Redis相关面试题

面试官：什么是缓存穿透？怎么解决？

候选人：

嗯~~，我想一下

缓存穿透是指查询一个一定不存在的数据，如果从存储层查不到数据则不写入缓存，这将导致这个不存在的数据每次请求都要到 DB 去查询，可能导致 DB 挂掉。这种情况大概率是遭到了攻击。

解决方案的话，我们通常都会用布隆过滤器来解决它

面试官：好的，你能介绍一下布隆过滤器吗？

候选人：

嗯，是这样~

布隆过滤器主要是用于检索一个元素是否在一个集合中。我们当时使用的是redisson实现的布隆过滤器。

它的底层主要是先去初始化一个比较大数组，里面存放的二进制0或1。在一开始都是0，当一个key来了之后经过3次hash计算，模于数组长度找到数据的下标然后把数组中原来的0改为1，这样的话，三个数组的位置就能标明一个key的存在。查找的过程也是一样的。

当然是有缺点的，布隆过滤器有可能会产生一定的误判，我们一般可以设置这个误判率，大概不会超过5%，其实这个误判是必然存在的，要不就得增加数组的长度，其实已经算是很划分了，5%以内的误判率一般的项目也能接受，不至于高并发下压倒数据库。

面试官：什么是缓存击穿？怎么解决？

候选人：

嗯！！

缓存击穿的意思是对于设置了过期时间的key，缓存在某个时间点过期的时候，恰好这时间点对这个Key有大量的并发请求过来，这些请求发现缓存过期一般都会从后端 DB 加载数据并回设到缓存，这个时候大并发的请求可能会瞬间把 DB 压垮。

解决方案有两种方式：

第一可以使用互斥锁：当缓存失效时，不立即去load db，先使用如 Redis 的 setnx 去设置一个互斥锁，当操作成功返回时再进行 load db的操作并回设缓存，否则重试get缓存的方法

第二种方案可以设置当前key逻辑过期，大概是思路如下：

①：在设置key的时候，设置一个过期时间字段一块存入缓存中，不给当前key设置过期时间

②：当查询的时候，从redis取出数据后判断时间是否过期

③：如果过期则开通另外一个线程进行数据同步，当前线程正常返回数据，这个数据不是最新

当然两种方案各有利弊：

如果选择数据的强一致性，建议使用分布式锁的方案，性能上可能没那么高，锁需要等，也有可能产生死锁的问题

如果选择key的逻辑删除，则优先考虑的高可用性，性能比较高，但是数据同步这块做不到强一致。

面试官：什么是缓存雪崩？怎么解决？

候选人：

嗯！！

缓存雪崩意思是设置缓存时采用了相同的过期时间，导致缓存在某一时刻同时失效，请求全部转发到DB，DB 瞬时压力过重雪崩。与缓存击穿的区别：雪崩是很多key，击穿是某一个key缓存。

解决方案主要是可以将缓存失效时间分散开，比如可以在原有的失效时间基础上增加一个随机值，比如1-5分钟随机，这样每一个缓存的过期时间的重复率就会降低，就很难引发集体失效的事件。

面试官：redis做为缓存，mysql的数据如何与redis进行同步呢？（双写一致性）

候选人：嗯！就说我最近做的这个项目，里面有xxxx（根据自己的简历上写）的功能，需要让数据库与redis高度保持一致，因为要求时效性比较高，我们当时采用的读写锁保证的强一致性。

我们采用的是redisson实现的读写锁，在读的时候添加共享锁，可以保证读读不互斥，读写互斥。当我们更新数据的时候，添加排他锁，它是读写，读读都互斥，这样就能保证在写数据的同时是不会让其他线程读数据的，避免了脏数据。这里面需要注意的是读方法和写方法上需要使用同一把锁才行。

面试官：那这个排他锁是如何保证读写、读读互斥的呢？

候选人：其实排他锁底层使用也是setnx，保证了同时只能有一个线程操作锁住的方法

面试官：你听说过延时双删吗？为什么不用它呢？

候选人：延迟双删，如果是写操作，我们先把缓存中的数据删除，然后更新数据库，最后再延时删除缓存中的数据，其中这个延时多久不太好确定，在延时的过程中可能会出现脏数据，并不能保证强一致性，所以没有采用它。

