问：java 中序列化是怎么实现的呢？

1.实现 Serializable 接口，就会实现数据序列化的效果。

2.调用 json 做序列化。（就比如：Jackson，fastjson 等等）

3.实现 Enternalizable 接口，就可以实现反序列化的效果。

问:java 的流有哪些呢？
从方向方面，主要就是输入流和输出流。

从单位方面，主要就是分为字节流和字符流。字节流主要就是 InputStream，OutputStream。字符流主要就是 Reader，Writer。

问：抽象类和接口有什么区别呢？
从方法编写方面，抽象类中可以抽象方法和普通方法，而接口中只能编写抽象方法。

从继承和实现方面，抽象方法只能继承一个类并且可以实现多个接口，而接口可以继承多个接口。

在变量的定义方面，接口只能定义静态变量，抽象类可以定义普通变量和静态变量。

问：final 关键值有了解过吗？
在修饰方法的时候，说明该方法无法被重写。

在修饰类的时候，说明该类无法被继承。

在修饰属性的时候，说明该变量从创建到销毁过程中不会改变。

在修饰形参的时候，说明该形参的引用在方法执行前后都会不发生改变。

问：异常类有哪些？
异常主要就是 Exception，Error。

Exception 主要就是运行时异常和检查时异常，为了保证代码恶的正常运行，我们需要主动使用 try/catch 进行捕获，常见的就是 IoEception，SQLExceptinon。

Error 主要就是就是发生在运行时的异常，就比如：oom，threadDeath。

问：Filter 和 Interceptor 有说明区别？哪个先执行呢？
在作用域方面，Filter 作用域每个请求，而 interceptor 主要就是对 Controller 方法的请求起作用。
在执行时机方面，filter 会在请求加入 dispatchServlet 前和返回给客户端前执行，而 interceptor 会在执行 Controller 方法的时候和进行视图渲染前执行。

Filter 会像执行而 interceptor 随后执行。
问：java 中共享变量有哪些?
1.静态变量。

2.使用 Voliate 修饰 java 中得变量就会变成共享变量。

篇幅限制下面就只能给大家展示小册部分内容了。整理了一份核心面试笔记包括了：Java 面试、Spring、JVM、MyBatis、Redis、MySQL、并发编程、微服务、Linux、Springboot、SpringCloud、MQ、Kafka 面试专题

需要全套面试笔记及答案【点击此处即可】免费获取

问：编写 sigleton 函数?（单例模式）
饿汉式：

```
public class sigleton {

//饿汉式,每次都会创建好,即使没有调用

private final static Test test = new Test();

public Test getInstance() {

return test;

}

}
```

AI 助手

懒汉式：

```
public class sigleton {

//懒汉式，只有在被第一次调用时才会被创建

private static Test test;

public Test getInstance() {

return test == null ? new Test() : test;

}
```

}

AI 助手

Redis 八股文
问：Redis 数据结构的底层实现由了解过吗？
String 类型的底层使用动态字符串实现的。

List 类型的底层使用双向链表+压缩数组实现的。

set 类型的底层使用哈希表+数组整数数组实现的。

hash 类型的底层使用压缩数组+哈希表实现的。

zset 类型的底层使用压缩数组+跳表实现的。

问：zset 的底层结构有了解过吗？
zset 的底层结构主要就是跳表+压缩列表。

压缩列表：本质就是一个数组，在其中记录 列表的长度，尾部得偏移量，列表的个数。

跳表：就是基于二分查找的思想，对链表创建多级索引，就是基于每隔一个节点做索引，以此类推，创建索引的层数就是 logn + 1。整个查找过程就是在多级索引间跳来跳去。

问：我看你在做项目的时候都使用到 redis，你在最近的项目中哪些场景下使用 redis 呢？
缓存和分布式锁都有使用到。

问：说说在缓存方面使用
1.在我最写的物流项目中就使用 redis 作为缓存，当然在业务中还是比较复杂的。

2.在物流信息查询模块使用中使用二级缓存，一级缓存使用的是 Caffeine，二级缓存就是使用 redis。

问：对缓存击穿，缓存雪崩，缓存穿透有了解过吗？说说者三个缓存问题的解决方案吧。
1.缓存击穿：某个热点 key 设置了过期时间，在高并发查询的情况下，该热点 key 过期了，导致大量的请求去访问数据库，最终压垮数据库。

解决方案：

（1）不给热点 key 设置过期时间（在 redis 中设置的过期时间都是逻辑过期时间，通过逻辑字段来判断 key 是否过期）。

（2）使用分布式锁，保证每次只有一个请求去访问数据库，在每次访问数据库前再做一次查询缓存的操作，然后获取锁并去数据库查询，将查到的数据重新缓存起来，下一次请求就会在缓存中查询到数据，就不会查询数据库，防止压垮数据库。

2.缓存雪崩：第一种情况:大量的 key 设置了相同的过期时间，在同一大量的 key 失效，导致大量的请求去访问数据库，导致压垮数据库。第二种情况:redis 宕机。

解决方案:

