Redis

哈喽大家好，我是厨子，这是我整理的面经PDF，并且对其进行了分类整理，应该能够对你有所帮助，阅读过程中，有任何问题大家都可以联系我，进行反馈，我的微信是 chefyuan105，备注你的问题即可。

这是我的两个 Github：PDF内容会实时在这里进行更新，大家可以 star 收藏一下。

https://github.com/chefyuan/interview-base

https://github.com/chefyuan/algorithm-base

另外我还建了一个秋招交流群，大家一起打卡，寒假时，还会有一个每日打卡活动，如果你需要的话，可以联系我，备注秋招即可。

秋招小队简介：http://www.chengxuchu.com/#/Exchange/README?id=%e7%a7%8b%e6%8b%9b%e5%b0%8f%e9%98%9f

入队方式

方式一

扫描下方二维码，关注公众号，点击求职交流，选择社招/校招

点击求职交流

方式二

扫码我的二维码，注明来意 校招/社招 + 个人技术栈 + 个人简单介绍，我会尽快同意你的申请，拉你进群。

备注校招/社招

希望大家都可以拿到满意的 offer 。

1.基础知识

1.0 什么是 Redis?

Redis提供了多种数据类型来支持不同的业务场景。Redis 支持事务持久化、LUA脚本、LRU驱动事件、多种集群方案

redis 是一个 key-value 数据库，定期通过异步操作将数据库数据 flush 到硬盘上保存。因为是纯内存操作，所以 Redis 的性能非常出色，是已知性能最快的 key-value 数据库。Redis 支持多种数据结构，然后单个Value 的最大限制为 1GB。所以Redis可以用来实现很多有用的功能。

Redis 的主要缺点是数据库容量受到物理内存的限制，不能用作海量数据的高性能读写，因此 Redis 适合的场景主要局限在较小数据量的高性能操作和运算上。

推荐阅读：https://www.runoob.com/redis/redis-tutorial.html

1.1 Redis 和 memcached 相比有什么优势?

Redis拥有更为丰富的数据类型。

Redis 的速度更快，另外Redis 可以持久化数据。

Redis 的value的最大限制更大。

1.2 Redis官方为什么不开发windows版本

现在linux 版本已经足够稳定，且用户量很大，无需添加windows版本，反而会带来兼容性问题

1.3 一个字符串类型能存储最大容量是多少?

512M，其键也是 512 M

1.4 Redis 的读和写的速度?

读 110000 写 81000

1.5 Redis 为什么将所有数据都放到内存中?

因为Redis 为了达到更快的读写速度，Redis 会通过异步的方式将数据写入磁盘，所以Redis具有快速和持久化的特征。如果不放到内存中，磁盘I/O会严重影响Redis性能。如果设置了最大使用内存，则数据已有记录到达内存限制后则不能继续插入新值。

1.6 使用 Redis 的优点

- 访问速度快：因为都在内存中
- 支持事务
- 数据类型丰富：支持五种数据存储形式
- 特性丰富：Redis可用于缓存，消息，按key设置过期时间，过期后将会自动删除。

1.7 Redis 中的字符串和C语言中的有什么不同

SDS可以直接获取到字符串的长度，时间复杂度为O(1)

会有空间预分配，这样可以减少分配空间的次数，然后长度减少时，不会改变数据结构的内存，会将减少部位标志为可用。

C语言是以 /0结尾，Redis 以长度进行结尾，避免存储字符串出现问题

可以保存文本和二进制数据

推荐阅读：https://blog.csdn.net/cjqh_hao/article/details/89741720

1.8 Redis线程模型

Redis基于Reactor模式开发了网络事件处理器，这个处理器被称为文件事件处理器。它的组成结构为4部分：多个套接字、IO多路复用程序、文件事件分派器、事件处理器。因为文件事件分派器队列的消费是单线程的，所以Redis 才叫单线程模型

推荐阅读：https://blog.csdn.net/m0_37524661/article/details/87086267

推荐阅读：https://blog.csdn.net/qq_14855971/article/details/118918980?spm=1001.2014.3001.5501

2.数据结构

2.0 5 种数据表现形式

- String字符串（主要用于常规计数，粉丝数等）
- Hash散列（用户信息，商品信息等）
- List 列表（消息列表，粉丝列表等）
- Set 集合（共同好友，共同关注等）
- SortedSet有序集合（礼物排行榜等）

