

一、Redis 概念理解.....	2
1. 什么是 Redis?.....	2
2. Redis 的特点有哪些?	3
3. Memcache 与 Redis 的区别都有哪些?.....	3
4. Redis 相比 Memcached 有哪些优势?	3
5. 如何实现本地缓存? 请描述一下你知道的方式.....	3
6. Redis 通讯协议是什么? 有什么特点?	3
二、Redis 数据结构与指令.....	3
1. Redis 支持的数据类型.....	4
2. Redis 常用的命令有哪些?	4
3. 一个字符串类型的值能存储最大容量是多少?	4
4. Redis 各个数据类型最大存储量分别是多少?.....	4
5. 请介绍一下 Redis 的数据类型 SortedSet (zset) 以及底层实现机制?	5
6. Redis 事务相关命令有哪些?	5
7. 什么是 Redis 事务? 原理是什么?	5
8. Redis 事务的注意点有哪些?	6
9. Redis 为什么不支持回滚?	6
10. 请介绍一下 Redis 的 Pipeline (管道), 以及使用场景.....	6
11. 请说明一下 Redis 的批量命令与 Pipeline 有什么不同?	6
12. 请介绍一下 Redis 的发布订阅功能.....	6
13. Redis 的链表数据结构的特征有哪些?	7
14. 请介绍一下 Redis 的 String 类型底层实现?	7
15. Redis 的 String 类型使用 SSD 方式实现的好处?	7
16. 设置键的生存时间和过期时间有哪些命令?	7
三、Redis 高并发处理策略.....	7
1. 为什么 Redis 需要把所有数据放到内存中?	8
2. Redis 是单线程的吗?	8
3. Redis 为什么设计成单线程的?	8
4. 什么是缓存穿透? 怎么解决?	8
5. 什么是缓存雪崩? 怎么解决?	9
6. 缓存的更新策略有几种? 分别有什么注意事项?	9
7. 请介绍几个可能导致 Redis 阻塞的原因.....	10
8. 怎么去发现 Redis 阻塞异常情况?	10
四、Redis 集群结构以及设计理念.....	11
1. Redis 集群架构模式有哪几种?	11
2. Redis 集群最大节点个数是多少?	11
3. Redis 集群的主从复制模型是怎样的?	11
4. 请介绍一下 Redis 集群实现方案.....	12
5. Redis 集群会有写操作丢失吗? 为什么?	13
6. Redis 慢查询是什么? 通过什么配置?	13
7. Redis 的慢查询修复经验有哪些? 怎么修复的?	13
8. 如何优化 Redis 服务的性能?	13
9. Redis 的主从复制模式有什么优缺点?.....	14
10. Redis sentinel (哨兵) 模式优缺点有哪些?	14

11. 如何设置 Redis 的最大连接数？查看 Redis 的最大连接数？查看 Redis 的当前连接数？	14
12. 介绍一些 Redis 常用的安全设置？	14
五、Redis 缓存管理与持久化机制.....	14
1. Redis 持久化机制有哪些？	14
2. Redis 持久化机制 AOF 和 RDB 有哪些不同之处？	15
3. 请介绍一下 RDB 持久化机制的优缺点.....	15
4. 请介绍一下 AOF 持久化机制的优缺点.....	16
5. 如果 AOF 文件的数据出现异常，Redis 服务怎么处理？	16
6. 常见的淘汰算法有哪些？	16
7. Redis 淘汰策略有哪些？	16
8. Redis 缓存失效策略有哪些？	17
9. Redis 如何做内存优化？	18
10. 什么是 bigkey？有什么影响？	18
11. 怎么发现 bigkey?.....	18
12. Redis 的内存消耗分类有哪些？内存统计使用什么命令？	18
13. 简单介绍一下 Redis 的内存管理方式有哪些？	18
14. 如何设置 Redis 的内存上限？有什么作用？	18
15. Redis 报内存不足怎么处理？	19
六、Redis 应用场景设计.....	19
1. Redis 适用场景有哪些？	19
2. Redis 常用的业务场景有哪些？	19
3. Redis 支持的 Java 客户端有哪些？简单说明一下特点。	19
4. 请简单描述一下 Jedis 的基本使用方法？	20
5. Jedis 连接池链接方法有什么优点？	20
6. 什么是分布式锁？有什么作用？	21
7. 分布式锁可以通过什么来实现？	21
8. 介绍一下分布式锁实现需要注意的事项？	21
9. Redis 怎么实现分布式锁？	21
10. 缓存命中率表示什么？	22
11. 怎么提高缓存命中率？	22
12. 请介绍一下 Spring 注解缓存.....	22

一、Redis 概念理解

1. 什么是 Redis?

Redis 全称为：Remote Dictionary Server（远程数据服务），是一个基于内存且支持持久化的高性能 key-value 数据库。

具备一下几个基本特征：

1. 多数据类型
2. 持久化机制
3. 主从同步

2. Redis 的特点有哪些？

1. Redis 本质上是一个 key-value 类型的数据库
2. 整个数据库都是在内存中进行操作，可定期刷新到磁盘进行持久化存储
3. 由于是在内存操作，读写能力非常好，每秒可以处理 10 万次读写操作
4. Redis 支持多种数据结构，提供了丰富的数据类型选择
5. Redis 同时支持数据备份，主从配置
6. Redis 的所有操作都是原子性的

3. Memcache 与 Redis 的区别都有哪些？

1. **存储方式不同：**Memcache 把数据全部存在内存之中，断电后会丢失。Redis 所有数据加载在内存，但也会持久化到磁盘，保证数据的持久性。
2. **支持数据类型不同：**Memcache 对数据类型支持相对简单，只支持 key-value 结构。Redis 有复杂的数据类型。
3. **底层模型不同：**底层实现方式以及客户端通信应用协议不一样。Redis 直接自己构建了 VM 机制。
4. **运行环境不同：**Redis 目前官方只支持 Linux 上运行。