(1) 错开 key 的过期时间（TTL）：在每个 key 的统一过期时间上在随机加上 1~5 分钟的过期时间。

(2)设置服务降级，服务熔断，服务限流，到达阈值的时候直接返回自定义的错误信息。(作为系统的兜底策略)

(3)为 redis 搭建集群,就包括哨兵模式，集群模式。

(4)做二级缓存。(可以重点介绍，就引导面试官到运单信息查询模块，给他介绍二级缓存的实现还有缓存同步的问题，主要讲 caffeine 缓存同步的问题)

3.缓存穿透:查询的 key 在缓存和数据库中都不存在，每次都会进入数据库查询，当大量的这种 key 访问数据库，就会导致缓存穿透。(这种情况多半上是恶意攻击)

解决方案:

(1)在缓存中储存空值：当在数据库中没有查到数据时，将 key 关联 null 的键值对存储到缓存中，后续就会走缓存，这种解决方案缺点很明显，就是存储大量无用的存储，浪费空间。

(2)使用布隆过滤器：在查询缓存的时候先去布隆过滤器中查询是否存在缓存，再去查询缓存。

问：具体讲讲你对这个布隆过滤器理解?
布隆过滤器类似 hash 表叫做 bitmap(位图)，在没有位置存放的是二进制的数据，1 表示存在，0 表示不存在。通过哈希函数计算储存位置。在后续查询缓存前先去布隆过滤器中查询数据是否存在，如果存在才去查缓存，如果不存在就直接返回空。

布隆过滤器的优点就是：储存二进制的数据，而非真实的数据，查询速度快。缺点：判断数据是否存在有误判率，不能做删除操作。

为了降低布隆过滤器的误判率，因为两个数据的 hashCode 可能相同，所以我们可以设置多

哈希函数操作，较低过滤器的误判率。

我们主要是根据 redisson 设置布隆过滤器，可以设置其误判率。在高并发的场景下，一般误判率控制在 5%之内就可以了。

问：说说分布式锁方面的使用
在项目的支付模块中的扫码支付为了保证交易单的状态不会被除当前业务的其他业务修改，计算运力，我们都会使用分布式锁。

问：你能简单的讲述一下分布式锁的实现吗？（引导面试官到你的项目中，去解释）
分布式锁

可以跟面试官说说优惠券超卖的问题。

在 redisson 中的获取锁方法底层主要是通过 Lua（能够调用 redis 命令，保证命令执行的原子性）实现的，如果获取锁方法没有设置过期时间，则分布式锁会有 watch dog 来保证延长锁的有效时间。

还可以通过 setnx 来实现分布式锁(因为 redis 是单线程的)。 通过 set If not exists 并设置过期时间实现分布式锁，但是存在的问题就是无法确定要给锁设置多长的有效时间，所以在项目中比较少使用。

公平锁 (我们项目中使用的分布式锁，保证可重入性)

公平锁主要还是基于 redisson 实现的。锁的结构采用的是 hash 结构，以大小 key 的形式，在我业务中，使用订单 id 和交易单 id 进行拼接作为大 key，使用当前的线程 id 作为小 key,vlaue 存放的就是上锁的次数。在业务修改交易单状态是需要先判断当前交易单是否被上锁（通过订单 id，交易单 id，进程 id 来获取对应的锁），如果被上锁，则当前线程的其他业务不能进行操作。保证订单状态的准确性。(判断幂等性) 保证可重入性，根据当前线程 id 来获取分布式锁，解决死锁的情况（分布式锁的底层实现）。

问：分布式锁可以实现重入吗？
重入的意思就是：在同一个线程中获取锁后继续获取锁。因为我们分布式锁中小 key 使用的是线程 id，value 就是上锁的次数，我们每次进行重入时就是 value 值+1，在释放锁的时候，就将对应的 value-1，在 value 为 0 的时候就删除对应的信息。其他微服务就可以获取此分布式锁了。通过线程 id 的不同来保证分布式锁的互斥。(蓝字选答)

问：redisson 实现的分布式锁，可以解决主从一致性的问题吗？

不能解决主从一致的问题，单 master 节点获取锁后就没释放锁就宕机了，此时 slave 节点变成了 master 节点，因为新的 master 没有上锁，所以新的 master 会进行上锁，破坏了锁的互斥性。为了解决这个问题我们可以使用红锁，也就是给一般以上的节点添加分布式锁，但是这样效率就变的很低，为了提高效率，建议采用 zookeeper 实现分布式锁。(听说过 zookeeper 实现分布式锁)

问：在集群的项目中为什么不能用关键字 synchronized 呢？

在两个微服务中，如果使用 synchronized 是无法达到同步上锁效果，因为两个微服务是两个单独的 JVM。

问：了解过双写一致性吗？
双写一致性：当数据库中的数据发生修改的时候，我们需要修改缓存中的数据，保证数据库和缓存中的信息相同。

在读操作的时候，会先到缓存中查询数据，如果没有命中的话就到数据库中查询。

在写操作的时候，会采用延迟双删。

问：在延迟双删中为什么要延迟删除缓存？
数据库采用的是主从模式，遵循读写分离，需要一些时间来将主数据库中的数据同步到从数据库，但是还是可能出现脏读的情况。

问：在延迟双删中为什么要删除两次缓存呢?
如果只有一次删除缓存操作的话就会有两种情况。

1.情况 1：先删除缓存，再修改数据库。线程 1 删除缓存。在线程 1 要修改数据库前。线程 2 去做查询操作，发现没有缓存，就去数据库中查询数据并做缓存，之后线程 1 修改了数据库中的信息，最终出现缓存数据和数据库数据不相同的情况。（当前线程在删除缓存并且在写数据库前其他线程查询了就的数据）