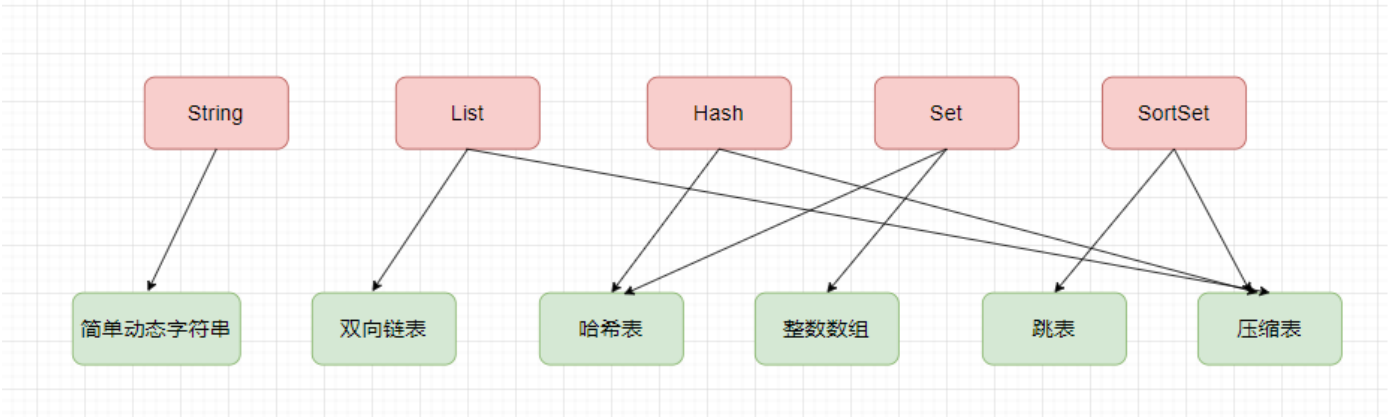
以上是键值对的保存形式

推荐阅读：https://www.runoob.com/redis/redis-data-types.html

2.1 6 种底层数据结构

共有六种

- 简单动态字符串
- 双向链表
- 压缩列表
- 哈希表
- 跳表
- 整数数组



简单动态字符串

SDS可以直接获取到字符串的长度，时间复杂度为O(1)

会有空间预分配，这样可以减少分配空间的次数，然后长度减少时，不会改变数据结构的内存，会将减少部位标志为可用。

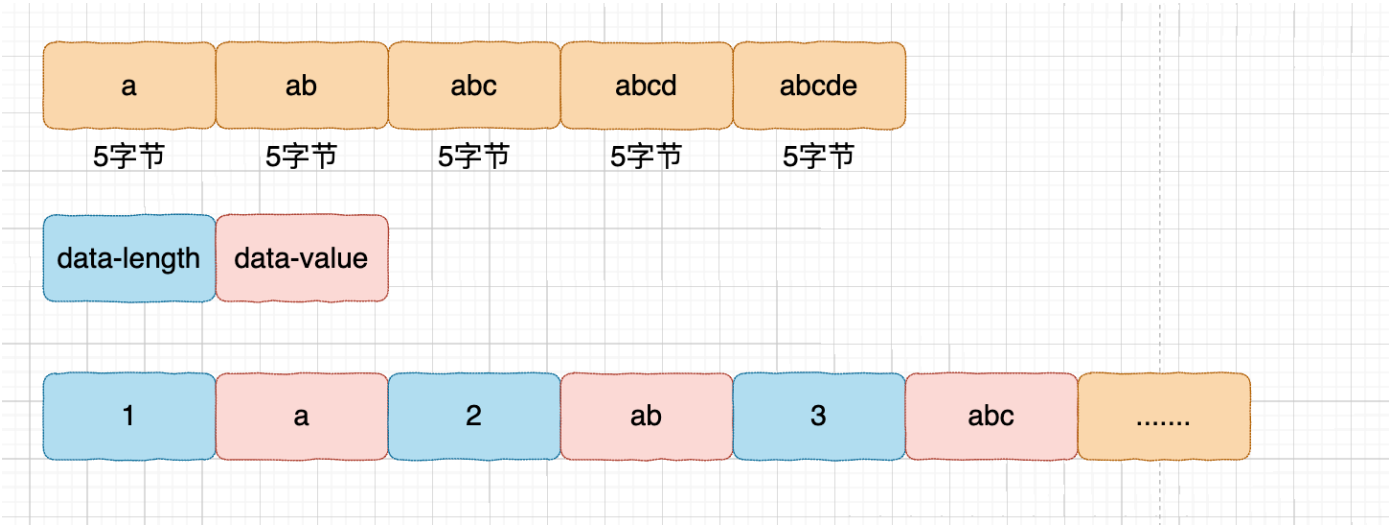
C语言是以 /0结尾，Redis 以长度进行结尾，避免存储字符串出现问题

可以保存文本和二进制数据

双向链表

压缩列表

压缩原理



原理很简单，因为数组的内存是连续分配的，如果我们给所有的value分配相同的内存大小，这样的话则会造成大量的内存浪费，所以我们对每个value 的内存进行存储时，先计算出其大小，然后通过大小为其分配内存。

这个也是高频考点

哈希表

跳表

推荐阅读：https://zhuanlan.zhihu.com/p/53975333?ivk_sa=1024320u

整数数组

3.缓存

3.0 缓存分类

只读缓存：缓存只负责读，任何写操作都会直接操作在数据库上，然后删除缓存里的数据，然后下次再请求数据时，发现缓存内没有，则重新请求数据库，这样缓存内存在的数据仍为新数据。

读写缓存：读写缓存，顾名思义，其写操作也会落到缓存上，会在缓存中进行修改，但是我们要考虑这个情况，那就是我们修改了缓存，数据库宕机的情况。则会造成数据不一致的情况。

然后写入数据库的时机有两种形式，一种是同步写回，一种是异步写回。

同步写回则是，缓存修改之后，立刻修改数据库，这样可能降低响应速度，但是能够保证安全性，这里我们就需要使用事务来搞定。一起执行才行，不然就会产生数据不一致的情况。

异步写回：则会缓存修改之后，找机会再进行修改数据库。

3.1 缓存淘汰机制

在设置了过期时间的数据中进行淘汰

- 随机淘汰（random）
- LRU（随机选取N个值，然后在这N个值里选择最久未使用的）
- LFU（使用次数最少）
- TTL根据过期时间淘汰，越早过期的越删除

所有范围内进行淘汰

- LRU
- LFU
- 随机

如何处理被淘汰的数据

如果数据是干净数据，则直接淘汰，如果数据是脏数据，则需要先写入数据库，然后再进行淘汰，不过Redis 使用的是修改完之后，直接写入数据库，所以不会出现清理缓存的时候出现脏数据的情况。