4. Redis 相比 Memcached 有哪些优势？

1. Memcached 所有的值均是简单的字符串，Redis 作为其替代者，支持更为丰富的数据类型
2. Redis 的速度比 Memcached 快很多
3. Redis 可以持久化其数据

5. 如何实现本地缓存？请描述一下你知道的方式

1. 程序中定义内存数据结构来实现，比如说定义一个成员变量 Map 或者 List 均可以实现
2. 使用开源的缓存框架 Ehcache，Ehcache 封装了对于内存操作的功能
3. Guava Cache 是 Google 开源的工具集，提供了缓存的边界操作工具

6. Redis 通讯协议是什么？有什么特点？

Redis 的通信协议是 Redis Serialization Protocol，简称 RESP。

有如下特性：

1. 是二进制安全的
2. 在 TCP 层
3. 基于请求—响应的模式

还总结出了各大互联网公司 java 程序员面试涉及到的绝大部分面试题及答案做成了文档和学习笔记文件以及架构视频资料免费分享给大家（包括 Dubbo、Redis、Netty、zookeeper、Spring cloud、分布

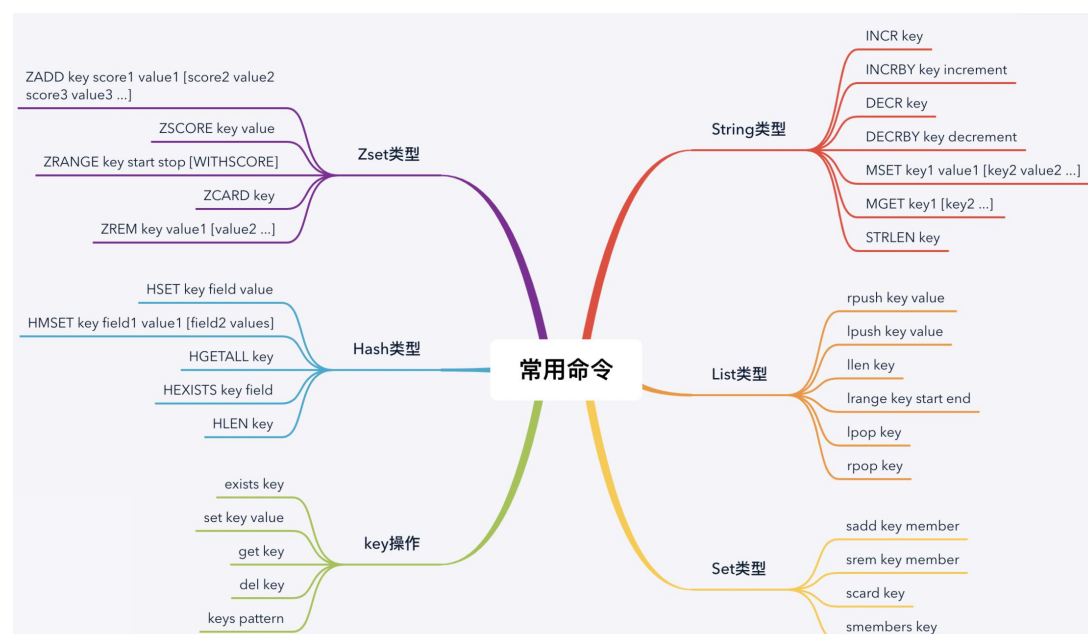
式、高并发等架构技术资料)，加 QQ 群：578486082 或加管理小姐姐 VX：rxh8515 免费获取！

二、Redis 数据结构与指令

1. Redis 支持的数据类型

1. String（字符串）
2. list（列表）：list 是字符串列表，按照插入顺序排序。元素可以在列表的头部（左边）或者尾部（右边）进行添加。
3. hash（哈希）：Redis hash 是一个键值对（key-value）集合。Redis hash 是一个 String 类型的 field 和 value 的映射表，hash 特别适合用于存储对象。
4. set（集合）：Redis 的 set 是 String 类型的无序集合。
5. zset（sorted set：有序集合）：Redis zset 和 set 一样也是 String 类型元素的集合，且不允许重复的成员。不同的 zset 是每个元素都会关联一个 double 类型的分数。zset 通过这个分数来为集合中所有元素进行从小到大的排序。zset 的成员是唯一的，但分数（score）却可以重复。

2. Redis 常用的命令有哪些？



3. 一个字符串类型的值能存储最大容量是多少？

512M

4. Redis 各个数据类型最大存储量分别是多少？

1. Strings 类型：一个 String 类型的 value 最大可以存储 512M

2. **Lists 类型**: list 的元素个数最多为 $2^{32}-1$ 个, 也就是 4294967295 个。
3. **Sets 类型**: 元素个数最多为 $2^{32}-1$ 个, 也就是 4294967295 个。
4. **Hashes 类型**: 键值对个数最多为 $2^{32}-1$ 个, 也就是 4294967295 个。
5. **Sorted sets 类型**: 跟 Sets 类型相似。

5. 请介绍一下 Redis 的数据类型 SortedSet (zset) 以及底层实现机制?

zset 有顺序, 不能重复。在业务场景下, 适合做排行榜之类的事情。

底层实现机制:

SortedSet 的实现方式可能有两种: **ziplist 或者 skiplist**。

当有序集合对象同时满足以下两个条件时, 对象使用 ziplist 编码:

1. 保存的元素数量小于 128;
2. 保存的所有元素长度都小于 64 字节。 不能满足上面两个条件的使用 skiplist 编码。
3. ziplist 编码的有序集合对象使用压缩列表作为底层实现, 每个集合元素使用两个紧挨在一起的压缩列表节点来保存, 第一个节点保存元素的成员, 第二个节点保存元素的分值。并且压缩列表内的集合元素按分值从小到大的顺序进行排列, 小的放置在靠近表头的位置, 大的放置在靠近表尾的位置。
4. skiplist 编码的有序集合对象使用 zset 结构作为底层实现, 一个 zset 结构同时包含一个字典和一个跳跃表。字典的键保存元素的值, 字典的值则保存元素的分值; 跳跃表节点的 object 属性保存元素的成员, 跳跃表节点的 score 属性保存元素的分值。

6. Redis 事务相关命令有哪些?

1. **discard 命令**: 取消事务, 丢弃事务中所有命令。
2. **exec 命令**: 执行所有事务内的命令。
3. **multi 命令**: 标记一个事务开始。
4. **unwatch 命令**: 取消 watch 命令对所有 key 的监视。
5. **watch 命令**: 监视一个 (或多个) key, 如果在执行事务之前这个 (这些) key 被其他命令所改动, 事务将被打断。

7. 什么是 Redis 事务? 原理是什么?

Redis 中的事务是一组命令的集合, 是 Redis 的最小执行单位, 一个事务要么都执行, 要么都不执行。

Redis 事务保证一个事务内的命令依次执行, 而不会被其他命令插入。

Redis 事务的原理是先将属于一个事务的命令发送给 Redis, 然后依次执行这些命令。

8. Redis 事务的注意点有哪些？

1. 不支持回滚，如果事务中有错误的操作，无法回滚到处理前的状态，需要开发者处理。
2. 在执行完当前事务内所有指令前，不会同时执行其他客户端的请求。

9. Redis 为什么不支持回滚？

Redis 事务不支持回滚，如果遇到问题，会继续执行余下的命令。这一点和关系型数据库不太一致。这样处理的原因有：

1. 只有语法错误，Redis 才会执行失败，例如错误类型的赋值，这就是说从程序层面完全可以捕获以及解决这些问题
2. 支持回滚需要增加很多工作，不支持的情况下，Redis 可以保持简单、速度快的特性

10. 请介绍一下 Redis 的 Pipeline（管道），以及使用场景

Redis 客户端与服务端通信模型使用的 TCP 协议进行连接，那么在单个指令的执行过程中，都会存在“交互往返”的时间。

Redis 提供了批量操作命令，例如 `mget`、`mset` 等，能够一定程度上节省这类时间，但大部分命令还是不支持批量操作。

因此 Pipeline 功能，能够改善这一类问题。Pipeline 将一组 Redis 命令进行组装，一次性传输给 Redis，再将这些命令执行结果，按照顺序返回给客户端。

适用场景：有批量的数据写入 Redis，并且这些数据允许一定比例的写入失败，那么这种场景就可以适用 Pipeline。失败的数据后期进行补偿即可。

11. 请说明一下 Redis 的批量命令与 Pipeline 有什么不同？

1. 批量命令保证原子性的，Pipeline 非原子性
2. 批量命令是一个命令对应多个 key，Pipeline 支持多个命令
3. 批量命令是 Redis 服务端实现，而 Pipeline 是需要服务端和客户端共同实现

12. 请介绍一下 Redis 的发布订阅功能

Redis 提供了基于“发布/订阅”模式的消息机制，消息发布者和订阅者不能直接通信，客户端发布消息的时候指定发送的频道，然后订阅了该频道的用户可以接收到该消息。具体指令如下：

1. 发布消息：`publish channel message`
2. 订阅消息：`subscribe channel [.....]`
3. 退订消息：`punsubscribe`

13. Redis 的链表数据结构的特征有哪些？

1. **双端引用**：链表的最前和最后节点都有引用，获取前后节点的复杂度为 $O(1)$
2. **无环链表**：对于链表的访问都是以 `null` 结束
3. **长度计数器**：通过 `len` 属性来记录链表长度

14. 请介绍一下 Redis 的 String 类型底层实现？

Redis 底层实现了简单动态字符串的类型（SSD），来表示 String 类型。没有直接使用 C 语言定义的字符串类型。

```
struct sdshdr{  
  
    //记录 buf 数组中已使用字节的数量  
  
    //等于 SDS 保存字符串的长度  
  
    int len;  
  
    //记录 buf 数组中未使用字节的数量  
  
    int free;  
  
    //字节数组，用于保存字符串  
  
    char buf[];  
  
}
```

15. Redis 的 String 类型使用 SSD 方式实现的好处？

1. **避免缓冲区溢出**：进行字符修改时候，可以根据 `len` 属性来检查空间是否满足要求
2. **减少内存分配次数**：`len` 和 `free` 两个属性，可以协助进行空间预分配以及惰性空间释放
3. **二进制安全**：SSD 不是以空字符串来判断是否结束，而是以 `len` 属性来判断字符串是否结束
4. **常数级别获取字符串长度**：获取字符串的长度只需要读取 `len` 属性就可以获取
5. **兼容 C 字符串函数**：可以重用 C 语言库的一部分函数

16. 设置键的生存时间和过期时间有哪些命令？

`EXPIRE` 以秒为单位，设置键的生存时间
`PEXPIRE` 以毫秒为单位，设置键的生存时间
`EXPIREAT` 以秒为单位，设置键的过期 UNIX 时间戳
`PEXPIREAT` 以毫秒为单位，设置键的过期 UNIX 时间戳