2.情况 2：先修改数据，再删除缓存。当前缓存中方没有数据，线程 1 做查询操作，准备将旧数据缓存前，线程 2 做了修改数据和删除缓存的操作(此时是没有缓存的)，最终缓存中就存储了旧的数据，出现脏读的情况。（当前线程要将旧的数据进行缓存的过程中进行了写数据库和删除缓存的操作）

为了防脏读的缓存被使用，所以在数据同步的需要两次删除缓存的操作。

问：延迟双删是没法保证强一致性的，有什么强一致性的方法吗？
方法一：使用分布式锁，每次只有一个线程进行操作，效率比较低下。

方法二：因为缓存中的数据大多是读多写少，所以我们可以使用在读取数据的时候使用共享锁， 多个线程同时可以读取缓存但是其他线程不可以写，在修改数据的时候使用排他锁，

会阻塞其他线程的读写操作。

排他锁和共享锁可以使用 redisson 实现，通过 redissonClient 获取对应的读写锁。

1.读锁也就是共享锁。

2.写锁也就是排他锁。

必须保证读写锁的名字相同。

问：那有了解过最终一致性的方法吗?

在我们支付模块中，每次支付都需要获取支付模板也就是支付宝支付和微信支付模板，这些数据我们基本上不会修改，会将其缓存起来，在修改 mysql 中模板的数据的时候，通过 rabbitmq 发送消息，异步修改缓存的数据，达到最终一致的效果。

为什么 mq 可以实现最终一致的效果呢?

mq 中的消息都是按照顺序进行消费的，消息的消费是和事务绑定的，如果事务进行了回滚操作则被消费的消息也会被重新放回队列的原先位置中，并且接收到消息的服务会异步进行处理。

当然我们不仅仅可以使用 rabbitmq 实现最终一致的效果，我们还可以使用 canal 实现最终一致的效果。canal 主要是通过 mysql 的主从同步实现的。通过监听 bin Log 日志的方式来修改缓存中的信息，达到最终一致的效果。(binLog 日志主要是储存 DDL(数据定义语句)和 DML(数据操纵语句))

问：redis 做为缓存，数据的持久化是怎么做的呢?
在持久化上用 RDB 和 AOF 两种方式。

问:说说你对 RDB 和 AOF 的理解吧。

1.RDB：通过对数据做快照的方式做持久化，将快照存放到磁盘上，后续需要恢复数据的时间就使用该快照进行恢复。

RDB 的执行原理：在主进程会有一个页表文件(用于映射数据在物理内存上的数据)，通过复制该页表到子进程中，子进程通过页表找到数据并做快照。

当是如果在修改数据的过程中做 RDB 就会出现脏读的情况，RDB 通过设置数据为只读，在修改数据的时候复制一份相同的数据进行修改如何修改页表的映射，最终解决脏读的问题。

2.AOF：在 redis 做操写的指令的时候，会将这些指令都存储到对应的 AOF 文件中，在后续数据需要恢复的时候就执行 AOF 文件中的所有指令。

AOF 执行的原理：就是每次去记录操作写的指令到 AOF 文件中。

问：RDB 和 AOF 有什么区别吗?

1.RDB 是对整个内存做快照，而 AOF 是记录 redis 每一条执行的语句。

2.RDB 的在两次备份间可能会出现数据丢失的情况(redis 宕机)完整性低， 而 AOF 的完整性比较高，其取决去刷盘的策略。

3.RDB 的文件比较小，而 AOF 会记录每日一条指令，所以 AOF 的文件比较大。

4.RDB 的数据恢复比较快，因为 AOF 文件比较大，需要一条条的执行指令，恢复速度慢。

5.RDB 系统占用高，需要大量的 CPU 和内存的消耗，而 AOF 系统占用比较小，只要是文件的 IO 操作，但是在 AOF 文件重写的时候需要大量的 CPU 和内存的消耗。



问：RDB 和 AOF 谁的恢复速度更快，我们在平时要怎么选择呢?