推荐阅读：https://zhuanlan.zhihu.com/p/343963744

3.2 如何解决缓存和数据库内容不一致的问题

`一致性`

缓存中有数据，那么需要和数据库一致

缓存中没有数据，那么数据库中的就是新数据

`单线程环境下`

`消息队列`

我们可以使用消息队列来搞定，将待修改的值存入消息队列，如果修改成功则删除消息队列里的数据，如果失败，则从消息队列中取出，然后继续修改。含义则是`重试`

我们平常开发中，应该优先使用先更新数据库再删除缓存的方法，这样能够减小数据库压力。因为先删缓存会有缓存缺失的情况。

`多线程环境下`

这种情况，则会出现新的问题，我们思考一下，当A线程修改了数据库，然后写入缓存，此时B线程又修改了数据库，则会导致缓存中存的仍然是旧值。还会有另外三种情况，如果在并发环境下，如果是读写问题，则会对业务影响不大，但是写写问题则会影响很大，所以这里我们可以借助`分布式锁`redlock来解决该问题。

推荐阅读：https://www.jianshu.com/p/a8eb1412471f

3.3 缓存穿透，缓存雪崩，缓存击穿

缓存雪崩

是指缓存中大量键到期，而查询量过大，发现缓存中没有该键，则会去请求后台，引起数据库压力过大甚至宕机。

解决方案

- 过期时间设置为随机，这样就不会出现短时间内大量过期的情况。
- 使用分布式数据库，分别请求不同的数据库，则会减少某一数据库的压力。
- 设置热点数据永远不过期
- 核心数据可以访问数据库，非核心直接返回，进行一个降级

缓存击穿

缓存击穿是指，针对某个访问非常频繁的热点数据的请求，无法在缓存中进行处理，紧接着，访问该数据的大量请求，一下子都发送到了后端数据库，导致了数据库压力激增，会影响数据库处理其他请求。

- 设置热点数据永远不过期

缓存穿透

缓存雪崩和缓存击穿都是因为键过期的情况，虽然缓存中没有该数据了，但是数据库中还有，虽然会对数据库造成压力，但是还是有机会解决。

情况如下，我们访问缓存时，发现缓存中没有该数据，则去查找数据库，发现数据库中也没有该数据，这样就会给缓存和数据库造成很大的压力。

发生这种事情有两种情况

误删除：被管理员一不小心将缓存中和数据库的数据都删了

恶意攻击：另一种情况就是被别人恶意攻击，故意拿数据库和缓存中没有的数据进行发出请求。

- 设定空值和缺省值
- 使用布隆过滤器判断是否含有该键，多用于缓存更新较少的情况，因为需要同步更新布隆过滤器
- 入口处检测（前端界面过滤，过滤掉部分）

注：布隆过滤器的原理：使用N个哈希函数，得出N个哈希值，并将哈希表的N个位置标记成 1，如果已经为 1 则无需标记，判断值是否存在时，计算N个哈希值，对应的位是否存在 0 ，存在 0 则表明该值不存在。

推荐阅读：https://www.cnblogs.com/xuanyuan/p/13665170.html

3.4 缓存污染

缓存污染的含义：某些不常被访问的数据，白白浪费内存空间

解决方法则是利用我们的缓存淘汰策略。我们来分析一下各个`缓存淘汰策略`能不能解决缓存污染的情况

- LRU 不能解决，他可以解决最久未访问的键，但是有些不常用的键，可能会偶尔被访问一次。
- LFU最近最少使用，在使用时间和使用次数上都有要求，所以则可以解决缓存污染的情况。