还总结出了各大互联网公司 java 程序员面试涉及到的绝大部分面试题及答案做成了文档和

三、Redis 高并发处理策略

1. 为什么 Redis 需要把所有数据放到内存中？

追求更快的读写速度，并异步方式持久化存储到磁盘。如果不将数据放到内存中，磁盘的 I/O 速度会严重影响 Redis 的性能。

2. Redis 是单线程的吗？

是。

这里的单线程指的是 Redis 网络请求模块使用了一个线程（所以不需考虑并发安全性），即一个线程处理所有网络请求，其他模块仍用了多个线程。

3. Redis 为什么设计成单线程的？

1. 绝大部分请求是纯粹的内存操作（非常快速）
2. 采用单线程，避免了不必要的上下文切换和竞争条件
3. 非阻塞 IO，内部采用 epoll，epoll 中的读、写、关闭、连接都转化成了事件，然后利用 epoll 的多路复用特性，避免 IO 代价。

4. 什么是缓存穿透？怎么解决？

缓存穿透是指缓存中查询一个不存在的数据，需要去数据库中获取。

如果数据库也查不到结果，将不会同步到缓存，导致这个不存在数据每次请求都要到数据库查询，失去了缓存的意义。

解决方法有两个：

1. 布隆过滤 (Bloom filter)

将所有查询的参数都存储到一个 bitmap 中，在查询缓存之前，先再找个 bitmap 里面进行验证。

如果 bitmap 中存在，则进行底层缓存的数据查询；如果 bitmap 中不存在查询参数，则进行拦截，不再进行缓存的数据查询。

适用范围：可以用来实现数据字典，进行数据的判重，或者集合求交集

2. 缓存空对象

如果查询返回的数据为空，仍然把这个空结果进行缓存。那么再次用相同 key 获取数据的时候，即使不存在的数据，缓存也可以直接返回空值，避免重复访问 DB。

缓存空对象有两个不足之处：

1. 缓存层将存储更多的键值对，如果是恶意的随机访问，将造成很多内存空间的浪费。这个不足之处可以通过将这类数据设置很短的过期时间来控制。
2. DB 与缓存数据不一致。这种可以考虑通过异步消息来进行数据更新的通知，在一定程度上减少这类不一致的时间。

5. 什么是缓存雪崩？ 怎么解决？

在集中的一段时间内，有大量的缓存失效，导致大量的访问没有命中缓存，从而将所有查询进行数据库访问，导致数据库的压力增大，从而造成了缓存雪崩。

比如，如果要做一个促销活动，我们将商品信息都刷新到缓存，过期时间统一为 30 分钟。那么在 30 分钟之后，这批商品将全部过期。这时候这批商品的访问查询，都落到了数据库，对于数据库而言，这一刻的压力会非常大。从而造成系统整体性风险。

解决方案：

方法 1：分散失效时间

分析缓存数据的特点，尽量将热点缓存的失效时间均匀分布。比如说将相同类型的缓存的失效时间设置成一个在一定区间内的随机值。从而有效的分散失效时间。

方法 2：DB 访问限制

对数据的访问进行限流性质的操作。比如说对数据库访问进行加锁的处理或者限流相关的处理。

方法 3：多级缓存设计

一级缓存为基础缓存，缓存失效时间设置一个较长时间，二级缓存为应用缓存，失效时间正常设置，一般会比较短。当二级缓存失效的时候，再从一级缓存里面获取。

6. 缓存的更新策略有几种？分别有什么注意事项？

缓存的更新策略包含：

1. 先更新数据库，再更新缓存
2. 先删除缓存，再更新数据库
3. 先更新数据库，再删除缓存

策略一：先更新数据库，再更新缓存

1. 这种策略会导致线程安全问题

例如：线程 1 更新了数据库，线程 2 也更新数据库，这时候由于某种原因，线程 2 首先更新了缓存，线程 1 后续更新。这样就导致了脏数据的问题。因为目前数据库中存储的线程 2 更新后的数据，而缓存存储的是线程 1 更新的老数据。

2. 更新缓存的复杂度相对较高

数据写入数据库之后，一般存入缓存的数据都要经过一系列的加工计算，然后写入缓存。这时候更新缓存相比较于直接删除缓存要比较复杂。

策略二：先删除缓存，再更新数据库

这种策略可能导致数据不一致的问题。线程 1 写数据删除缓存；这时候有线程 2 查询该缓存，发现不存在，则去访问数据库，得到旧值放入缓存；线程 1 更新数据库。这时候就出现了数据不一致的问题。如果缓存没有过期时间，这个脏数据一直存在。

解决方案：在写数据库成功之后，再次淘汰缓存一次。

策略三：先更新数据库，再删除缓存

可能会造成比较短暂的数据不一致。在更新完成数据库，还没有删除缓存的时刻，如果有缓存数据访问，就会造成数据不一致的情形。但这种如果数据同步机制比较科学，一般都会比较快，不一致的影响比较小。

7. 请介绍几个可能导致 Redis 阻塞的原因

内部原因：

1. Redis 的 API 或者指令数据结构使用不合理
2. Redis 主机 CPU 负载过高，导致系统崩溃
3. 持久化工作资源占用过多

外部原因：

1. CPU 竞争
2. 内存交换
3. 网络问题

8. 怎么去发现 Redis 阻塞异常情况？

1. 通过应用服务监控发现

当 Redis 阻塞的时候，线上应用服务应该会感知发现。比如说发现 Redis 链接超时等。这种就需要应用服务增加对于异常的统计，并针对 Redis 相关的异常，进行报警。

