Redis简介

原创 于 2025-03-31 22:58:22 发布 · 3.3k 阅读 · 26 · 38 · CC 4.0 BY-SA版权

文章标签： #redis #数据库 #缓存

一、什么是Redis ？

Redis是一种基于内存的数据库 ，对数据的读写操作都是在内存中完成的，因此读写速度非常快，常用于缓存、消息队列 、分布式锁等场景。

Redis提供了多种数据类型来支持不同的业务场景，比如String（字符串）、Hash（哈希）、List（列表）、Set（集合）、Zset（有序集合）等，并且对数据类型的操作都是原子性的，因为执行命令由单线程负责的，不存在并发竞争问题。

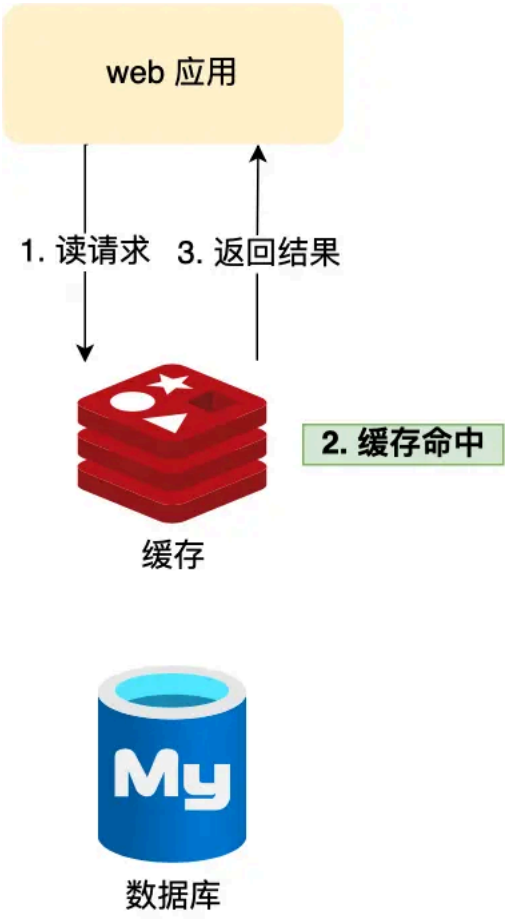
除此之外，Redis还支持事务、持久化、Lua脚本、多种集群方案（**主从复制模式**、哨兵模式、切片机群模式）、发布/订阅模式、内存淘汰机制、**过期删除机制**等等。

二、为什么用Redis作为Mysql 的缓存?

主要是因为Redis具备高性能和高并发两种特性。

1、Redis具备高性能

假如用户第一次访问Mysql中的某些数据，这个过程会比较慢，因为是从硬盘上读取的。将该用户访问的数据缓存在Redis中，这样下一次在访问这些数据的时候就可以直接从缓存中读取了，操作Redis缓存 就是直接操作内存，所以速度相当快。（读写操作）



如果Mysql中的对应数据改变之后，同步改变Redis缓存中相应的数据即可。

2、Redis具备高并发

单台设备的Redis的QPS（Query Per Second,每秒钟处理完请求的次数）是Mysql的10倍，Redis单机的QPS能轻松突破10w，而Mysql单机的QPS很难破1W。

所以，直接访问Redis能够承受的请求是远远大于直接访问Mysql的，所以我们可以考虑把数据库中的部分数据转移到缓存中去，这样用户的一部分请求会直接到缓存这里而不用经过数据库。

三、Redis线程模型

1、Redis是单线程吗?

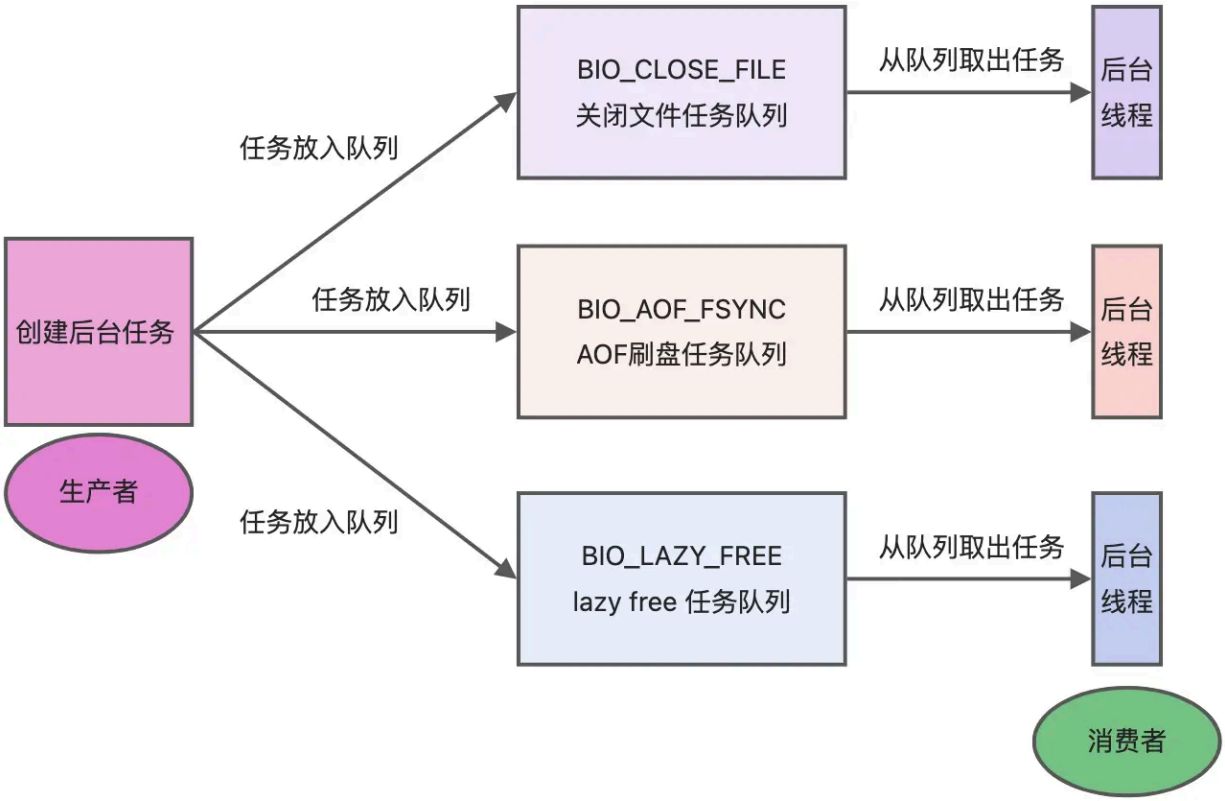
Redis单线程指的是接收客户端　请求->解析请求->进行数据读写等操作->发送数据给客户端这个过程是由一个线程（主线程）来完成，这也是我们常说Redis是单线程的原因。