LFU原理，使用一个字符串，然后前半部分保存时间戳，后半部分保存使用次数。

3.5 缓存的过期删除策略

定期删除：Redis 每隔一段时间（默认 100ms），就会随机选出一定数量的数据，检查它们是否过期，并把其中过期的数据删除，这样就可以及时释放一些内存。

惰性删除：当一个数据的过期时间到了以后，并不会立即删除数据，而是等到再有请求来读写这个数据时，对数据进行检查，如果发现数据已经过期了，再删除这个数据。

4 Redis集群方案

https://www.cnblogs.com/51life/p/10233340.html

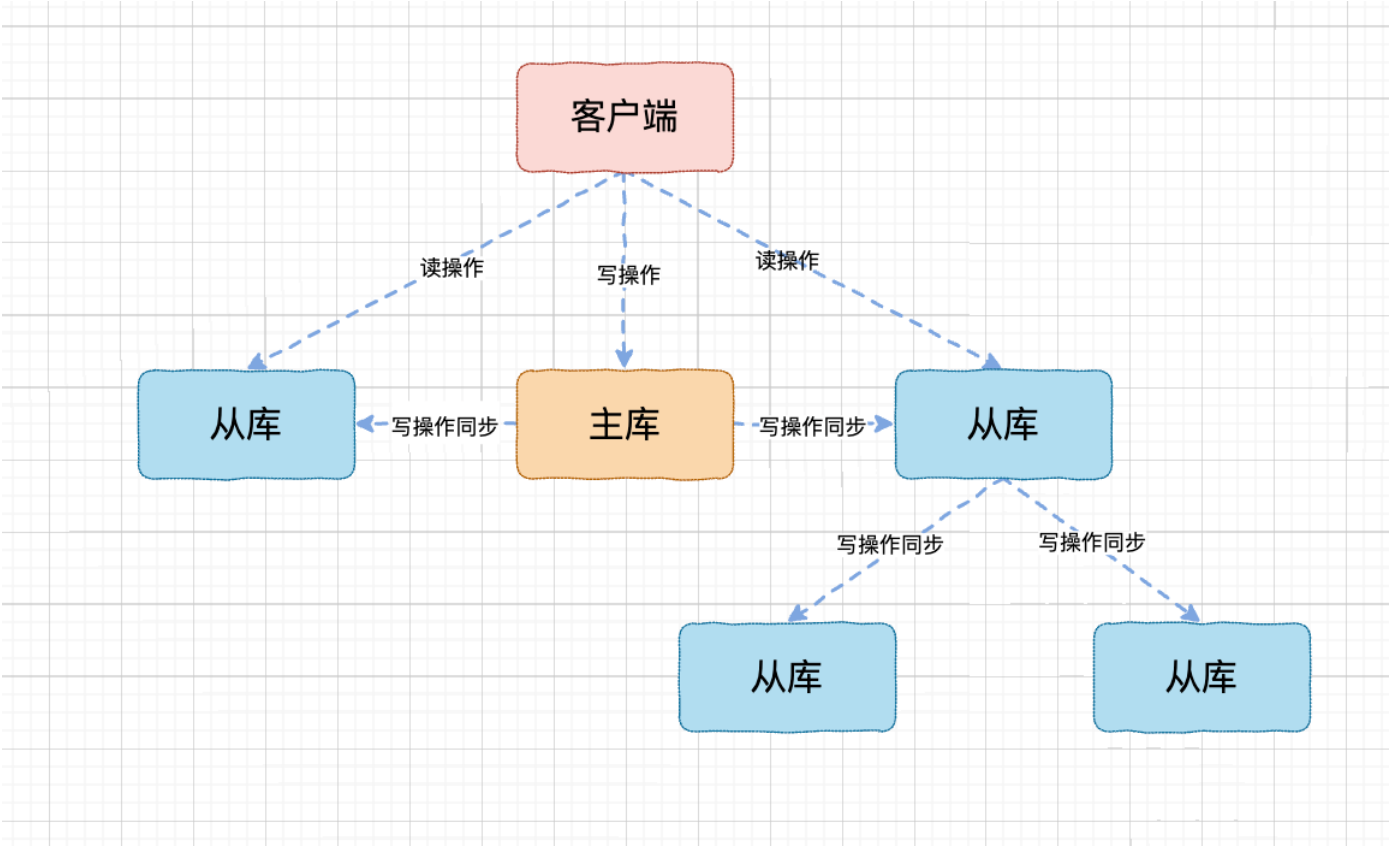
4.0 主从复制

主从复制模式，主库和从库的数据一致，采取的是读写分离的方式，进而来降低Redis 服务器的压力

读操作：主和从都可

写操作：先写主，然后主同步给从（防止数据不一致）

主从复制过程



`建立链接，` 使用`RDB快照进行同步。`

但是我们思考一下，同步过程中，主服务器继续执行写操作，那么从服务器则会漏掉这部分操作，所以主服务器在从服务器的同步过程中，会将新的写操作放到缓冲区里，从库同步完毕后，再执行缓冲区的文件，实现同步。

另外为了缓解主库同步压力，我们采取主-从-从的模式

如果主库和从库之间网络断开，则会导致不一致，主库这时会将其中断期间执行的操作，写入缓冲区，然后从库苏醒之后，会执行缓冲区的语句完成同步，不过缓冲区为环形数据结构，如果从库写入较慢，同样会有覆盖的风险，所以我们可以将该数据结构设置的大一些。

4.1 主从同步和故障切换存在哪些坑

`主从数据不一致`

用户从从库读到的数据和主库中的不一致，因为我们的写操作是落在主库上的，先操作主库，然后再同步到从库，所以有可能存在延迟的情况。

我们可以这样解决

- 保证网络链接顺畅
- 设定一个监视器，然后通过监视器来监控复制进度，当复制进度大于阈值时，则不让其从从库访问

`读取过期数据`

因为当有数据过期时，如果是在主库读取到，则会执行删除策略，然后同步到从库，但是如果在从库读到过期键时，则不会删除，3.2之前会返回原来的值，3.2之后会返回空值。

`不合理配置`

4.2 哨兵集群

`哨兵模型算是主从复制模型的拓展，我们假设有四个节点，一主三从，当某个主节点挂掉之后，某个从节点顶替主节点。`

我们可以通过哨兵节点来执行这个操作，我们的哨兵节点用来监控主节点，当发现主节点挂掉之后，选取某个从节点来替代主节点，

但是如果某天哨兵节点挂了之后，我们整个监控系统也就完蛋了，所以我们可以搭建哨兵集群，这样整个哨兵集群挂掉的概率就很小了，然后当其中一个哨兵判断主节点挂掉之后，询问其他节点该主节点是否挂掉，如果超过一定值的哨兵节点判断挂掉之后，则认定该节点已经挂掉。