2. 通过 Redis 自身监控系统

借助 Redis 监控系统发现阻塞问题，当监控系统发现各个监控指标存在异常的时候，发送报警。指标有：CPU/内存/磁盘等，慢查询，命令耗时增加等。

四、Redis 集群结构以及设计理念

1. Redis 集群架构模式有哪几种？

1. Redis 单节点单机器部署
2. Redis 主从节点部署
3. Redis Sentinel（哨兵）模式部署
4. Redis 集群模式

2. Redis 集群最大节点个数是多少？

16384 个

3. Redis 集群的主从复制模型是怎样的？

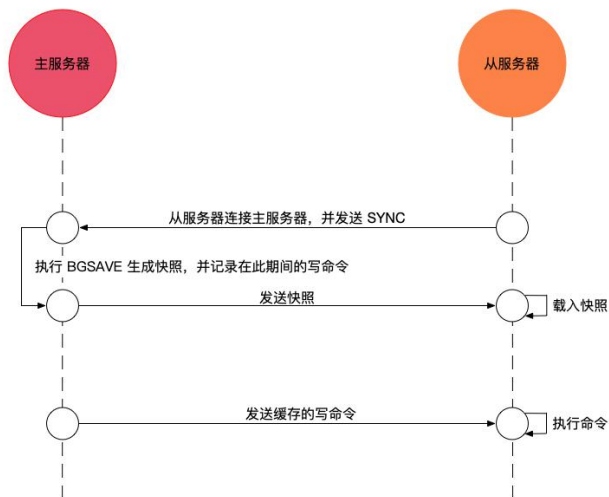
Redis 中的主从复制，也就是 Master-Slave 模型，多个 Redis 实例间的数据同步以及 Redis 集群中数据同步会使用主从复制。

主从复制主要是数据同步，数据同步分为**全量同步**和**增量同步**。

全量同步：

全量同步一般发生在 Slave 机器初始化阶段，这时候 Slave 需要将 Master 上的所有数据都进行同步复制。

大概步骤如下所示：



1. 从服务器发送 SYNC 命令，链接主服务器
2. 主服务器收到 SYNC 命令后，进行存盘的操作，并继续收集后续的写命令，存储缓冲区
3. 存盘结束后，将对应的数据文件发送到 Slave 中，完成一次全量同步
4. 主服务数据发送完毕后，将进行增量的缓冲区数据同步
5. Slave 加载数据文件和缓冲区数据，开始接受命令请求，提供操作

增量更新：

Slave 节点初始化完成之后，开始正常工作，Master 节点进行的写操作都会同步到 Slave 节点上面。Master 节点每执行一个写命令都会向从服务器发送相同的写命令，从服务器接收到命令，并执行。

4. 请介绍一下 Redis 集群实现方案

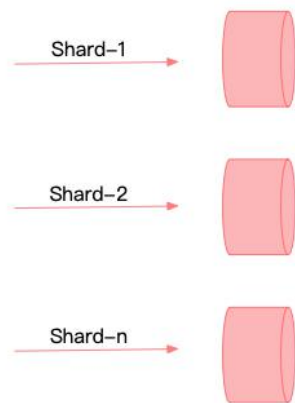
1. Redis Cluster 集群方案（服务端 Sharding 技术）

Redis Cluster 是 3.0 版本开始正式提供的服务器 Sharding 技术。引入 slot（槽）的概念，整个集群共有 16384 个槽。根据 key 进行散列（CRC16 后 16384 取模），分配到其中一个槽当中。

2. Redis Sharding 集群（客户端 Sharding 技术）

Redis Sharding 出现先与 Redis Cluster 方案，他是多 Redis 实例集群方案。

客户端 Sharding 方式，该向哪个 Redis 节点操作数据，由客户端来进行控制。服务端 Redis 还是一个个相对独立的 Redis 实例节点，没有做任何变动。节点选择可采用的方法：一致性 hash 算法或者虚拟节点算法。



3. Twemproxy 中间件实现

Twemproxy 就是一种中间件 Sharding 分片的技术，处于客户端和服务器的中间，将客户端发来的请求，进行一定的处理后（如 Sharding），再转发给后端真正的 Redis 服务器。

也就是说，客户端不直接访问 Redis 服务器，而是通过 Twemproxy 代理中间件间接访问。

5. Redis 集群会有写操作丢失吗？为什么？

Redis 并不能保证数据的强一致性，这意味这在实际中集群在特定的条件下可能会丢失写操作。

6. Redis 慢查询是什么？通过什么配置？

所谓慢查询就是指，系统执行命令之后，计算统计每条指令执行的时间，如果执行时间超过设置的阈值，就说明该命令为慢指令。

Redis 慢查询配置参数为：

1. **slowlog-log-slower-than**: 设置慢查询定义阈值，超过这个阈值就是慢查询。单位为微妙，默认为 10000。
2. **slowlog-max-len**: 慢查询的日志列表的最大长度，当慢查询日志列表处于最大长度的时候，最早插入的一个命令将从列表中移除。

7. Redis 的慢查询修复经验有哪些？怎么修复的？

1. 将一些效率比较低或者算法复杂度比较高的命令，禁用或者替换为效率较高的指令。
例如禁用：sort、keys 命令
2. 拆分大对象数据，防止一次命令操作过多的数据

8. 如何优化 Redis 服务的性能？

1. Master 节点禁止持久化工作

2. 持久化策略要有正确的选择，关键数据可以采用 slave 节点 AOF 备份
3. 主从节点部署在同一个局域网内，保证复制速度与稳定性
4. 设置或者增加从库需要考虑主库现有的压力
5. 主从复制一定要单向结构，避免使用图状结构