RDB 的快照文件本质是二进制文件，其体积比较小，而 AOF 文件需要保存 redis 中的写操作指令，在体积上比较大，所以在恢复速度上 RDb 比较快，但是 RDB 存在数据丢失的风险，在我的项目中只要主要还是使用 AOF，像支付方式模板这种比较重要的数据，我们应当降低其丢失的风险，在刷盘策略上使用每秒进行一次刷盘，也就是每秒批量写入一次。

问：redis 的 key 过期后，会立刻做删除操作吗?
redis 有两种数据过期策略，分别是惰性删除和定期删除。

惰性删除：我们会为每个 key 设置一个过期时间，在每次获取数据的时候会先去判断该 key 是否过期，如果过期直接删除 key，如果没有过期则直接返回，也就是说在没有使用数据时不会主动删除。（优点：对 CPU 友好，只在查询时才做过期判断。缺点：内存消耗大。）

定期删除：定期去判断一定数量的 key 是否过期，如果过期则直接进行删除操作，定期删除又分为 SLOW 模式和 FAST 模式。（优点：内存消耗小。缺点：定期查询 key 需要消耗大量时间。）

SLOW 模式：默认的频率为 10hz（一秒内进行十次），每次不大于 25ms，我们可以通过配置文件中的 hz 修改频率。

FAST 模式：执行频率是不固定的，两次删除的间隔不小于 2ms，每次耗时小于 1ms。

redis 默认采用的数据过期删除策略：惰性删除+定期删除配合使用。

问：假如缓存过多，内存有限，内存满了怎么办呢?
在 redis 中提供了八种数据淘汰的策略，那默认的处理就是 noeviction，就是直接报错。我们可以提供配置文件修改对应的淘汰策略。策略中有两种重要的思想，分别是 LRU 和 LFU。

LRU:就是将当前使用间距最长的数据进行淘汰。（对应两个）

LFU:就是将当前使用频率最少的数据进行淘汰。（对应两个）

随机淘汰策略。

八个策略的不同就在于 key 的类型(全部 key 和设置过期时间的 key)和使用的思想不同(LRU 和 LFU)。

在我的项目中主要使用的淘汰策略就是: allkey-lru,淘汰掉当前使用最少的数据。

问：数据库中有 100w 条数据，redis 只能存储 20w 数据，如何保证 redis 中的数据是热点数据呢?

使用 allkey-lru 策略，保证经常使用的数据不被淘汰。

问：redis 缓存的空间用完了会怎么样?

如果是默认情况下的话，会直接报错，因为默认的淘汰策略就是 noeviction，如果设置了其他的策略则会对数据进行淘汰。

问：能介绍一下 redis 的主从复制和主从复制的流程吗?
单个 redis 节点的并发能力是有限的，所以为了提高并发能力，我们需要搭建 redis 集群，就比如：主从复制。

主从复制的流程

主从复制主要分为：全量同步和增量同步。

全量同步：在 salve 请求数据同步的时候会携带 application Id 和 offset，如果 master 判断出 applid 和自己的不一样，就认为 slave 是第一次进行同步，所以会进行全量同步。 master 会执行 bgsave 生成 RDB 文件给 slave，slave 进行同步，在此过程中 master 可能会进行新的指令，master 会将这些指令存储到日志文件，在加载 RDB 完后再将日志文件传给 slave 进行最终的同步，master 同步 applid 和 offset 给 slave。

增量同步： 在 master 判断出 slave 中的 applid 和自己一样就认为不是第一次同步，直接进行增量同步，从日志文件中获取到 offset 的位置，将 offset 之后的数据发送给 slave，进行数据的同步。

问：那主节点宕机了又该怎么办呢？
为了提高 redis 集群的高可用，我们可以使用哨兵模式，解决主节点宕机的问题。

问：那说说你对哨兵模式的理解吧

哨兵模式：通过 sentinel 的心跳机制去监测 master 的状态，当然为了保证高可用，我们也需要对 sentinel 搭建集群。sentinel 每隔 1 秒就会向集群的节点发送 ping 指令，当 master 失效后会就选择出新的 master。

在 sentinel 中有两种概念：主观下线和客观下线。

主观下线：当有一个 sentinel 发现 redis 节点没有返回响应就认为其为主观下线。

客观下线：当有一半以上的 sentinel 节点发现 redis 节点没有返回响应就认为其为客观下线。

当 master 发生客观下线就会筛选 slave 作为新的 master。

筛洗新 master 的优先级为下：

1.判断 master 与 slave 断开的时长，如果时长超过指指定值则直接排除。

2.判断 slave 的权重，如果权重越小优先级就越高。

3.如果权重相同的话，就比较 slave 的 offst 值也就是偏移量，如果 offset 越大优先级就越高。

4.判断 slave 运行 id 的大小，如果运行 id 越小则优先级越高。

问：哨兵模式可能会出现脑裂的情况，有了解过吗？

由于网络不稳定的因素，多个 sentinel 都没有 ping 到 master，此时 master 是没有宕机的，而哨兵模式就会选择出新的 master，就出现两个 master 的情况，而客户还是对旧的 master 进行数据的操作，在网络稳定后，旧的 master 就会变成新 master 的 slave，最终导致数据操作的丢失。从而形成脑裂的现象。

解决脑裂的方案：

1.在 redis 的配置文件中设置最少的 slave 个数（对应的配置项:min-replicas-to-write）。当出现脑裂的情况时，旧的 master 会监测到没有 slave，就不会做数据的操作直接返回错误信息，防止数据操作的丢失。

2.在 resdis 的配置文件中设置最大数据同步的延迟时间。（对应的配置项:min-replicas-max-lag）。当出现脑裂的情况时。旧的 master 在做数据的同步时一直找不到 slave，当时间超过最大的延迟时间就会直接返回错误信息，防止数据操作的的丢失。