但是，Redis并不是单线程的，Redis在启动的时候，是会启动后台线程的：

- Redis在2.6版本，会启动2个后台线程，分别处理关闭文件、AOF刷盘这两个任务；
- Redis在4.0版本之后，新增了一个后台线程，用来异步释放Redis内存，也就是lazyfree线程。例如执行unlink key/ flushdb async/ flushall async等命令，会把这些删除操作交给后台线程来执行，好处是不会导致主线程卡顿。因此，当我们要删除一个大key的时候，不要使用del命令删除，因为del是在主线程处理的，这样会导致Redis主线程卡顿，因此我们应该使用unlink命令来异步删除大key。

之所以Redis为关闭文件、AOF刷盘、释放内存这些任务创建单独的线程来处理，是因为这些任务的操作都是很耗时的，如果把这些任务都放在主线程来处理，那么Redis主线程就很容易发生阻塞，这样就无法处理后续的请求了。

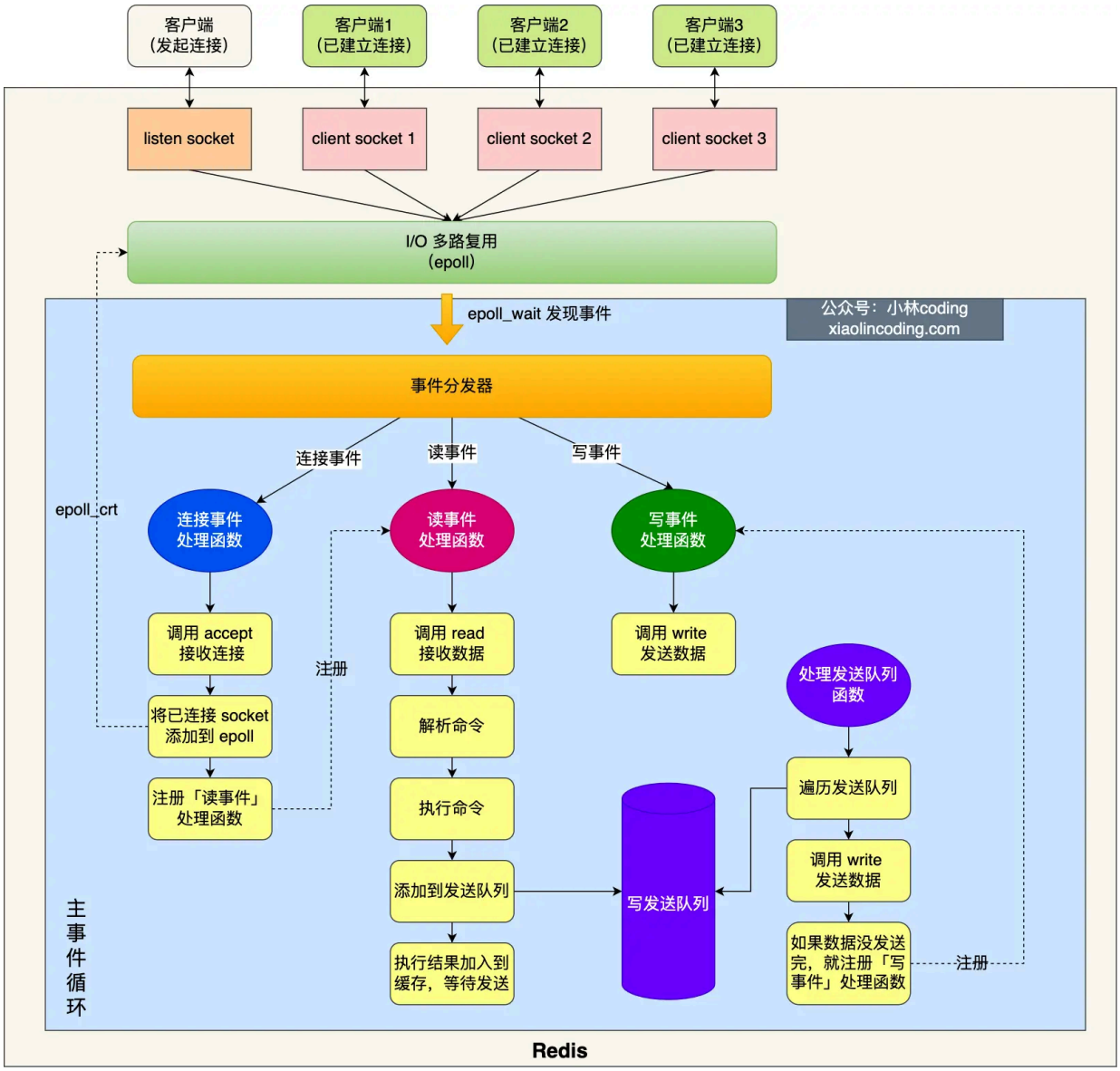
后台线程相当于一个消费者，生产者把耗时任务丢到任务队列中，消费者（BIO）不停轮询这个队列，拿出任务就去执行对应的方法即可。