9. Redis 的主从复制模式有什么优缺点？

- **优点：**可靠性相对于单个节点部署模式有所提高，实现读写分离提高读写效率
- **缺点：**写数据依赖于主节点，主节点空间有限，而且主节点存在单点的风险

10. Redis sentinel（哨兵）模式优缺点有哪些？

- **优点：**保证高可用，各个节点自动故障转移
- **缺点：**主从模式，依旧存在主节点单点风险，主从切换有丢失数据的问题

11. 如何设置 Redis 的最大连接数？查看 Redis 的最大连接数？查看 Redis 的当前连接数？

- **设置 Redis 的最大连接数使用命令：**`redis-server --maxclients 50000`
- **查看 Redis 最大连接数使用命令：**`config get maxclients`
- **查看 Redis 连接数：**使用 `info` 命令；在 `redis-cli` 端输入 `info` 命令即可查看

12. 介绍一些 Redis 常用的安全设置？

1. 网络安全

除了信任的客户端发出的请求以外，其他所有请求都拒绝。对于暴露到外网的服务，使用防火墙阻止外部访问 Redis 端口。

2. 身份验证

Redis 提供了简单的身份验证功能，在 `redis.conf` 文件中进行配置生效。客户端可以通过发送（`AUTH 密码`）命令进行身份认证。

3. 禁用特定的命令集

Redis 可以选择禁止使用某些命令，这样即使是正常的客户端也无法使用这些命令集合。

五、Redis 缓存管理与持久化机制

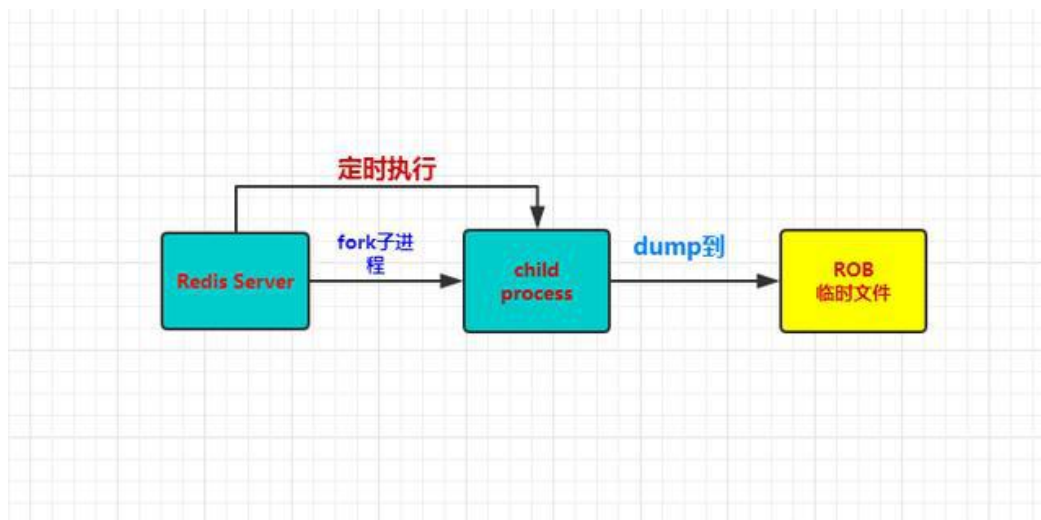
1. Redis 持久化机制有哪些？

Redis 提供两种方式进行持久化。

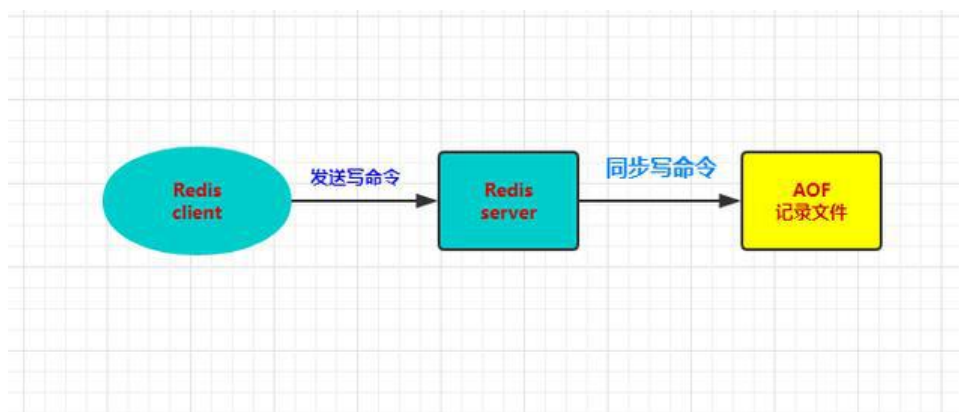
1. **RDB 持久化**：原理是将 Redis 在内存中的数据库记录定时 dump 到磁盘上的 RDB 持久化。
2. **AOF (append only file) 持久化**：原理是将 Redis 的操作日志以追加的方式写入文件。

2. Redis 持久化机制 AOF 和 RDB 有哪些不同之处？

两者的区别：RDB 持久化是指在指定的时间间隔内将内存中的数据集快照写入磁盘，实际操作过程是 fork 一个子进程，先将数据集写入临时文件，写入成功后，再替换之前的文件，用二进制压缩存储。