问：你们使用 redis 是单点还是集群？

我们的 redis 使用主从模式（一主一从）+ 哨兵模式。当然在容量不够的时候，会为不同的服务配置独立的 redis 主从节点。

问：redis 分片集群有什么作用？
1,存在多个 master,这些 master 存储不同的数据，多个 master 可以解决并发写的问题。

2.每一个 master 都可以有多个 slave，解决并发读的问题。

3.在分片集群中，不再通过 sentinel 监测 master 的健康状态，而是通过 master 之间 ping 状态，判断各个 master 的健康状态，最终达到哨兵模式的效果。

4.客户端在访问对应的数据时会路由到对应的节点上。

问:redis 分集群中的数据是如何存储和读取的呢？
redis 分片集群采用的是哈希槽的结构实现的，哈希槽总共有 16384 个。master 节点等量的获取哈希槽范围用于存储数据。

在存储数据的时候，通过有效部分取哈希槽总数的模计算出哈希值（这里的有效部分指 key 前大括号中的有效值（就比如: set {}key value），如果没有大括号 key 就是有效部分），找到 master 后做写操作。

在读数据的时候，通过计算出的哈希值确定存储数据的 master 位置，从该 master 对应的 slave 中读取数据，到达读写分离的效果。

问：redis 是单线程的，为什么会那么快呢?
1.完全基于内存的，是 C 语言编写的。

2.采用单线程，避免了不必要的上下文切换可竞争条件，多线程还需要考虑线程问题。

3.采用多路 IO 复用模型，非阻塞 IO 模型。

问：说说对阻塞 IO 和非阻塞 IO 的理解吧。

我们先需要知道内存的使用情况为：用户空间和内核空间。

用户空间：只能执行受限制的指令，不能直接调用系统资源，需要通过内核提供的接口来调用系统资源。

内核空间：可以执行特权指令，可以直接调用系统资源。

阻塞 IO:在用户线程要获取内核中获取数据，而此时内核中没有数据，用户线程就会等待从而导致用户线程阻塞，当内核中有数据后，数据需要重内核缓冲区复制到用户缓冲区，在这个过程中用户线程也需要等待从而导致线程堵塞。在这两个阶段中用户线程都是堵塞的，这就是堵塞 IO。

非堵塞 IO: 用户线程要从内核中获取数据，但内核中没有数据，此时内核直接返回错误信息给用户线程，反之线程堵塞，用户线程会循环的调用内核的方法直到内核中有数据，当内核中有数据后，用户线程就会等待内核缓冲区中的数据复制到用户缓冲区中，此时用户线程是阻塞的。在第一阶段是不阻塞的，而第二阶段是堵塞的，这就是非堵塞 IO。比阻塞 IO 优化没多少，而且忙等机制可能会导致 CPU 的空转，CUP 使用率暴增。

问：解释一下什么是多路 IO 复用模型？

单线程同时监听多个 Socket（操作客户端）的状态，某个 Socket 可读可写时得到通知，防止出现忙等的情况，提高 CPU 的利用率，可能同时存在多个可用 Socket，通过循环做读取数据的操作，多路 IO 复用主要通过 epoll 模式实现的，将已就绪的 socket 存到用户空间中，就不需要遍历判断 socket 是否就绪，从而提高性能。

问：有了解过 redis 的网络模型吗？

通过多路 IO 复用 + 事件处理器实现的。事件处理器主要是：连接应答处理器，命令回复处理器，命令请求处理器。在 redis6.0 之后，使用多线程来处理命令的回复和命令的请求从而

实现高效的网络请求，在命令执行的时候依旧是单线程（线程安全的）。

MySQL 八股文
问：Mysql 的存储引擎有理解过吗？
我比较了解就是 Innodb，myisam，Memory。

Innodb：现在的 mysql 默认存储引擎就是 innodb，主要就是因为它是唯一一个支持事务的存储引擎，支持表级锁和行级锁，其索引的底层结构使用的是 B+树，在数据，索引，表结构都存储到.idb 中。

Myisam：其不支持事务，仅支持表级锁，其索引的底层结构为 B+树，表结构存储到.sdi 中，索引存储到.myi，数据存储到.myd 中。

Memory:基础内存进行存储的，主要就是用 sdi 存储表结构。

问：如何定位慢查询？
在我们的项目中，在上线时使用 skywalking 来定位慢的查询，如果发现是某个 SQL 执行速度慢，我们就可以使用 skywalking 的追踪功能，来确定 SQL 语句。

而在测试环境中，我们使用 MySQL 提供的慢日志来确定慢查询的位置。mysql 是默认没有开启慢日志的，需要通过配置文件开启并设置快查询的最大时间，超过这个时间就认为其为慢查询，我们就可以在慢日志中找到慢查询的 sql,在我们的项目中设置最大的时间为 2 秒。