关闭文件、AOF刷盘、释放内存这三个任务都有各自的任务队列：

- BIO_CLOSE_FILE，关闭文件任务队列：当队列有任务后，后台线程会调用close(fd)，将文件关闭；
- BIO_AOF_FSYNC，AOF刷盘任务队列：当AOF日志配置成everysec选项后，主线程会把AOF写日志操作封装成一个任务，也放到队列中。当发现队列有任务后，后台线程会调用fsync(fd)，将AOF文件刷盘；
- BIO_LAZY_FREE，lazy_free任务队列：当队列有任务后，后台线程会free(obj)释放对象/free(dict)删除数据库所有对象/free(skiplist)释放跳表对象。

2、Redis单线程模式是怎样的?



图中的蓝色部分是一个事件循环，是由主线程负责的，可以看到网络I/O和命令处理都是单线程。Redis初始化的时候，会做下面这几件事情：

- 首先，调用epoll_create()创建一个epoll对象和调用socket()创建一个服务端socket
- 然后，调用bind()绑定端口和调用listen()监听该socket
- 然后，将调用epoll_ctl()将listen socket加入到epoll，同时注册连接事件处理函数

初始化完后，主线程就进入到一个事件循环函数，主要会做以下事情：

- 首先，会调用处理发送队列函数，看是发送队列里是否有任务，如果有发送任务，则通过write函数将客户端发送缓冲区里的数据发送出去，如果这一轮数据没有发送完，就会注册写事件处理函数，等待epoll_wait发现可写后再处理。
- 接着，调用epoll_wait函数等待事件的到来

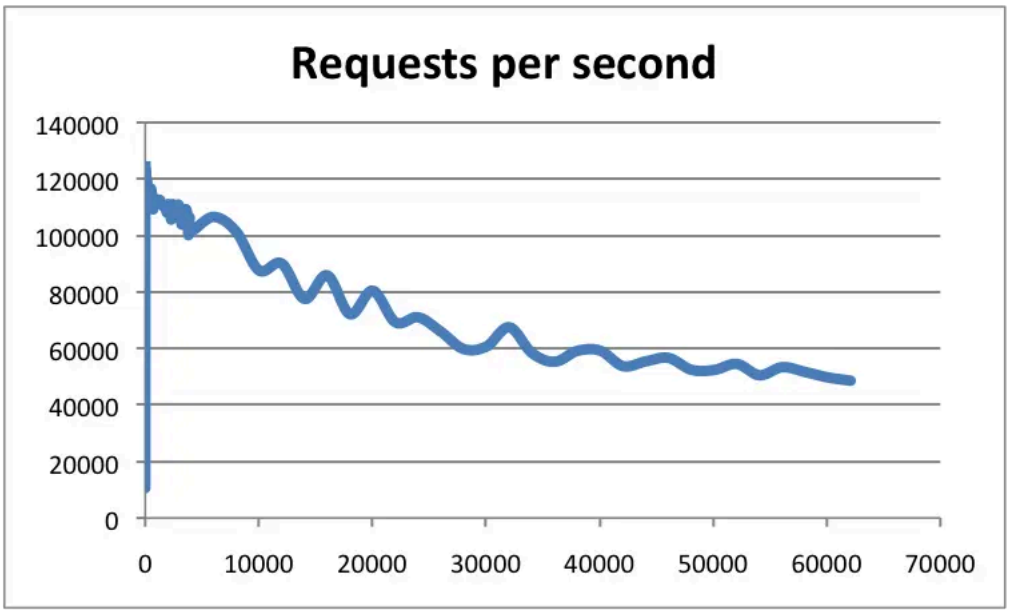
如果是连接事件到来，则会调用连接事件处理函数，该函数会做这些事情：调用accept获取已连接的socket->调用epoll_ctl将已连接的socket加入到epoll->注册读事件处理函数；

如果是读事件到来，则会调用读事件处理函数，该函数会做这些事情：调用read获取客户端发送的数据->解析命令->处理命令->将客户端对象添加到发送队列->将执行结果写到发送缓冲区等待发送；

如果是写事件到来，则会调用写事件处理函数，该函数会做这些事情：通过write函数将客户端发送缓冲区的数据发送出去，如果这一轮数据没有发送完，就会继续注册写事件处理函数，等待epoll_wait发现可写后在处理。

3、Redis采用单线程为什么还这么快?

官方使用基准测试的结果是，单线程的Redis吞吐量达到10W/每秒，如下图所示：



之所以Redis采用单线程（网络I/O和执行命令）那么快，有如下几个原因：

- Redis的大部分操作都在内存中完成，并且采用了高效的数据结构，因此Redis瓶颈可能是机器的内存或者网络带宽，而并非CPU，既然CPU不是瓶颈，那么自然就采用单线程的解决方案了；
- Redis采用单线程模型可以避免多线程之间的竞争，省去了多线程切换带来的时间和性能上的开销，而且也不会导致锁问题。
- Redis采用了I/O多路复用机制处理大量的客户端Socket请求，IO多路复用机制是指一个线程处理多个IO流。

四、RDB持久化

1、Redis如何实现数据不丢失?