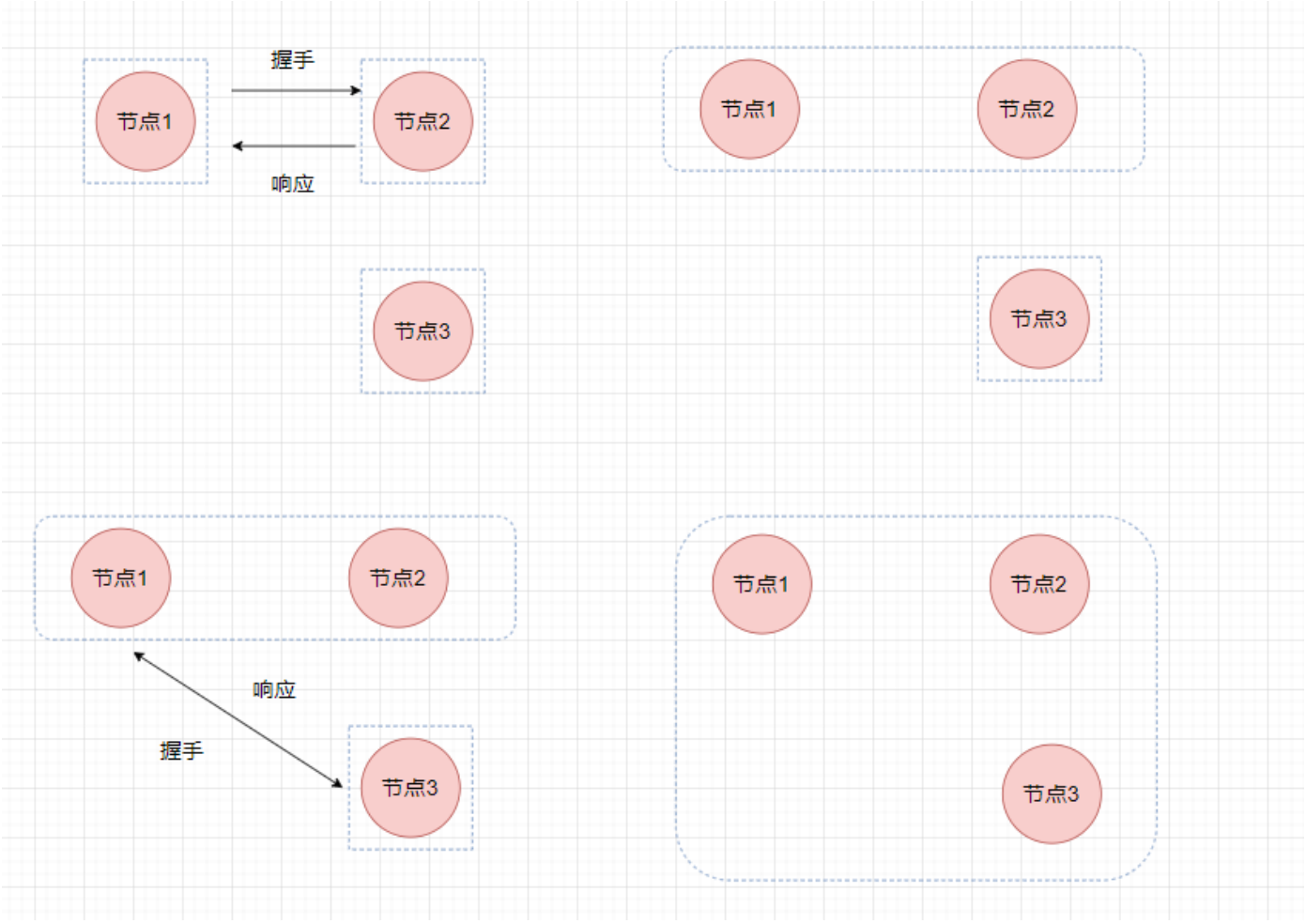
https://zhuanlan.zhihu.com/p/65504905

https://www.cnblogs.com/flashsun/p/14692643.html

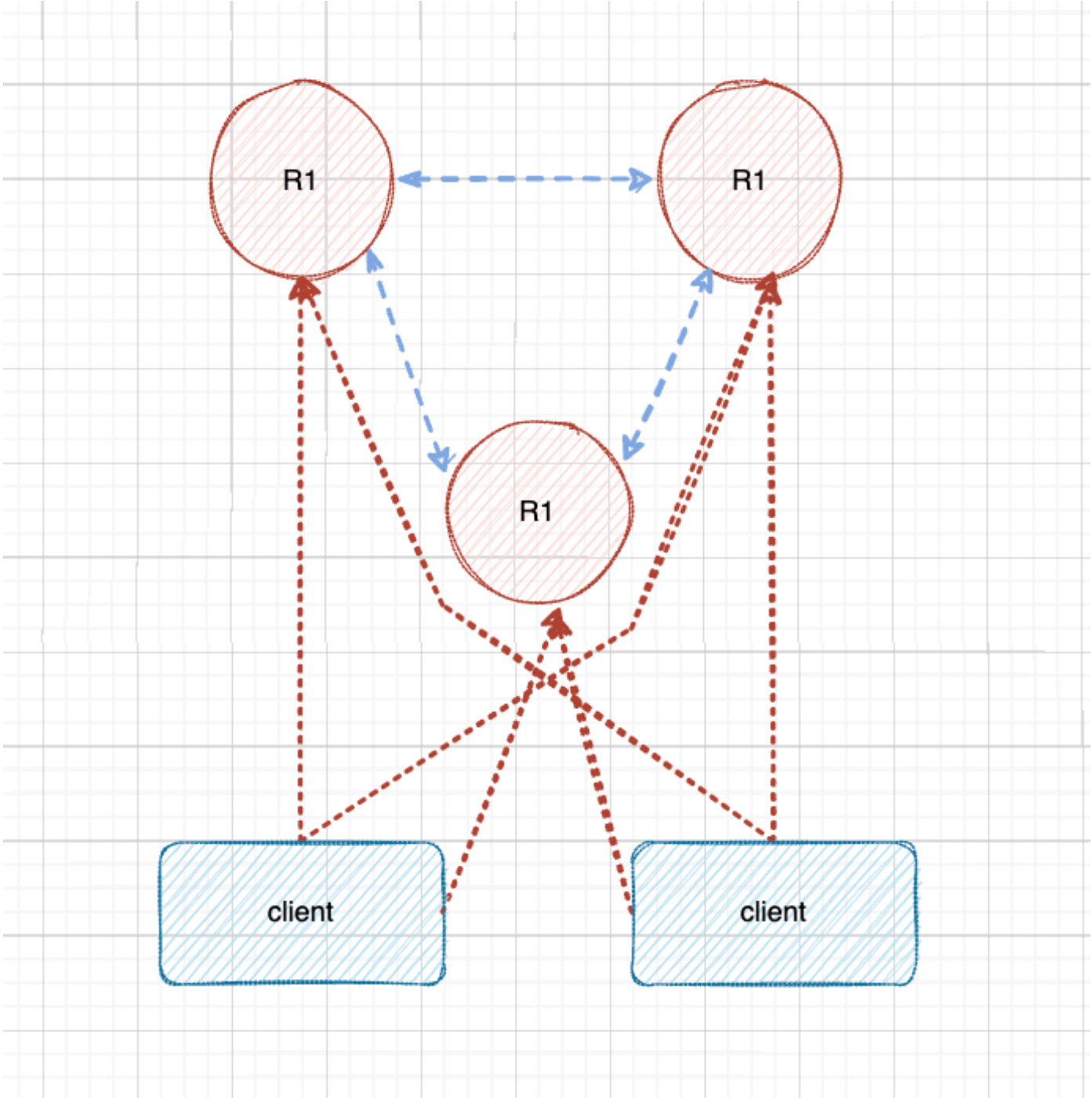
4.3 Cluster

Redis集群是Redis 提供的分布式数据库方案，集群通过分片的来进行数据共享，并提供复制和故障转移功能。

一个Redis 集群通常由多个节点组成，下图为集群的创建过程。



集群的数据结构，clusterNode 结构保存了一个节点的当前状态，会保存自己的状态，还会保存其他节点的状态。



4.4 什么是槽指派?

Redis 集群通过分片的方式来保存数据库中的键值对，如果`有槽没有得到处理，那么集群将处于下线状态。`

集群中会通过slots数组进行来判断节点是否处理槽i,节点之间会告知其他节点自己负责什么槽。

在集群中执行命令

- 如果键所在的槽并正好指派给了当前节点，那么节点立刻执行该命令
- 如果键所在的槽没有指派给当前节点，那么节点会像客户端发送一个`MOVED错误`，指引客户端转向至正确节点。

Redis共有 16384个哈希槽，每个key通过CRC16的校验和对16383（含有0）做与运算，来计算key在哪个槽。

4.5 说说什么是 ASK 错误?

这个错误多发生于重新分片的过程中，我们将某节点的槽迁移到另一节点时，此时客户端传来命令，如果没有在对应的节点内查询到该槽中值，则说明已经发生了迁移，此时则返回 ASK 错误，然后开始去目标节点进行查询。