#开启慢日志

slow_query_log=1

#设置快查询的最大时间

long_query_time=2

AI 助手

问：一个 SQL 语句执行很慢，应该如何分析呢？
我们可以借助 Mysql 提供的关键值 explain 来展示出某个 SQL 语句的状态。

在该状态中包含属性 key 和 key_lenSQL 中使用到的索引，如果提供我们的索引出现失效的情况就可以修改和 SQL 和添加索引。属性 type 表示 SQL 的性能，通常其值为 const，提供 type 判断是否存在全索引扫描或全盘扫描。属性 extra 表示建议属性，提供该属性判断是否出现回表的情况。

问：有了解过索引吗？

1.索引是帮助 Mysql 高效查询数据的数据结构。

2.索引提高检索效率，大大降低 IO 成本。

3.通过索引列对数据排序，大大降低了排序的成本。

问：B+树和 B 树的区别?
　B+树作将数据存放在叶子节点上，非叶子节点就可以组织更宽的结构，就会变的更矮更胖，提高查询速度。
　B+树中叶子节点使用双向链表进行存储的，并且按顺序进行存储的，不同于 B 树，查询范围数据的时候进行多次的从根节点进行查询，而 B+树在查询范围数据的时候只需要从根节点查询一次即可。且排序的性能更好。(B+树在做范围查询的时候只需要从根节点遍历一次，二 B 树则需要遍历对应数量的个数)
问：什么是聚簇索引和非聚簇索引?
聚簇索引: 索引结构和数据是存放在一起的, 也就是在 B+树的叶子节点中存放整行的数据。

非聚簇索引（也叫二级索引）：索引结构和数据不是存放在一起的，也就是在 B+树的叶子节点上存放对应的主键且是不唯一的。我们为字段添加索引通常就是二级索引。

问：知道什么是回表操作吗?

通过二级索引查到的主键再去聚簇索引中查询数据行的过程就是回表。而直接查询聚簇索引则不会出现回表的情况。

问：有了过覆盖索引吗?
查询数据通过索引进行查询，返回列都可以在索引的数据中找到（包含在其中），就是覆盖查询。

使用 id 主键进行查询就是覆盖索引查询，因为聚簇索引的数据中包含 id 主键，性能高。

在做查询的时候如果返回列吧全部存在于索引中就会回表查询，所以尽量避免使用 select *。
问：Mysql 超大分页查询怎么进行优化?

Mysql 做 limit 分页查询的时候，需要做排序，这个过程非常耗时。

优化方案：覆盖索引 + 子查询。先通过子查询出分页排序完后的 id 主键，因为是主键所以会直接进行覆盖索引查询。通过子查询的 id 关联表中的 id 查询出分页后的数据。

select * from table t

(select id from table limit 0 10 order by id) s

where t.id = s.id

AI 助手

问：索引的创建原则有哪些？
1.数据量大于十万且查询的频率表较高的表我们才会考虑创建索引。

2.如果一个表需要添加索引，我们应该选择作为查询字段，排序字段，分组字段的字段作为索引，且字段的区分度要高。

3.在添加索引的时候都使用复合索引来创建，尽量使用覆盖查询，降低回表的概率。

4.如果需要对长字符串添加索引我们可以使用前缀索引。

5.控制索引的数量，并不是越多越快，在增删改的时候我们也需要消耗时间来维护索引。

问：什么情况下索引会失效？
复合索引

1.在使用复合索引的时候不遵循最左前缀法则。在做条件查询的时候跳跃某一列字段导致索引失效。

2.在条件查询中的范围查询的右遍的列不能使用索引，使用也会失效。（如果三个都有效的话 key_len 应该为六百多，说明此时 address 字段失效）

3.不能在索引列进行运算操作，这会导致索引失效。

4.在条件查询的时候如果没有加单引号也会导致索引失效。（就比如：0 和'0'，会进行类型转换，导致索引失效）

5. 在模糊查询的时候，如果字符串中是以%开头的就会导致索引失效。（就比如: "%abc"）

在我遇到的随影失效问题就是没有遵循最左前缀原则，只要实在测试的时候通过 explain 查询 SQL 语句的执行状态来判断的。

问：谈谈你对 SQL 的优化经验？
在做 SQL 优化的时候主要从：建表时，使用索引时，sql 语句编写，主从复制，读写分离的方面进行考虑，当数据量过大的时候考虑使用分库分表。

问：创建表的时候你是怎么优化的？

我们主要遵循阿里的开发手册，就比如在使用整数类型的时候就考虑使用：tinyInt, Int, bigInt, 如果是逻辑字段就使用 tinyInt，在使用字符串是考虑使用：char，varchar，text。

问：那在使用索引的时候如何进行优化？

讲出索引失效的五种情况，再使用 SQL 的时候避免使用 select *，使用覆盖索引减少回表的操作。

问：你平时 SQL 语句是怎么优化的？

select 指明字段，不要使用 select * from 防止回表的操作。在使用聚合查询的时候尽量使用 union all 而不是 union，union 会多一次过滤，在效率上比较低。使用 inner join 而不使用 left join/right join，如果必须使用的，一定要以小表为驱动。