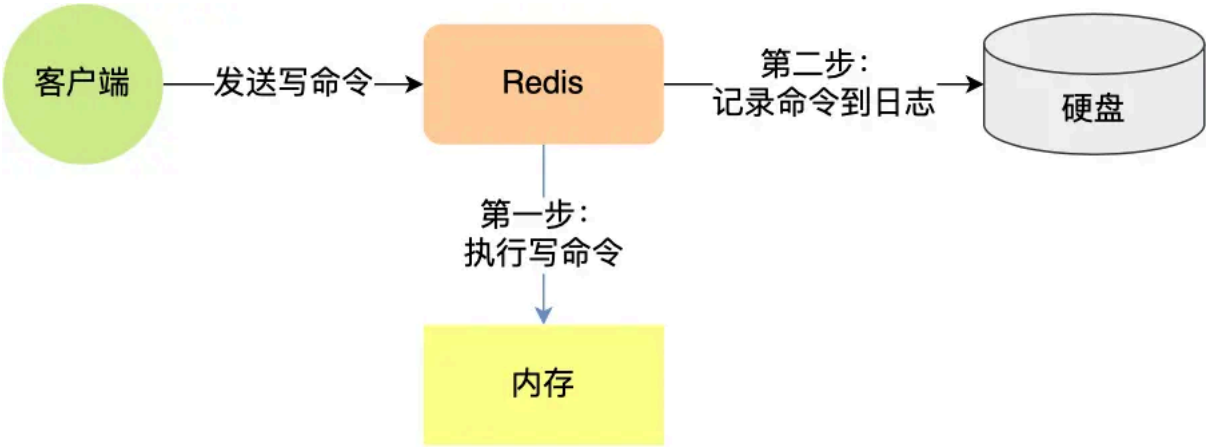
Redis的读写操作都是在内存中，所以Redis性能才会高，但是当Redis重启后，内存中的数据就会丢失，那为了保证内存中的数据不会丢失，Redis实现了数据持久化的机制，这个机制会把数据存储到磁盘，这样在Redis重启就能够从磁盘中恢复原有的数据。

Redis共有三种数据持久化的方式：

- AOF日志：每执行一条写操作命令，就把该命令以追加的方式写入到一个文件里；
- RDB快照：将某一时刻的内存数据，以二进制的方式写入磁盘；
- 混合持久化方式：Redis 4.0新增的方式，继承了AOF和RDB的优点。

2、AOF日志是如何实现的?

Redis在执行完一条写操作命令后，就会把该命令已追加的方式写入到一个文件里，然后Redis重启时，会读取该文件记录的命令，然后逐一执行命令的方式来进行数据恢复。



五、Redis集群

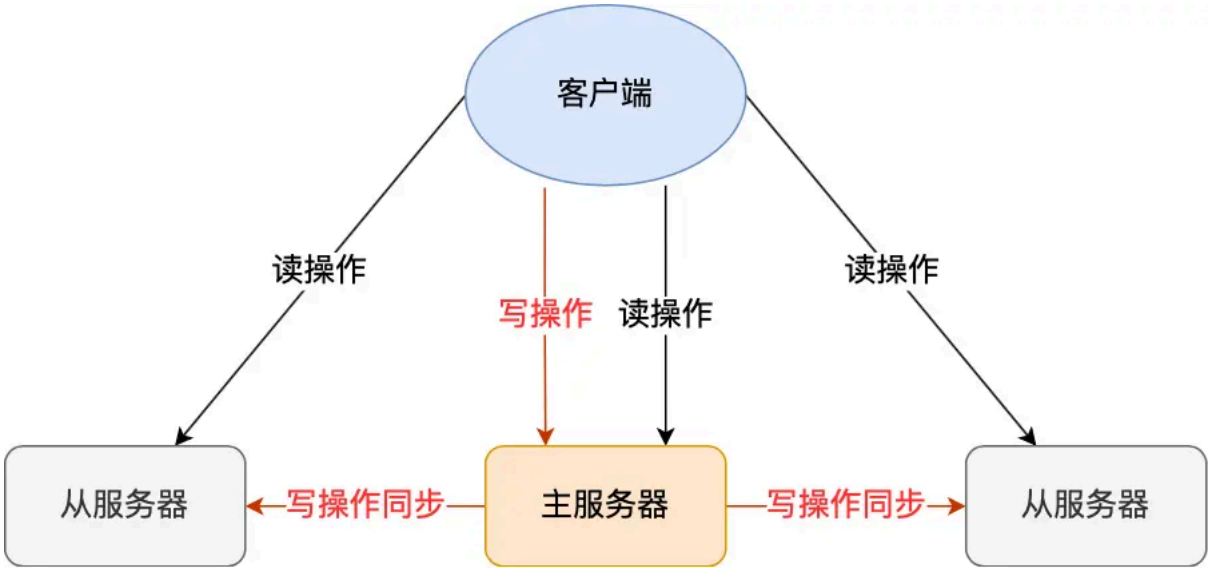
1、Redis如何实现服务高可用?

要想设计一个高可用的Redis服务，一定要从Redis的多服务节点来考虑，比如Redis的主从复制、哨兵模式、切片集群。

（1）主从复制

主从复制是Redis高可用服务的最基础的保证，实现方案就是将从前的一台Redis服务器 ，同步数据到多台从Redis服务器，即一主多从的模式，且主从服务器之间采用的是读写分离的方式。