面试官：redis做为缓存，mysql的数据如何与redis进行同步呢？（双写一致性）

候选人：嗯！就说我最近做的这个项目，里面有xxxx（根据自己的简历上写）的功能，数据同步可以有一定的延时（符合大部分业务）

我们当时采用的阿里的canal组件实现数据同步：不需要更改业务代码，部署一个canal服务。canal服务把自己伪装成mysql的一个从节点，当mysql数据更新以后，canal会读取binlog数据，然后在通过canal的客户端获取到数据，更新缓存即可。

面试官：redis做为缓存，数据的持久化是怎么做的？

候选人：在Redis中提供了两种数据持久化的方式：1、RDB 2、AOF

面试官：这两种持久化方式有什么区别呢？

候选人：RDB是一个快照文件，它是把redis内存存储的数据写到磁盘上，当redis实例宕机恢复数据的时候，方便从RDB的快照文件中恢复数据。

AOF的含义是追加文件，当redis操作写命令的时候，都会存储这个文件中，当redis实例宕机恢复数据的时候，会从这个文件中再次执行一遍命令来恢复数据

面试官：这两种方式，哪种恢复的比较快呢？

候选人：RDB因为是二进制文件，在保存的时候体积也是比较小的，它恢复的比较快，但是它有可能会丢数据，我们通常在项目中也会使用AOF来恢复数据，虽然AOF恢复的速度慢一些，但是它丢数据的风险要小很多，在AOF文件中可以设置刷盘策略，我们当时设置的就是每秒批量写入一次命令

面试官：Redis的数据过期策略有哪些？

候选人：

嗯~，在redis中提供了两种数据过期删除策略

第一种是惰性删除，在设置该key过期时间后，我们不去管它，当需要该key时，我们在检查其是否过期，如果过期，我们就删掉它，反之返回该key。

第二种是 定期删除，就是说每隔一段时间，我们就对一些key进行检查，删除里面过期的key

定期清理的两种模式：

- SLOW模式是定时任务，执行频率默认为10hz，每次不超过25ms，以通过修改配置文件redis.conf 的 **hz** 选项来调整这个次数
- FAST模式执行频率不固定，每次事件循环会尝试执行，但两次间隔不低于2ms，每次耗时不超过1ms

Redis的过期删除策略：**惰性删除 + 定期删除**两种策略进行配合使用。

面试官：Redis的数据淘汰策略有哪些？

候选人：

嗯，这个在redis中提供了很多种，默认是noeviction，不删除任何数据，内部不足直接报错

是可以在redis的配置文件中进行设置的，里面有两个非常重要的概念，一个是LRU，另外一个是LFU

LRU的意思就是最少最近使用，用当前时间减去最后一次访问时间，这个值越大则淘汰优先级越高。

LFU的意思是最少频率使用。会统计每个key的访问频率，值越小淘汰优先级越高

我们在项目设置的allkeys-lru，挑选最近最少使用的数据淘汰，把一些经常访问的key留在redis中

面试官：数据库有1000万数据 ,Redis只能缓存20w数据, 如何保证Redis中的数据都是热点数据？

候选人：

嗯，我想一下~~

可以使用 allkeys-lru （挑选最近最少使用的数据淘汰）淘汰策略，那留下来的都是经常访问的热点数据

面试官：Redis的内存用完了会发生什么？

候选人：

嗯~，这个要看redis的数据淘汰策略是什么，如果是默认的配置，redis内存用完以后则直接报错。我们当时设置的 allkeys-lru 策略。把最近最常访问的数据留在缓存中。

面试官：Redis分布式锁如何实现？

候选人：嗯，在redis中提供了一个命令setnx(SET if not exists)

