个人有一个分支，写好对应功能就合并入分支，然后解决冲突。
第一次比较正式的多人线上合作，很多功能和方式还没有很好的使用到，希望之后进入公司之后可以好好学习在一个项目上的合作工作方式。
讲一个你最近在做的项目
其实我最近都在准备面试，而且也要参加字节和学校合作的一个项目，没怎么做项目的，最近一个就是当时学习的开源项目 RuoYi，那我就跟他说了一下学习这个开源项目的方式
首先这是一个后台管理系统的框架，主要使用到了 SpringBoot，和 Shiro 等。
首先我查看了该项目的 pom 文件，查看其依赖情况，因为依赖很大程度展示了该项目的技术栈，那么我就根据这些依赖对陌生的知识进行一个熟悉，整理总结。
然后我查看了在线文档，先看一下其模块和文件组织架构，知道模块功能和分布。其次我根据文档的主要功能描述，一个个找到对应实现的代码，进行分析。
后来我发现太多知识，所以我选择以一种小白的视角去记录该项目对应功能的实现，包括的就有分页功能，多数据源，shiro 的缓存控制等，然后配合上里面一些算法的解析，数据库的表整理等合成了自己的后台开发手册。
之后也有给项目代码提出过问题并得到了作者的回复与修改，算是真正参与到了该项目吧。
之后面试官跟我说，也可以不要再 issue 中提出，有时候你有改好的代码，或者不错的代码可以提交到 Pull Request 里面，如果对***得好的，会直接采用。
mysql 防止注入，安全性了解吗
SQL 注入即是指 web 应用程序对用户输入数据的合法性没有判断或过滤不严，攻击者可以在 web 应用程序中事先定义好的查询语句的结尾上添加额外的 SQL 语句，在管理员不知情的情况下实现非法操作，以此来实现欺骗数据库服务器执行非授权的任意查询，从而进一步得到相应的数据信息。
防止的方式：
像 mybatis 有一个用#{}去代替${}的方式。前者解析传递进来的参数数据，并且进行一个预编译处理，而后者是字符串替换。
还有就是像我之前做过的开源项目，里面采用的就是一种正则匹配的方式去识别 sql 语句，只能包含某种字符，不能包含像‘；’这种特殊的字符等等。
其次我个人觉得在一些比较敏感的语句中，我们可以把用户自定义的输入改成我们定义好的按钮选择，这样的话就可以防止恶意注入。
还有就是可以在数据传输过程中加上一个过滤器，利用注入的语句具有的特征，过滤掉一些有危险的语句，提示用户重新输入。
讲讲 MySql 的 B+树
特征：（这不是我的回答，而我也认为不该回答这些，只是放出来让大家知道）
有 n 棵子树的非叶子结点中含有 n 个关键字（b 树是 n-1 个），这些关键字不保存数据，只用来索引，所有数据都保存在叶子节点（b 树是每个关键字都保存数据）。
所有的叶子结点中包含了全部关键字的信息，及指向含这些关键字记录的指针，且叶子结点本身依关键字的大小自小而大顺序链接。
所有的非叶子结点可以看成是索引部分，结点中仅含其子树中的最大（或最小）关键字。

通常在 b+树上有两个头指针，一个指向根结点，一个指向关键字最小的叶子结点。
同一个数字会在不同节点中重复出现，根节点的最大元素就是 b+树的最大元素。
我是从 B+树的优势去讲的：
首先 B+树的查询效率是和 AVL 平衡二叉树差不多的，甚至更低一点。但是他相对二叉树的优势就是它的结点存储的数据更多。对于 mysql 来说，是以页进行存储的，每个结点的获取可能就会进行对应的一次 IO，而 IO 效率相对于内存慢，所以希望获取的数据尽量多，所以才采用 B+树的形式。
b+树的中间节点不保存数据，所以相对于 B 树磁盘页能容纳更多节点元素，更“矮胖”
b+树查询必须查找到叶子节点，因为之后叶结点是存储数据的，查询效率会相对更加的稳定。
对于范围查找来说，b+树只需遍历叶子节点链表即可，b 树却需要重复地中序遍历。对于我们的哈希索引，甚至无法进行一个最左匹配和范围查询。
而且对于新版本的一个索引下推的操作，也是 B+树比较好。
说一下 mysql 的一二级索引
这个最好从两种常见的数据库引擎介绍起
MyISAM：
文件三个：.frm，.MYD，.MYI
索引结构为 B+树，索引和数据分开，索引指向了文件地址，这是非聚集索引。
主键索引和辅助索引结构相同，在叶子结点存放的是索引和文件指针。
Innodb：
文件两个：.frm，.ibd
索引结构为 B+树，索引结构中储存的是实际的数据，为聚集索引。
主键索引在叶子结点存放的是索引和数据内容，而辅助索引存放的是索引和主键。
期间问到了聚集索引的优势，我提到了与数据放在一起，也就是与物理顺序一致，更有利于查询（我一直以来理解都是那字典的拼音查找和笔画查找来比较的，拼音查找其实就是所谓的聚集索引）
redis 书 bitmap 位图
一开始说有用到 redis 吗，我说有学习到，也有启动过，但是没有在自己的项目中使用过，学习的比较多是数据结构的底层
位图我只是用过它的操作，其实并不了解，所以没有回答出来，以下是我网上查找总结的，积累下：
BitMap，即位图，其实也就是 byte 数组，用二进制表示，只有 0 和 1 两个数字。
重要 API：
复制代码