主服务器可以进行读写操作，当发生写操作时自动将写操作同步给从服务器，而从服务器一般是只读，并接受主服务器同步过来写操作命令，然后执行这条命令。



也就是说，所有的数据修改只在主服务器上进行，然后将最新的数据同步给从服务器，这样就使得主从服务器的数据是一致的。

注意，主从服务器之间的命令复制是异步进行的。

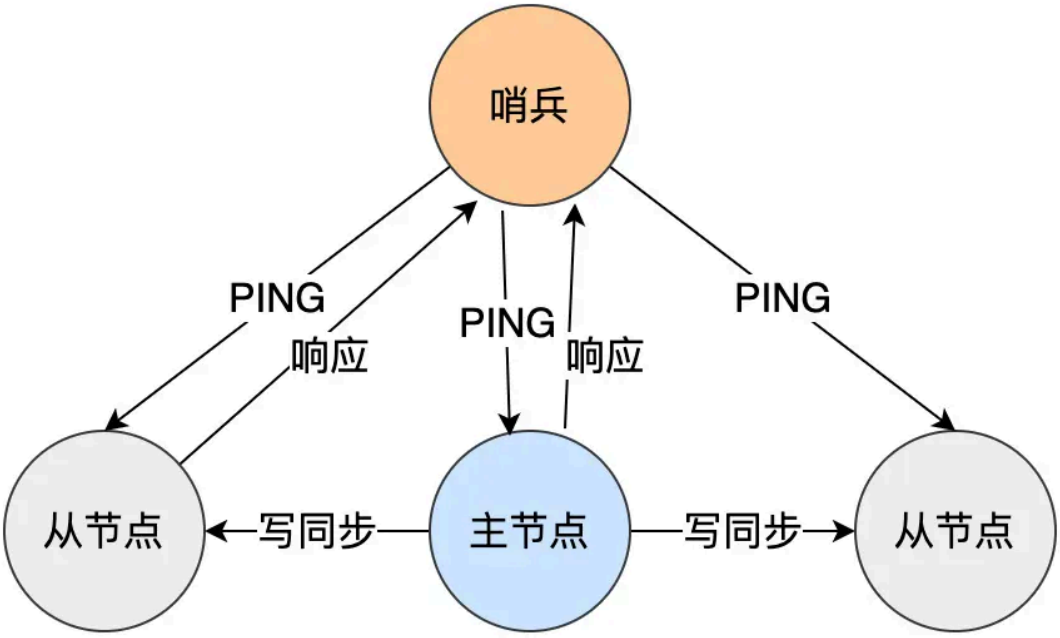
具体来说，在主从服务器命令传播阶段，主服务器接收到新的写命令后，会发送给从服务器。但是，主服务器并不会等到从服务器实际执行完命令后，再把结果返回给客户端，而是主服务器自己在本地执行完命令后，就会向客户端返回结果了。如果从服务器还没有执行主服务器同步过来的命令，主从服务器之间的数据就不一致了。

所以，无法实现强一致性保证（主从数据时时刻刻保持一致），数据不一致是难以避免的。

（2）哨兵模式

在使用Redis主从服务的时候，会有一个问题，就是当Redis的主从服务器出现故障宕机时，需要手动进行恢复。

为了解决这个问题，Redis增加了哨兵模式，因为哨兵模式做到了可以监控主从服务器，并且提供主从节点故障转移的功能。



（3）切片机群模式

当Redis缓存数据量大到一台服务器无法缓存时，就需要使用Redis切片集群方案，它将数据分布在不同的服务器上，以此来降低系统对单主节点的依赖，从而提高Redis服务的读写性能。

六、Redis过期删除与内存淘汰

1、Redis使用的过期策略是什么?