由于redis的单线程的，用了命令之后，只能有一个客户端对某一个key设置值，在没有过期或删除key的时候是其他客户端是不能设置这个key的

面试官：好的，那你如何控制Redis实现分布式锁有效时长呢？

候选人：嗯，的确，redis的setnx指令不好控制这个问题，我们当时采用的redis的一个框架redisson实现的。

在redisson中需要手动加锁，并且可以控制锁的失效时间和等待时间，当锁住的一个业务还没有执行完成的时候，在redisson中引入了一个看门狗机制，就是说每隔一段时间就检查当前业务是否还持有锁，如果持有就增加加锁的持有时间，当业务执行完成之后需要使用释放锁就可以了

还有一个好处就是，在高并发下，一个业务有可能会执行很快，先客户1持有锁的时候，客户2来了以后并不会马上拒绝，它会自旋不断尝试获取锁，如果客户1释放之后，客户2就可以马上持有锁，性能也得到了提升。

面试官：好的，redisson实现的分布式锁是可重入的吗？

候选人：嗯，是可以重入的。这样做是为了避免死锁的产生。这个重入其实在内部就是判断是否是当前线程持有的锁，如果是当前线程持有的锁就会计数，如果释放锁就会在计算上减一。在存储数据的时候采用的hash结构，大key可以按照自己的业务进行定制，其中小key是当前线程的唯一标识，value是当前线程重入的次数

面试官：redisson实现的分布式锁能解决主从一致性的问题吗

候选人：这个是不能的，比如，当线程1加锁成功后，master节点数据会异步复制到slave节点，此时当前持有Redis锁的master节点宕机，slave节点被提升为新的master节点，假如现在来了一个线程2，再次加锁，会在新的master节点上加锁成功，这个时候就会出现两个节点同时持有一把锁的问题。

我们可以利用redisson提供的红锁来解决这个问题，它的主要作用是，不能只在一个redis实例上创建锁，应该是在多个redis实例上创建锁，并且要求在大多数redis节点上都成功创建锁，红锁中要求是redis的节点数量要过半。这样就能避免线程1加锁成功后master节点宕机导致线程2成功加锁到新的master节点上的问题了。

但是，如果使用了红锁，因为需要同时在多个节点上都添加锁，性能就变的很低了，并且运维维护成本也非常高，所以，我们一般在项目中也不会直接使用红锁，并且官方也暂时废弃了这个红锁

面试官：好的，如果业务非要保证数据的强一致性，这个该怎么解决呢？

候选人：嗯~，redis本身就是支持高可用的，做到强一致性，就非常影响性能，所以，如果有强一致性要求高的业务，建议使用zookeeper实现的分布式锁，它是可以保证强一致性的。

面试官：Redis集群有哪些方案, 知道嘛？

候选人：嗯~~，在Redis中提供的集群方案总共有三种：主从复制、哨兵模式、Redis分片集群

面试官：那你来介绍一下主从同步

候选人：嗯，是这样的，单节点Redis的并发能力是有上限的，要进一步提高Redis的并发能力，可以搭建主从集群，实现读写分离。一般都是一主多从，主节点负责写数据，从节点负责读数据，主节点写入数据之后，需要把数据同步到从节点中

面试官：能说一下，主从同步数据的流程

候选人：嗯~~，好！主从同步分为了两个阶段，一个是全量同步，一个是增量同步

全量同步是指从节点第一次与主节点建立连接的时候使用全量同步，流程是这样的：

第一：从节点请求主节点同步数据，其中从节点会携带自己的replication id和offset偏移量。

第二：主节点判断是否是第一次请求，主要判断的依据就是，主节点与从节点是否是同一个replication id，如果不是，就说明是第一次同步，那主节点就会把自己的replication id和offset发送给从节点，让从节点与主节点的信息保持一致。

第三：在同时主节点会执行bgsave，生成rdb文件后，发送给从节点去执行，从节点先把自己的数据清空，然后执行主节点发送过来的rdb文件，这样就保持了一致

当然，如果在rdb生成执行期间，依然有请求到了主节点，而主节点会以命令的方式记录到缓冲区，缓冲区是一个日志文件，最后把这个日志文件发送给从节点，这样就能保证主节点与从节点完全一致了，后期再同步数据的时候，都是依赖于这个日志文件，这个就是全量同步

增量同步指的是，当从节点服务重启之后，数据就不一致了，所以这个时候，从节点会请求主节点同步数据，主节点还是判断不是第一次请求，不是第一次就获取从节点的offset值，然后主节点从命令日志中获取offset值之后的数据，发送给从节点进行数据同步