4.6 什么是 Moven 错误?

这是正常请求错误，告知该数据不在该服务器的槽里。

4.7 故障检测?

每隔一段时间，主节点之间会发送ping信息，如果没有一定时间内，得到回应，那么则被认为是疑似下线，当半数节点认为其疑似下线时，则认定其为已下线。

4.8 故障转移步骤?

- 复制下线主节点的所有从节点里面，会有一个节点被选中。
- 被选中的节点会执行SLAVEOF no one 命令，成为新的主节点，
- 将下线节点的所有槽指派下线，然后指向新的主节点。
- 向其他节点发送ping信息，告知别人自己变成了主节点。

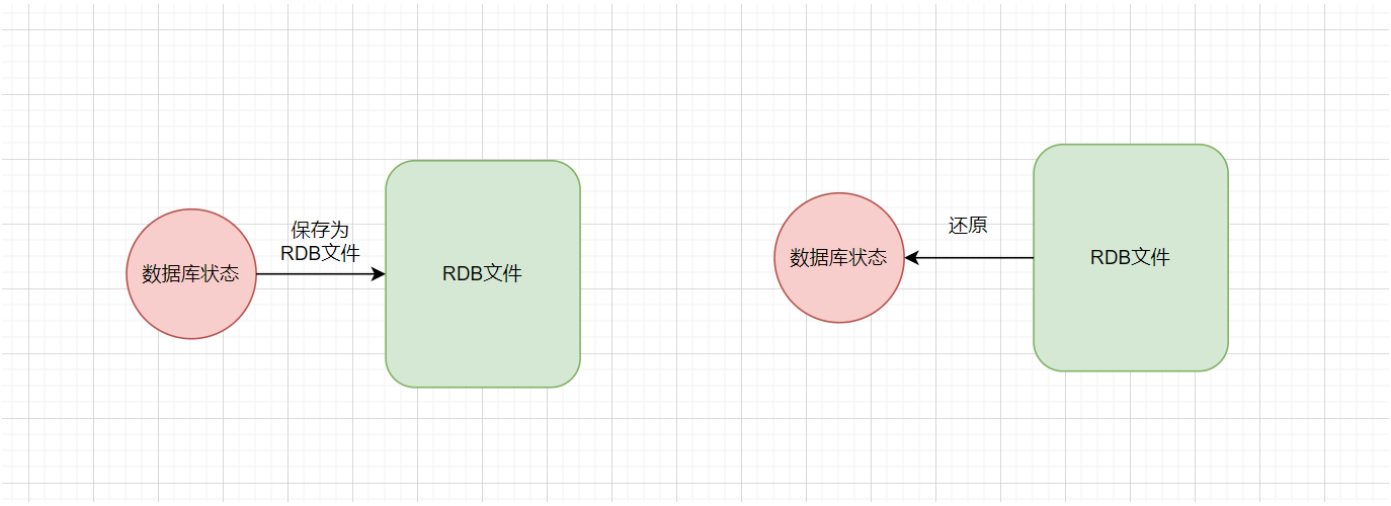
5.Redis 持久化

Redis主要通过两种方法实现持久化，一种是RDB持久化，一种是AOF 持久化，下面详细说一下两种持久化机制

5.0 RDB 持久化

因为Redis整个加载都在内存中，所以我们可以通过RDB持久化将内容中的数据保存到内存中去，避免数据丢失。

RDB 持久化可以手动执行，也可以定期自动执行。不过AOF 的更新频率更高，当开启AOF持久化时，会优先使用AOF文件来还原持久化。



Redis中主要有两个命令生成Redis的RDB 文件，一个是SAVE,一个是BGSAVE文件。

其中SAVE文件会堵塞Redis服务器进程，直到文件创建完毕（不好用），`BGSAVE会派生出一个子进程来进行创建文件。`

另外需要注意的是，RDB文件的话，会随着数据量越来越大，数据量太大时，则会加大生成快照的难度。

但是我们思考一下，我们生成RDB的过程中，如果有数据发生了更改该怎么办呢?