Redis是可以对KEY设置过期时间的，因此需要有相应的机制将已过期键值对删除，而做这个工作的就是过期键值删除策略。

每当我们对一个key设置了过期时间时，Redis会把该key带上过期时间存储到一个过期字典中，也就是说过期字典保存了数据库中所有key的过期时间。

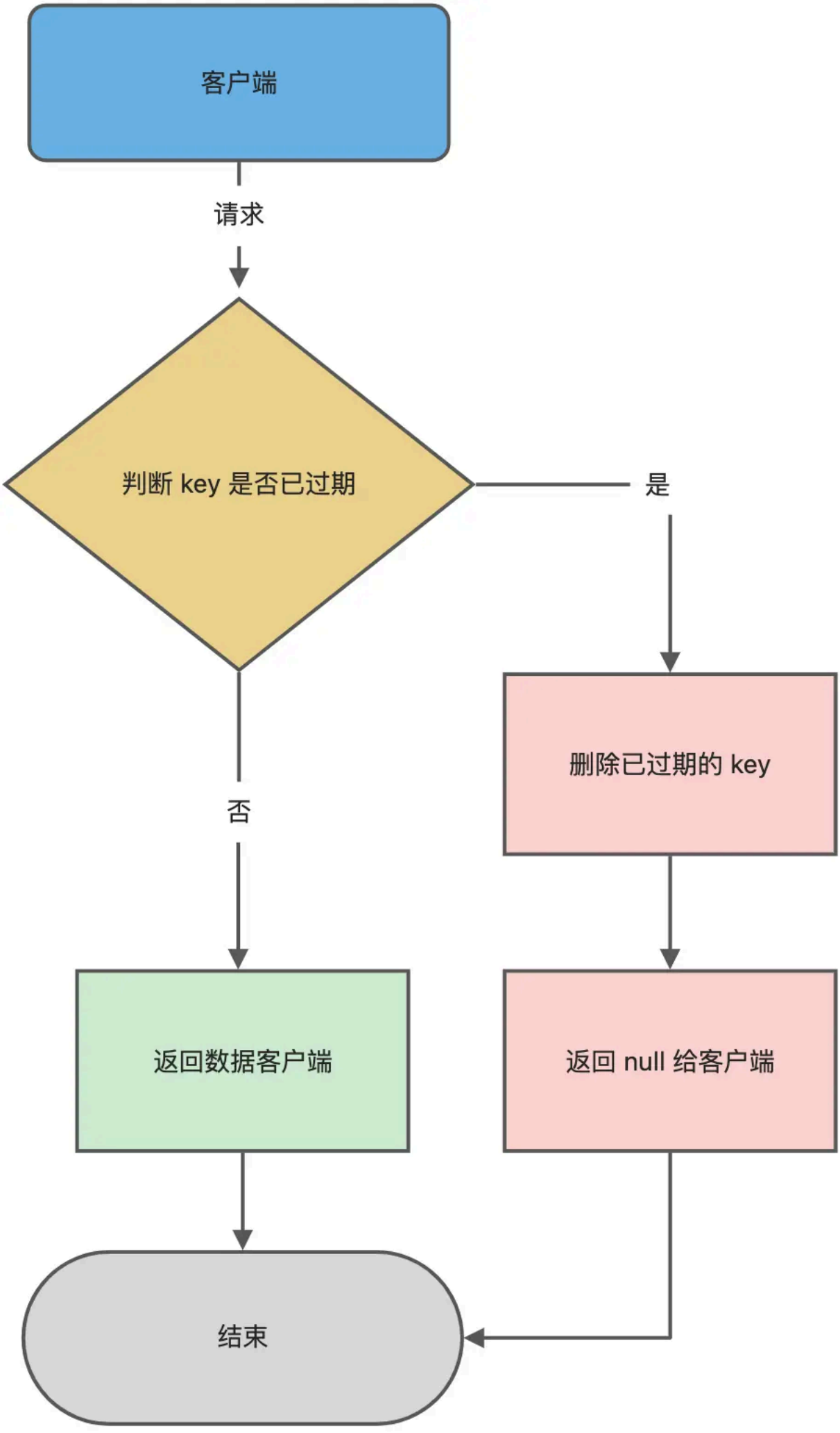
当我们查询一个Key时，Redis首先检查key是否存在于过期字典中：

- 如果不在，则正常读取键值；
- 如果存在，则会获取该key的过期时间，然后与当前系统时间进行比对，如果比系统时间大，那就没有过期，否则判定该key已过期。

Redis使用的过期删除策略是惰性删除+定期删除这两种策略配合使用。

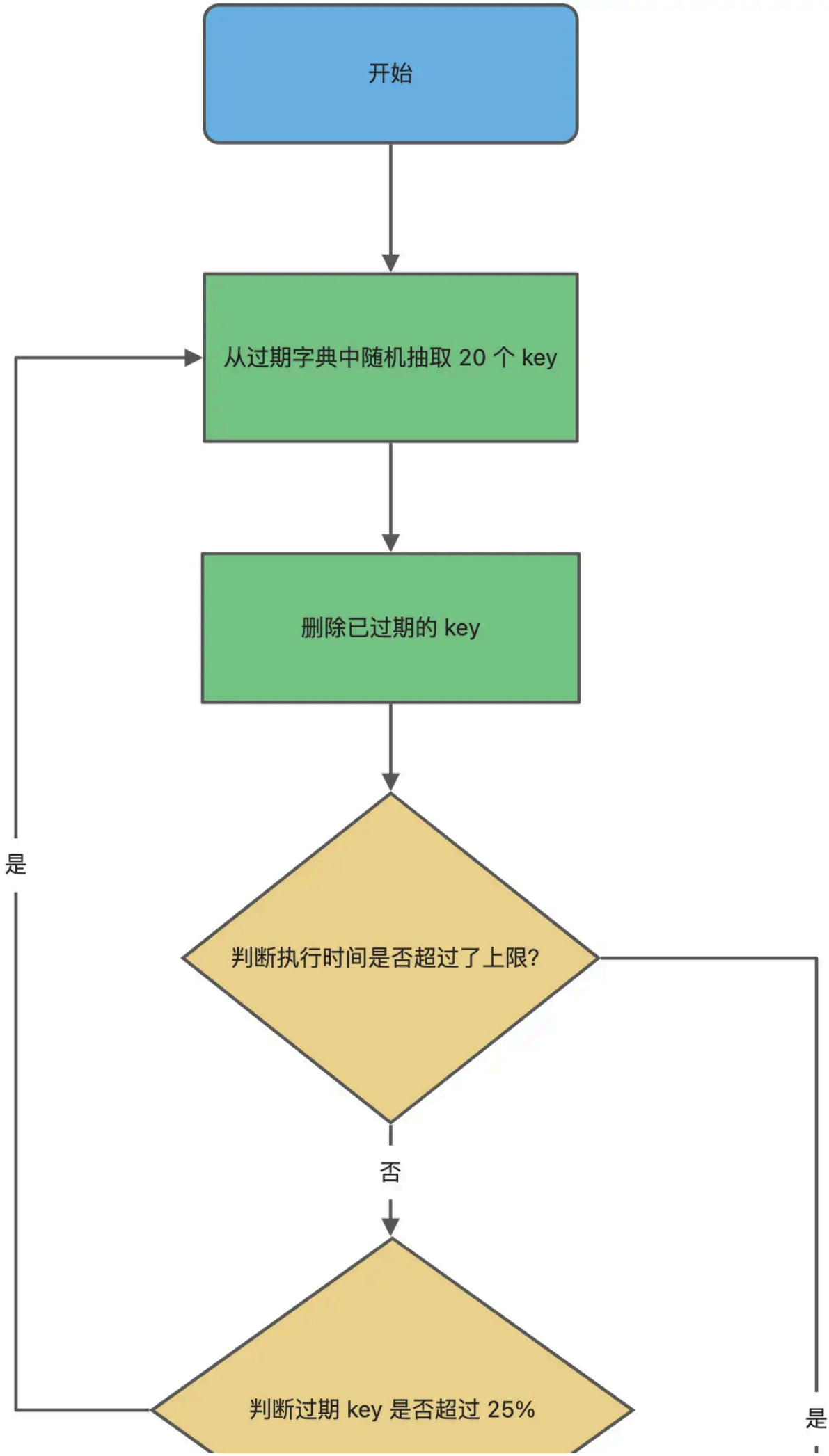
2、什么是惰性删除策略?

惰性删除策略的做法是，不主动删除过期键，每次从数据库访问key时，都检测key是否过期，如果过期则删除该key。


客户端
请求
判断 key 是否已过期
是
否
删除已过期的 key
返回数据客户端
返回 null 给客户端
结束

3、什么是定期删除策略?