AOF 持久化以日志的形式记录服务器所处理的每一个写、删除操作，查询操作不会记录，以文本的方式记录，可以打开文件看到详细的操作记录。



3. 请介绍一下 RDB 持久化机制的优缺点

优点

1. RDB 是紧凑的二进制文件，比较适合备份，全量复制等场景
2. RDB 恢复数据远快于 AOF

缺点

1. RDB 无法实现实时或者秒级持久化；
2. 新老版本无法兼容 RDB 格式。

4. 请介绍一下 AOF 持久化机制的优缺点

优点

1. 可以更好地保护数据不丢失；
2. append-only 模式写入性能比较高；
3. 适合做灾难性的误删除紧急恢复。

缺点：

1. 对于同一份文件，AOF 文件要比 RDB 快照大；
2. AOF 开启后，会对写的 QPS 有所影响，相对于 RDB 来说 写 QPS 要下降；
3. 数据库恢复比较慢， 不合适做冷备。

5. 如果 AOF 文件的数据出现异常， Redis 服务怎么处理？

如果 AOF 文件出现异常， Redis 在重启的时候将会拒绝加载 AOF 文件，从而保证数据的一致性。

这时候，可以试着对 AOF 文件进行修复：`redis-check-aof -fix`。

6. 常见的淘汰算法有哪些？

FIFO: First In First Out

先进先出。判断被存储的时间，离目前最远的数据优先被淘汰。

LRU: Least Recently Used

最近最少使用。判断最近被使用的时间，目前最远的数据优先被淘汰。

LFU: Least Frequently Used

最不经常使用。在一段时间内，数据被使用次数最少的，优先被淘汰。

7. Redis 淘汰策略有哪些？

Redis 可以看作是一个内存数据库,通过 Maxmemory 指令配置 Redis 的数据集使用指定量的内存。设置 Maxmemory 为 0,则表示无限制。

当内存使用达到 Maxmemory 极限时,需要使用某种淘汰算法来决定清理掉哪些数据,以保证新数据的存入。

Redis 的缓存淘汰策略有:

1. **noeviction:** 当内存使用达到上限,所有需要申请内存的命令都会异常报错。
2. **allkeys-lru:** 先试图移除一部分最近未使用的 key。
3. **volatile-lru:** 淘汰一部分最近使用较少的 (LRC), 但只限于过期设置键。
4. **allkeys-random:** 随机淘汰某一个键。
5. **volatile-random:** 淘汰任意键, 但只限于有过期设置的驱逐键。
6. **volatile-ttl:** 优先移除具有更早失效时间的 key。

8. Redis 缓存失效策略有哪些?

策略有: 定时删除策略, 惰性删除策略, 定期删除策略。

定时删除策略

在设置 key 的过期时间的同时,为该 key 创建一个定时器,让定时器在 key 的过期时间来临时,对 key 进行删除。

- 优点: 保证内存尽快释放。
- 缺点: 若 key 过多,删除这些 key 会占用很多 CPU 时间, 而且每个 key 创建一个定时器,性能影响严重。

惰性删除策略

key 过期的时候不删除,每次从数据库获取 key 的时候去检查是否过期,若过期,则删除,返回 null。

- 优点: CPU 时间占用比较少。
- 缺点: 若 key 很长时间没有被获取, 将不会被删除,可能造成内存泄露。

定期删除策略

每隔一段时间执行一次删除(在 redis.conf 配置文件设置 hz, 1s 刷新的频率)过期 key 操作。

- 优点: 可以控制删除操作的时长和频率,来减少 CPU 时间占用,可以避免惰性删除时候内存泄漏的问题。
- 缺点: 对内存友好方面,不如定时策略;对 CPU 友好方面,不如惰性策略

Redis 一般采用: 惰性策略 + 定期策略两个相结合。

9. Redis 如何做内存优化？

- Redis 的所有数据都进行 redisObject 来封装
- 缩减键、值对象的长度，简化键名，提高内存使用效率
- 尽量使用 hash 数据结构，减少 key 的数量
- 共享对象池，并设置空间极限值

10. 什么是 bigkey？有什么影响？

bigkey 是指存储 value 占用内存空间非常大的 key。例如一个 String 类型的 value 占用了 100MB 空间。

bigkey 的影响主要体现在：

1. **造成内存空间不平衡：**如果 bigkey 存储量比较大，同一个 key 在同一个节点或者服务器中存储，造成一定的影响
2. **超时阻塞：**由于占用空间比较大，那么操作起来效率肯定比较低，也就表示出现阻塞可能性增加
3. **网络阻塞：**获取 bigkey 的时候，自然传输的数据量比较大，导致宽带的压力。

11. 怎么发现 bigkey？

redis-cli-bigkeys 命令可以统计 bigkey 的分布。

生产环境下执行 debug object key 查看 serializedlength 属性，表示 key 对应的 value 序列化之后的字节数。

12. Redis 的内存消耗分类有哪些？内存统计使用什么命令？

Redis 的内存消耗分为：

1. **对象内存：**这是占用最大的一块，存储用户的所有数据。可以理解为：所有 key 的大小 + 所有 value 的大小
2. **缓冲内存：**客户端缓冲、复制积压缓冲、AOF 缓冲
3. **内存碎片：**更新操作、过期数据都可能造成内存碎片

13. 简单介绍一下 Redis 的内存管理方式有哪些？

Redis 内存管理方式主要方向上有两个：

1. 控制内存上限
2. 优化回收策略，进行内存回收

14. 如何设置 Redis 的内存上限？有什么作用？

Maxmemory 参数可以限制最大可用内存。

15. Redis 报内存不足怎么处理？

1. 修改配置文件，增加 Redis 的内存空间 maxmemory
2. 设置缓存淘汰策略，具体的淘汰策略如（Redis 缓存管理与持久化机制）题 7 所示
3. 使用 Redis 集群模式来存储