fork 出的子进程用于生成RDB文件，然后在生成过程中，如果有写操作，则让主进程将这块数据复制一份，在复制份上进行写。

我们可以通过使用命令，设置BGSAVE每隔一段时间执行。但是这个时间设置多大，是一件比较困难的事。

所以我们采用`增量快照`

制作一次全局快照之后，后续将修改的值，补写入全局快照即可。

但是我们单独记录的话， 修改的数据是身份庞大的，所以我们可以采用 AOF 和 RDB 结合的方法来搞定。使用AOF来记录这块修改的内容。

5.1 RDB文件结构

REDIS

db_version

database

EOF

check_sum

REDIS:判读是否是Redis文件

db_version:代表RDB文件的版本

database:代表对几号数据库进行保存。

EOF：代表正文部分结束

check_sum：根据前几项求得的校验和

每个非空数据库在RDB文件中都可以保存为

SELECTDB

db_number

key_value_pairs

SELECTDB:表示将要读一个数据库号码。

db_number：代表数据库的号码。

key_value_pairs内容

5.2 AOF持久化

优点：AOF写后日志，他是数据写完之后才写入日志中，这种不会影响写操作。写前日志WAL

缺点：

- 有可能会缺失日志，那就是当AOF 还没来的及写的时候发生宕机，则会造成缺失日志的情况。

- 另一种情况则是会影响下次写入操作

AOF三种写入磁盘时机

同步写回：发生写操作之后，立马写入磁盘中

每秒写回：每秒写入一次

操作系统控制写回：

配置项	写回时机	优点	缺点
Always	同步写回	可靠性高，数据基本不丢失	每个写命令都要落盘，性能影响较大
Everysec	每秒写回	性能适中	宕机时丢失1秒内的数据
No	操作系统控制的写回	性能好	宕机时丢失数据较多

5.3 AOF重写

AOF重写的含义是将多条语句合并成一条，AOF写入是以追加的形式来的，重写就是保证其最新状态。

AOF的重写过程是由子线程来完成的，另外重写过程是这样的，子线程会拷贝主线程的内存，然后进行重写，这时如果有写操作继续执行的话，则放入缓冲区内，会同时放入重写缓冲区和写入缓冲区。

这时我们又有问题了，为什么不让父子进程同时写一个文件，因为这样会产生竞争，然后我们处理竞争又会影响速度。

推荐阅读：https://www.cnblogs.com/xuanyuan/p/13689484.html

6.Redis分布式

6.0 Redis可以做消息队列吗?

消息队列需要有三种特性

- 有序性
- 重复消息处理
- 消息可靠性

我们使用List 则可以做到上面三条，有序性，重复消息（设置id），然后通过某些关键字实现可靠性（RPOPLPUSH）

基于 Streams 的消息队列解决方案

这个是Redis 专门为消息队列设置的数据类型

可以保证插入有序，并生成唯一id

会使用内部队列来保证可靠性。

6.1 并发访问（原子）

主要使用两种方法实现并发，`加锁` 和 `原子操作`

加锁缺点：会降低性能，另一个就是需要加分布式锁，加分布式锁会比较困难。

原子操作主要有两种方法

1.将多条操作合并为一个

比如我们修改库存的情况，获取值-减库存-存值，这就是三个操作，我们可以使用Redis 自带的关键字来使其变为原子操作

INCR/DECR增值和减值

2.使用lua 脚本，使其保证原子性

使用lua 脚本，将减库存的三个操作进行合并，其实保持原子性

6.2 Redis 实现分布式锁

单个节点上的锁

分布式锁的实现需要两个条件

- 保证加锁和解锁的原子性
- 在共享存储上设置锁变量，必须保证锁变量的可靠性

实现分布式锁的方法

使用setnx 函数，完成分布式锁

但是这个函数有两个问题

1. 没有id，有可能 A 加的锁，被B 给解了
2. 加锁之后，A线程崩溃，导致锁一直未被释放

解决方法，给每个线程设置一个id，解析还需系铃人

给锁的设定设置一个有效时间，到期未解锁，则进行，主动释放。注意上诉操作要保证原子性

多个节点的锁

Redlock 算法的基本思路，是让客户端和多个独立的 Redis 实例依次请求加锁，如果客户端能够和半数以上的实例成功地完成加锁操作，那么我们就认为，客户端成功地获得分布式锁了，否则加锁失败。

缺点是锁比较重，降低业务效率。

http://zhangtielei.com/posts/blog-redlock-reasoning.html

7. Redis Java客户端

Redision、Jedis、lettuce等等，官方推荐是Redission

7.1 Redission 和 Redis有什么关系

Redission是一个高级的分布式协调Redis客户端。

7.2 Jedis和Redission相比有什么优缺点

Jedis

- 轻量，简洁，便于集成和改造
- 支持连接池
- 支持pipelining、事务、LUA Scripting、Redis Sentinel、Redis Cluster
- 不支持读写分离，需要自己实现
- 文档差