面试官：怎么保证Redis的高并发高可用

候选人：首先可以搭建主从集群，再加上使用redis中的哨兵模式，哨兵模式可以实现主从集群的自动故障恢复，里面就包含了对主从服务的监控、自动故障恢复、通知；如果master故障，Sentinel会将一个slave提升为master。当故障实例恢复后也以新的master为主；同时Sentinel也充当Redis客户端的服务发现来源，当集群发生故障转移时，会将最新信息推送给Redis的客户端，所以一般项目都会采用哨兵的模式来保证redis的高并发高可用

面试官：你们使用redis是单点还是集群，哪种集群

候选人：嗯！，我们当时使用的是主从（1主1从）加哨兵。一般单节点不超过10G内存，如果Redis内存不足则可以给不同服务分配独立的Redis主从节点。尽量不做分片集群。因为集群维护起来比较麻烦，并且集群之间的心跳检测和数据通信会消耗大量的网络带宽，也没有办法使用lua脚本和事务

面试官：redis集群脑裂，该怎么解决呢？

候选人：嗯！ 这个在项目很少见，不过脑裂的问题是这样的，我们现在用的是redis的哨兵模式集群的

有的时候由于网络等原因可能会出现脑裂的情况，就是说，由于redis master节点和redis salve节点和sentinel处于不同的网络分区，使得sentinel没有能够心跳感知到master，所以通过选举的方式提升了一个salve为master，这样就存在了两个master，就像大脑分裂了一样，这样会导致客户端还在old master那里写入数据，新节点无法同步数据，当网络恢复后，sentinel会将old master降为salve，这时再从新master同步数据，这会导致old master中的大量数据丢失。

关于解决的话，我记得在redis的配置中可以设置：第一可以设置最少的salve节点个数，比如设置至少要有一个从节点才能同步数据，第二个可以设置主从数据复制和同步的延迟时间，达不到要求就拒绝请求，就可以避免大量的数据丢失

面试官：redis的分片集群有什么作用

候选人：分片集群主要解决的是，海量数据存储的问题，集群中有多个master，每个master保存不同数据，并且还可以给每个master设置多个slave节点，就可以继续增大集群的高并发能力。同时每个master之间通过ping监测彼此健康状态，就类似于哨兵模式了。当客户端请求可以访问集群任意节点，最终都会被转发到正确节点

面试官：Redis分片集群中数据是怎么存储和读取的？

候选人：

嗯~，在redis集群中是这样的

Redis 集群引入了哈希槽的概念，有 16384 个哈希槽，集群中每个主节点绑定了一定范围的哈希槽范围， key通过 CRC16 校验后对 16384 取模来决定放置哪个槽，通过槽找到对应的节点进行存储。

取值的逻辑是一样的

面试官：Redis是单线程的，但是为什么还那么快？

候选人：

嗯，这个有几个原因吧~~~

1、完全基于内存的，C语言编写

2、采用单线程，避免不必要的上下文切换可竞争条件

3、使用多路I/O复用模型，非阻塞IO

例如：bgsave 和 bgrewriteaof 都是在后台执行操作，不影响主线程的正常使用，不会产生阻塞

面试官：能解释一下I/O多路复用模型？

候选人：嗯~~，I/O多路复用是指利用单个线程来同时监听多个Socket ，并在某个Socket可读、可写时得到通知，从而避免无效的等待，充分利用CPU资源。目前的I/O多路复用都是采用的epoll模式实现，它会在通知用户进程Socket就绪的同时，把已就绪的Socket写入用户空间，不需要挨个遍历Socket来判断是否就绪，提升了性能。

其中Redis的网络模型就是使用I/O多路复用结合事件的处理器来应对多个Socket请求，比如，提供了连接应答处理器、命令回复处理器，命令请求处理器；

在Redis6.0之后，为了提升更好的性能，在命令回复处理器使用了多线程来处理回复事件，在命令请求处理器中，将命令的转换使用了多线程，增加命令转换速度，在命令执行的时候，依然是单线程