定期删除策略的做法是，每隔一段时间随机从数据库取出一定数量的key进行检查，并删除其中的过期key。

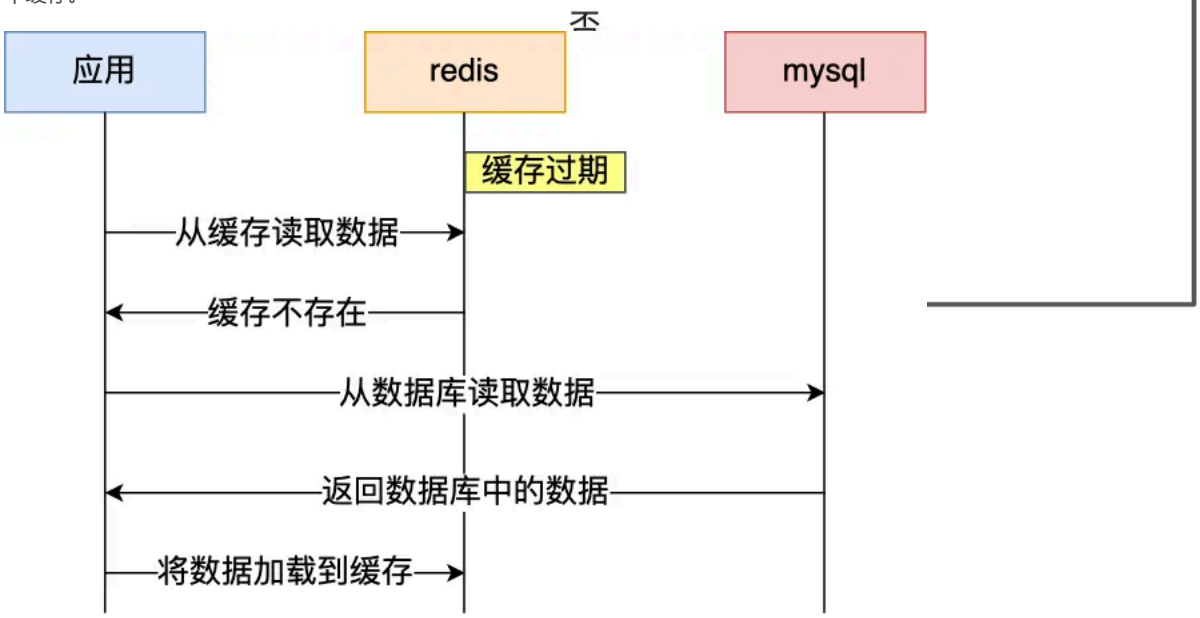

开始
从过期字典中随机抽取 20 个 key
删除已过期的 key
判断执行时间是否超过了上限?
是
否
判断过期 key 是否超过 25%
是

七、Redis缓存设计

1、如何避免缓存雪崩、缓存击穿、缓存穿透?

（1）缓存雪崩

通常我们为了保证缓存中的数据与数据库中的数据一致性，会给Redis里的数据设置过期时间，当缓存数据过期后，用户访问的数据如果不在缓存里，业务系统需要重新生成缓存，因此就会访问数据库，并将数据更新到Redis里，这样后续请求都可以直接命中缓存。



那么，当大量缓存数据在同一时间过期时，如果此时有大量的用户请求，都无法在Redis中处理，于是全部请求都直接访问数据库，从而导致数据库的压力骤增，严重的会造成数据库宕机，从而形成一系列连锁反应，造成整个系统崩溃，这就是缓存雪崩的问题。