问：事务的特性是什么？可以详细说一下？
这里你可以取钱的例子来引导模式官。

原子性：在事务中的语句要么都成功要么都失败。

一致性：在事务中数据的总量不会变。

持久性： 提交和会滚的数据都会持久化到数据库。

隔离性：事务中间是是相互隔离的，是不会相互影响的。

问：并发事务带来了哪些问题？
并发事务可能会出现三种问题。

1.脏读：事务 1 读取到事务 2 未提交的数据。

2.不可重复读：事务 2 先后读取事务 1 中的某合个数据，两次的结果不一样。

3.幻读：一个事务在按条件查询数据时没有查到数据吗，但是插入操作时，又发现该数据已经存在。因为其他事务在这个过程中插入数据（选答）

怎么解决解决这些问题？

通过设置过隔离级别来解决。隔离级别包括：

1.读取未提交，无法解决并发事务带来的问题。

2.读取已提交，可以解决脏读。

3.可重复读，可以解决脏读，不可重复读。

4.串行化，事务只能一个一个执行，可以解决脏读，不可重复读，幻读，隔离级别最高，效率最低。

MySQL 默认的隔离级别是什么?

Mysql 默认使用的隔离级别是：可重复读。

问：undo log 和 rado log 有什么区别?
redo log：用于记录数据页的物理变化，当服务宕机的时候进行数据同步操作。保证了事务的持久性。

undo log: 记录逻辑日志, 就比如: 当做插入操作时会在日志中记录逆向的操作也即是删除, 在事务回滚的时候会执行逻辑日志中的指令。保证了事务的持久性和原子性。



问：隔离级别是怎么实现的?
排他锁+MVCC 实现的。

问：说说你对 MVCC 的理解吧?

多版本并发控制。维护一个数据的多个版本，使得读写操作没有冲突。

mvcc 主要有三个重点:

1.隐藏字段：trx_id(事务 id)：记录当前事务的 id，其为自增的。 roll-pointer：指向上一个版本的事务记录地址。

2.undo log：回滚日志，存储老版本的数据，版本链：多个同时修改某条记录，产生多版本的数据，通过 rool-pointer 指针形成链表。

3.readview：解决一个事务查询选择版本的问题。

根据 readView 的匹配规则和当前事务 id 找到对应的版本信息。(问规则时答：1.判断是事务 id 是否为当前事务的 id。2.是否是活跃事务 id。3.判断事务是否是在 readview 创建后开启的，也就是事务 id 大于当前事务。4.判断事务中的数据是否已提交，事务 id 小于最小的事

务 id。）

不同的隔离级别快照读是不一样的，最终的访问结果也是不一样的。（问时答：当前读：读取的是最新的数据并且会加锁。快照读：读取的是记录数据的可见版本，不会加锁。）

读已提交：在每次快照读的时候都会生成 readview。

可重复读：在有在第一次快照读的时候才会生成 readview，后续的快照读都是使用该 readview 的复制，保证数据的一致性。

问：隔离级别中可重复读有什么缺点呢？
1.无法解决幻读的问题，当我们事务去读取一定范围的数据，且在该过程中其他的事务修改了该范围的数据，此时两次读取就会出现查询结果不一致的情况，最终导致幻读。

2.无法读取到最新的数据，因为可重复读在每次读取的时候都使用同一个 readView，且 readView 并非当前最新的数据，最终导致无法读取到新的数据。

问：MySQL 的主从同步有了解过吗？
Mysql 主从同步的核心就是 bin log(二进制日志)，这个日志中主要记录 DDL(表的操作)，DML(表中数据的操作)。

1.master 中事务提交数据后，会将修改的数据保存到 bin log 中。

2.slave 有个 iothread 线程会监控的 bin log 的变化，并将变化写入 relay log 中。

3.slave 有个 SQLthread 线程会监控 relay log，将改变的数据写入 slave 中。

问：你在项目中有使用过分库分表吗？
在物流项目中的订单服务的数据非常庞大，请求数多且业务累计大。差不多单表的数据有 100w 条，这时我们就使用分库分表。

分库分表有四种策略：

1.水平分库：通过将一个库中的数据拆分到多个库中，解决海量数据存储和高并发的问题。主要通过 sharing-sphere 和 mtycat 实现。

2.水平分表：解决单表存储和性能的问题。

3.垂直分库：根据业务来拆分库中的表，在高并发的情况下提高磁盘 IO 和网络连接数。每个微服务都有自己的表。

4.垂直分表：冷热数据分离，多表不会相互影响。就比如：表中字段为 id,name，des，将

id,name 和 des 分离，id 和 name 都是热数据而 des 为冷数据，访问频率较低。

框架八股文
问：Spring 中的设计模式有哪些?
我们目前主要了解的就是工厂模式，代理模式，单例模式，策略模式，责任链模式。

工厂模式：常见的工厂模式主要就是 BeanFactory（延迟注入）和 ApplicationContext（完全注入），我们无需知道类如何创建，直接从工厂中获取即可。

单例模式：ioc 默认情况下就是一个单例模式，每次获取相同恶的类的时候，获取的对象是同一个。不会就行多次的创建。单例模式还分为饿汉式（主动创建单例）及懒汉式（需要时再创建单例）。