Redission

- 基于Netty实现，采用非阻塞IO，性能高
- 支持异步请求
- 支持连接池
- 支持pipelining、LUA Scripting、Redis Sentinel、Redis Cluster
- 不支持事务，官方建议以LUA Scripting代替事务
- 支持在Redis Cluster架构下使用pipelining
- 支持读写分离，支持读负载均衡，在主从复制和Redis Cluster架构下都可以使用
- 内建Tomcat Session Manager，为Tomcat 6/7/8提供了会话共享功能
- 可以与Spring Session集成，实现基于Redis的会话共享
- 文档较丰富，有中文文档

8.常考其他

https://zhuanlan.zhihu.com/p/272181377

8.0 有MySQL不就够了吗？为什么还要使用Redis

高性能：我们用MySQL查询数据时，需要从磁盘进行查找，速度是比较慢的，如果我们经常对该值进行查找，则会一遍一遍的请求MySQL数据库，但是如果我们将其存入缓存，再次请求时，因为是直接访问内存，速度是极快，当数据库的内容发生改变时，只需改变缓存的值即可。

高并发：直接操作缓存能够承受的请求是远远大于直接访问数据库的，所以我们可以将部分值存到缓存中去，当用户请求时直接从缓存中获取即可。

8.1 为什么不用Guava，而选择Redis做缓存

严格意义上来说，缓存分为分布式缓存和本地缓存。

本地缓存含义为，每个程序实例有一个自己的缓存，其中的内容也不一致，当某个实例结束时缓存也就消失。

Redis和Memcached缓存则是分布式缓存，在多实例的情况下，每个缓存共享一个缓存。具有缓存一致性。这也是它们的优点所在，但是他们也有缺点，就是要保持高可用性。

8.2 Redis和Memcached的区别

- 类型

  Redis是一个开源的内存数据结构存储系统，用作数据库，缓存和消息代理

  Memcached是一个免费的开源高性能分布式内存对象，通过减少负载来加速动态web应用程序

- 数据结构

  Redis支持字符串，散列表，列表，集合，有序集，哈希表

  Memcached支持字符串和整数

- 执行速度

  Memcached的读写速度慢于Redis

- Value值

  Redis最大512M

  Memcache最大只能1M

https://blog.csdn.net/guoguo527/article/details/108818556

8.3 Redis为啥那么快?

- 内存数据库，所有操作都在内存上
- 高效的数据结构
- 单线程避免了上下文切换
- 非阻塞I/O：Redis采用epoll做为I/O多路复用技术的实现，再加上Redis自身的事件处理模型将epoll中的连接，读写，关闭都转换为了时间，不在I/O上浪费过多的时间。

Redis基于Reactor模式开发了网络事件处理器，处理器为文件处理器，然后这个是单线程的，它采用IO多路复用机制（epoll），来监听Socket，根据Socket上的事件，来处理事件，并为其分配相应的事件处理器来处理这个事件。

推荐阅读：https://blog.csdn.net/qq_14855971/article/details/113564022

8.4 Redis扩容

redis 解决冲突的方法是使用链地址法，另外当容量不足的时候，则使用Rehash 进行扩容。

rehash

给哈希表 2 分配更大的空间，

例如是当前哈希表 1 大小的两倍；

把哈希表 1 中的数据重新映射并拷贝到哈希表 2 中；

释放哈希表 1 的空间。

渐进式rehash则是不一次性拷贝，当访问到某个数据时，再进行拷贝。

8.5 Redis如何实现事务?

事务的生命周期

开启事务（Redis 中使用 MUlTI 来实现）

执行事务中的操作：增删改查等操作，Redis 会将这些操作保存在一个队列中，暂且先不执行

提交事务 EXEC 提交事务

Redis 的事务能保证哪些特性?

`原子性`

- 命令入队时就报错，会放弃事务执行，保证原子性；命令入队时没报错，
- 实际执行时报错，不保证原子性；
- EXEC 命令执行时实例故障，如果开启了 AOF 日志，可以保证原子性。

`一致性`：在事务开始之前和事务结束以后，数据库的完整性没有被破坏。这表示写入的资料必须完全符合所有的预设约束、触发器、级联回滚等

Redis 事务满足一致性

`隔离性`

如果开启了WATCH 机制则没有被破坏，因为其会监控键值对的改变。保证一致性

`持久性`

不能保证持久性

推荐阅读：https://zhuanlan.zhihu.com/p/135241403

8.6 Redis的使用场景

缓存：用来保存热点数据

限时业务：因为我们可以设置淘汰时间，比如验证码，登陆时间等

计数器相关

排行榜：我们可以使用zset

点赞共同好友等

消息队列

8.7 脑裂

数据丢失的问题

脑裂：`多个主服务器，然后客户端，不知道该写入那个主服务器。则出现了多个客户端写入多个主服务器的情况。`

出现原因：主库假失败

脑裂为什么会发生数据丢失?

- 主库中的数据，未同步到从库，然后此时主库崩溃，则就出现了数据丢失的情况。
- 主服务器下线之后，有可能从库依旧和原来的主库进行交互，这样会导致新数据，放在不同的主库上。

因为我们此时产生脑裂，然后此时有两个主机，新写的数据会写入两个主机，然后造成两个主机都会保存数据，另外当切换新的主机的时候，原来的主机会清理掉所有数据，然后执行新主机发来的RDB文件，进而出现了数据丢失的情况。

如何防止脑裂，通过配置从库和主库发送ACK消息的延迟时间来解决。

min-slaves-max-lag

8.8 Redis 冲突的怎么办？Rehash，负载因子?

采用链地址法解决冲突，rehash扩容，渐进式rehash，负载因子为大于 1 或 大于 5 的时候进行扩容。