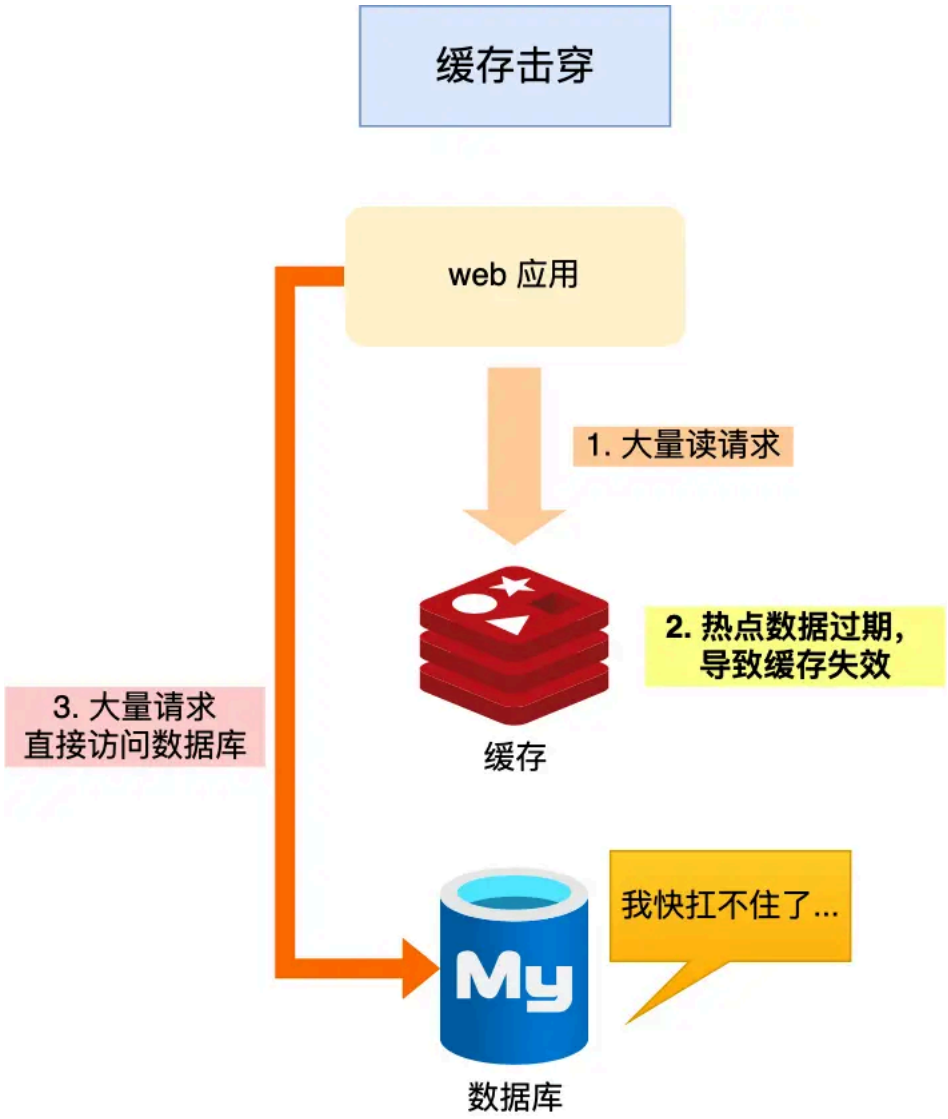
对于缓存雪崩的问题，我们可以采用两种方案解决：

- 将缓存失效时间随机打散。在原有失效时间的基础上增加一个随机值，这样每个缓存过期的时间就不重复了。
- 设置缓存不过期。

（2）缓存击穿

我们的业务通常会有几个数据会被频繁地访问，比如秒杀活动，这类被频繁地访问的数据称为热点数据。

如果缓存中的某个热点数据过期了，此时大量的请求访问了该热点数据，就无法从缓存中读取，直接访问数据库，数据库很容易就被高并发的请求冲垮，这就是缓存击穿的问题。



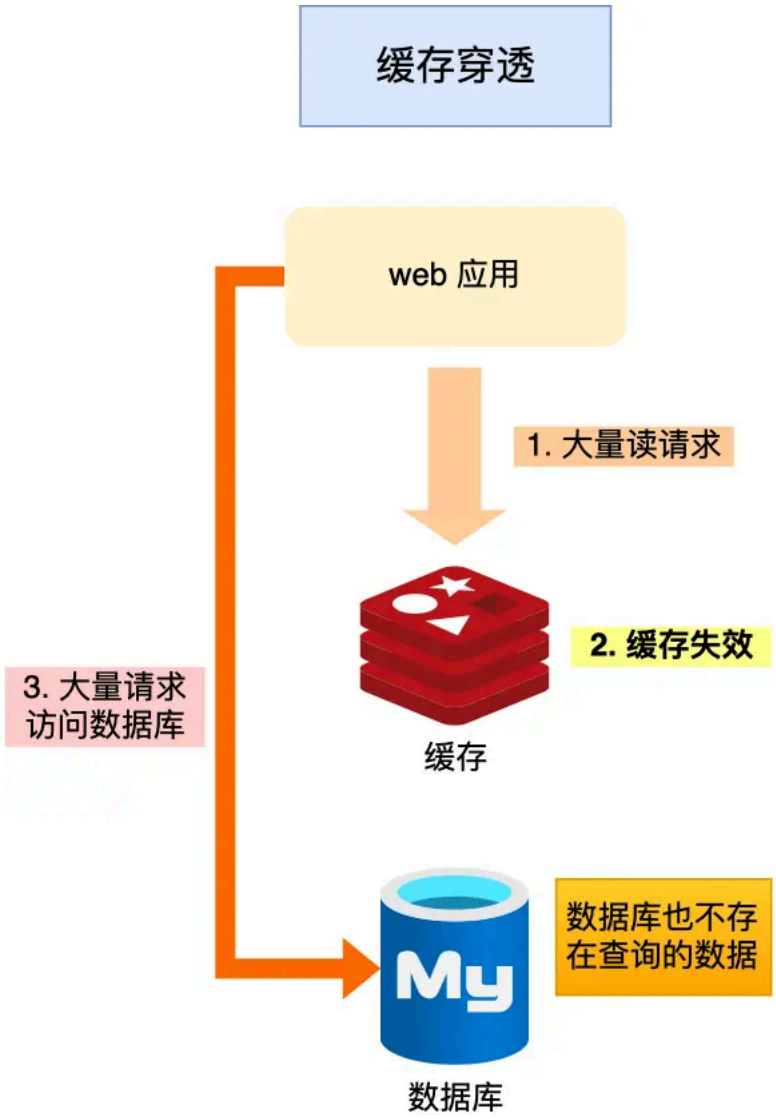
应对缓存击穿可以采取前面说到的两种方案：

- 不给热点数据设置过期时间，由后台异步更新缓存，或者在热点数据准备更新前，提前通知后台线程更新缓存以及重新设置过期时间。

（3）缓存穿透

当发生缓存雪崩或击穿时，数据库中还是保存了应用要访问的数据，一旦恢复相对应的数据，就可以减轻数据库的压力，而缓存穿透就不一样了。

当用户访问的数据，既不在缓存，也不在数据库中，导致请求在访问缓存时，发现缓存缺失，再去访问数据库时，发现数据库中也没有要访问的数据，没办法构建缓存数据，来服务后续的请求。那么当有大量这样的请求到来时，数据库的压力骤增，这就是缓存穿透的问题。



缓存穿透的发生一般由这两种情况：

- 业务误操作，缓存中的数据和数据库中的数据都被误删除了，所以导致缓存和数据库中没有数据；
- 黑客恶意攻击，故意大量访问某些读取不存在数据的业务。

应对缓存穿透的方案，常见的方案有三种：

- 非法请求的限制：当有大量恶意请求访问不存在的数据的时候，也会发生缓存穿透，因此要在API处判断请求参数是否合理、是否有非法值、请求字段是否存在，如果判断不符合就直接返回错误。
- 设置空值或者默认值

显示推荐内容