代理模式 ： aop 就是采用代理模式来实现的。当要代理的对象实现接口的时候，我们就会使用 JDK Proxy 来生成代理对象。如果代理的对象没有实现接口的话，我们就回使用 CGLIB 生成一个被代理对象的子类作为代理对象。

策略模式: 在我们编写项目的时候就有使用到策略模式, 因为我要控制分布式锁的失败策略，我们会在枚举类编写一个抽象方法，编写多个内部方法去重新该方法，不同的内部方法就是不同的策略。
责任链模式：stram 流就是使用该模式的，按照对应的顺序去执行方法，并向下传递对象。起到解耦合的作用。

问：spring 框架的单例 bean 是线程安全的吗?
在 spring 框架中有个注解叫@Scope 可以设置 bean 的状态，默认就是 singleton 也就是单例。

bean 进行注入的时都是无状态的，其不会被修改的。所以没有线程安全的问题。但是如果 bean 中有成员变量时就可能会有线程安全的问题, 因为该成员变量可能会被多个线程修改，为了解决这个问题我们可以加锁或将 bean 设置为多例。(@Scope 设置为 prototype)

问：什么是 AOP?
面向切面编程，将那些于业务无关的复用性比较高的代码快抽取出来，较低代码的耦合度。

问：在你的项目中有使用过 AOP 吗?

在我的云盘项目中就使用到 AOP，在记录日志的时候，我创建有个自定义注解，aop 的切面就是这个注解，使用环绕通知在方法中，我们通过传入的参数（joinPoint）获取对应的类和方法，从而获取前端传来的参数和其他主要信息，实现记录日志的效果。

问：Spring 中的事务是怎么实现的?

本质就是通过 AOP 实现的, 通过环绕通知对应方法进行前后拦截, 在方法执行前开启事务，在执行后提交事务，会对此过程进行 try/catch，如果报错直接回滚。（就是@transactional）

问：spring 中事务失效场景有哪些？
1.在出现异常后，方法中 try/catch 了该异常并且没有主动抛出异常，这时候就会导致事务失效。解决方法：在方法 try/catch 异常后手动的抛出异常。(就会导致事务不知道出现异常了)

2.抛出检查异常时会导致事务失效,spring 中的事务只会对 runtime 异常进行回滚。就比如:Not found Exception 就是检查异常。解决方法：在@transactional 中设置属性 rollbackFor = Exception.class。使得事务会对所有的异常进行回滚。

3.非 public 方法会导致事务失效。解决方法：将方法的作用域该为 public。

4.在非事务方法中调用了事务的方法，此时就会导致事务失效。(就比如某给没有加事务注解的方法调用了加了事务注解的方法)

5.回滚异常类型不匹配。(我们可能会设置需要进行回滚的异常，就是 rollback 的值，如果抛出的异常类型不匹配就会导致事务失效)

6.事务的传播行为错误。(就比如在事务方法中调用了其他的事务方法，初始化如果其他的时候方法设置开启新事务的话，在其事务成功后，就不会参与外部事务的回滚操作)

问：事务的传播性有哪些？
主要就是三个：Propagation_Required，Propagation_Required_new，Propagation_nested。

Propagation_Required：如果 B 为 A 的子方法，并且都为该传播类型，那么它们都会在同一个事务中。如果主方法中没有开启事务的话就会开启新事务。
Propagation_Required_new：如果 B 为 A 子方法，B 会创建一个独立的事务，B 不会受 A 事务的影响。
Propagation_nested：如果 B 为 A 子方法，B 会创建一个事务内嵌到 A 的事务中，如果 A 中没有事务的话，就单独开启一个事务。
问：@Transactional 的原理有了解过嘛？
事务注解是基于 AOP 来实现的，也就是在执行方法前开启事务，如果 try/cath 捕获到异常的话就会进行回滚，在方法执行完成后就会进行提交。那 AOP 实现涉及到动态代理的流程。如果被代理目标对象实现了接口的话，就会使用 JDK Proxy 生成代理对象。如果代理对象没有实现接口的话，就会使用 CGLIB Proxy 生成被代理对象的子类作为代理对象。

问：说说 CGLIB 的执行流程？
问：spring 中 bean 的生命周期有了解过吗？
结构图：

生命周期的流程为下：

1.通过 BeanDefinition 获取 bean 的定义信息。

2.通过构造函数创建 bean，可以将当前的 bean 理解为一个空壳。

3.进行依赖注入，对 bean 中的属性进行赋值。

4.处理 Aware 接口，也就是一些以 Aware 结尾的接口，就比如：beanNameAware，beanFactoryAware，applicationContextAware。如果实现了个接口的话，我们需要重写一些方法。

5 加载 BeanPostProcessor 对象，并执行处理器的前置初始化方法(PostProcessorAfterInitiazilation 方法)。

6.执行初始化方法，包括 IntializingBean 和自定义的初始化方法。

7. 加载 BeanPostProcessor 对象，并执行处理器的后置初始化方法(PostProcessorBeforeInitiazilation 方法)。在此后置处理器中我们可以通过 aop 对原始的 bean 做增强也就是进行代理，代理就包括 JVM 代理和 CGLIB 代理。

8.将 bean 进行销毁。