命令	含义
getbit key offset	对 key 所存储的字符串值，获取指定偏移量上的位

1（bit）
2 setbit key offset value 对 key 所存储的字符串值，设置或清除指定偏移量上的3位（bit）
 1. 返回值为该位在 setbit 之前的值
2. value 只能取 0 或 1 <br 4/>3. offset 从 0 开始，即使原位图只能 10 位，offset 可以取 1000
bitcount key [start end] 获取位图指定范围中位值为 1 的个数
如果不指定 start 与 end，则取所有
bitop op destKey key1 [key2...] 做多个 BitMap 的 and（交集）、or（并集）、not（非）、xor（异或）操作并将结果保存在 destKey 中

bitpos key tartgetBit [start end] 计算位图指定范围第一个偏移量对应的的值等于 targetBit 的位置
1. 找不到返回-1
2. start 与 end 没有设置，则取全部
3. targetBit 只能取 0 或者 1
应用场景：
统计每日用户的登录数。每一位标识一个用户 ID，当某个用户访问我们的网页或执行了某个操作，就在 bitmap 中把标识此用户的位设置为 1。
相对 set 来说会占用空间会小，速度也会快，但是也要考虑偏移量，可能有较大的消耗。
哨兵机制
如果主服务器挂了，我们可以将从服务器升级为主服务器，等到旧的主服务器(挂掉的那个)重连上来，会将它(挂掉的主服务器)变成从服务器。
我整理面经会总结比较详细，面试时候只需要回答出重点就好了
Sentinel 本质上只是一个运行在特殊模式下的 Redis 服务器。
Sentinel 在初始化的时候并不会载入 AOF/RDB 文件，因为 Sentinel 根本就不用数据库
然后，在启动的时候会将普通 Redis 服务器的代码替换成 Sentinel 专用代码。(所以 Sentinel 虽然作为 Redis 服务器，但是它不能执行 SET、DBSIZE 等等命令，因为命令表的代码被替换了)
初始化 Sentinel 的状态，并根据给定的配置文件初始化 Sentinel 监视的主服务器列表。
最后，Sentinel 会创建两个连向主服务器的网络连接：
命令连接(发送和接收命令)
订阅连接(订阅主服务器的 sentinel:hello 频道)
Sentinel 通过主服务器发送 INFO 命令来获得主服务器属下所有从服务器的地址信息，并为这些从服务器创建相应的实例结构。
当发现有新的从服务器出现时，除了创建对应的从服务器实例结构，Sentinel 还会创建命令连接和订阅连接。
在 Sentinel 运行的过程中，通过命令连接会以每两秒一次的频率向监视的主从服务器的 sentinel:hello 频道发送命令(主要发送 Sentinel 本身的信息，监听主从服务器的信息)，并通过订阅连接接收 sentinel:hello 频道的信息。
这样一来一回，我们就可以更新每个 Sentinel 实例结构的信息。
判断下线：
主观下线
Sentinel 会以每秒一次的频率向与它创建命令连接的实例(包括主从服务器和其他的 Sentinel)发送 PING 命令，通过 PING 命令返回的信息判断实例是否在线
如果一个主服务器在 down-after-milliseconds 毫秒内连续向 Sentinel 发送无效回复，那么当前 Sentinel 就会主观认为该主服务器已经下线了。
客观下线
当 Sentinel 将一个主服务器判断为主观下线以后，为了确认该主服务器是否真的下线，它会向同样监视该主服务器的 Sentinel 询问，看它们是否也认为该主服务器是否下线。
如果足够多的 Sentinel 认为该主服务器是下线的，那么就判定该主服务为客观下线，并对主服务器执行故障转移操作。
选举头 Sentinel 和故障转移：
当一个主服务器认为为客观下线以后，监视这个下线的主服务器的各种 Sentinel 会进行协商，选举出一个领头的 Sentinel，领头的 Sentinel 会对下线的主服务器执行故障转移操作。

选举领头 Sentinel 的规则也比较多，总的来说就是先到先得(哪个快，就选哪个)
在已下线主服务器属下的从服务器中，挑选一个转换为主服务器
让已下线主服务器属下的所有从服务器改为复制新的主服务器
已下线的主服务器重新连接时，让他成为新的主服务器的从服务器
挑选某一个从服务器作为主服务器也是有策略的，大概如下：
跟 master 断开连接的时长
slave 优先级
复制 offset
run id（上一个主服务器 id）