六、Redis 应用场景设计

1. Redis 适用场景有哪些？

适用场景：

- 数据（热点）高并发的读写
- 海量数据的读写
- 对扩展性要求高的数据

不适场景：

- 需要事务支持（非关系型数据库）
- 基于 SQL 结构化查询储存，关系复杂

2. Redis 常用的业务场景有哪些？

1. **热点数据缓存：**由于 Redis 访问速度快、支持的数据类型比较丰富，所以 Redis 很适合用来存储热点数据
2. **限时业务实现：**expire 命令设置 key 的生存时间，到时间后自动删除 key。收集验证码、优惠活动等业务场景。
3. **计数器实现：**incrby 命令可以实现原子性的递增，所以可以运用于高并发的秒杀活动、分布式序列号的生成。比如限制一个手机号发多少条短信、一个接口一分钟限制多少请求、一个接口一天限制调用多少次等等。
4. **排行榜实现：**借助 SortedSet 进行热点数据的排序。例如：下单量最多的用户排行榜，最热门的帖子（回复最多）等。
5. **布式锁实现：**利用 Redis 的 setnx 命令进行。后面会有详细的实现介绍。
6. **队列机制实现：**Redis 有 list push 和 list pop 这样的命令，所以能够很方便的执行队列操作。

3. Redis 支持的 Java 客户端有哪些？简单说明一下特点。

官方推荐的有三种：Jedis、Redisson 和 lettuce。

Jedis:

1. 轻量，简洁，便于集成和改造
2. 支持连接池
3. 支持 pipelining、事务、Lua Scripting、Redis Sentinel、Redis Cluster
4. 不支持读写分离，需要自己实现

Redisson:

1. 基于 Netty 实现，采用非阻塞 IO，性能高
2. 支持异步请求
3. 支持连接池
4. 支持 pipelining、Lua Scripting、Redis Sentinel、Redis Cluster
5. 不支持事务
6. 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
7. 内建 Tomcat Session Manager，为 Tomcat 6/7/8 提供了会话共享功能
8. 可以与 Spring Session 集成，实现基于 Redis 的会话共享
9. 文档较丰富，有中文文档

4. 请简单描述一下 Jedis 的基本使用方法？

```
Jedis jedis = new Jedis("127.0.0.1", 6379, 500, 500);
```

```
jedis.set("hello", "world");
```

```
String value = jedis.get("hello");
```

上面代码就是一个简单的 Redis 数据存储操作。

初始化 Jedis 需要四个参数：

```
Jedis(final String host, final int port, final int connectionTimeout, final int soTimeout)
```

1. Host: Redis 节点的服务 IP
2. Port: Redis 服务的端口
3. connectionTimeout: 客户端连接超时时间
4. soTimeout: 客户端读写超时时间

5. Jedis 连接池链接方法有什么优点？

上一个题目介绍了 Jedis 的直连方式，每次操作 Redis 都会创建一个新的 TCP 连接，使用完之后就会断开。很明显消耗了不必要的资源。

为了减少这方面的消耗，Jedis 提供了连接池的链接方式：JedisPool。每次连接 Redis 只需要池子中拿，用完了归还就可以了。

1. 无需每次连接都要生成一个 Jedis 对象
2. 使用连接池保护和控制资源的使用

6. 什么是分布式锁？有什么作用？

分布式锁是控制分布式系统之间同步访问共享资源的一种方式。在单机或者单进程环境下，多线程并发的情况下，使用锁来保证一个代码块在同一时间内只能由一个线程执行。比如 Java 的 Synchronized 关键字和 Reentrantlock 类。

多进程或者分布式集群环境下，如何保证不同节点的线程同步执行呢？这就是分布式锁。

7. 分布式锁可以通过什么来实现？

1. Memcached 分布式锁

Memcached 提供了原子性操作命令 add，才能 add 成功，线程获取到锁。key 已存在的情况下，则 add 失败，获取锁也失败。

2. Redis 分布式锁

Redis 的 setnx 命令为原子性操作命令。只有在 key 不存在的情况下，才能 set 成功。和 Memcached 的 add 方法比较类似。

3. ZooKeeper 分布式锁

利用 ZooKeeper 的顺序临时节点，来实现分布式锁和等待队列。

4. Chubby 实现分布式锁

Chubby 底层利用了 Paxos 一致性算法，实现粗粒度分布式锁服务。

8. 介绍一下分布式锁实现需要注意的事项？

分布式锁实现要保证几个基本点。

1. 互斥性：任意时刻，只有一个资源能够获取到锁。
2. 容灾性：在未成功释放锁的情况下，一定时限内能够恢复锁的正常功能。
3. 统一性：加锁和解锁保证同一资源来进行操作。

9. Redis 怎么实现分布式锁？

简单方案：

最简单的方法是使用 setnx 命令。释放锁的最简单方式是执行 del 指令。

问题：

锁超时：如果一个得到锁的线程在执行任务的过程中挂掉，来不及显式地释放锁，这块资源将会永远被锁住（死锁），别的线程再也别想进来。

优化方案：

setnx 没办法设置超时时间，如果利用 expire 来设置超时时间，那么这两步操作不是原子性操作。

利用 set 指令增加了可选参数方式来替代 setnx。set 指令可以设置超时时间。

10. 缓存命中率表示什么？

- **缓存命中：** 可以从缓存中获取到需要的数据
- **缓存不命中：** 缓存中无法获取所需数据，需要再次查询数据库或者其他数据存储载体。

缓存命中率 = 缓存中获取数据次数 / 获取数据总次数

通常来说，缓存命中率越高，缓存的收益越高，应用的性能也就越好。

11. 怎么提高缓存命中率？

通常的手段有：

1. 缓存预加载
2. 增加缓存存储量
3. 调整缓存存储数据类型
4. 提升缓存更新频次

12. 请介绍一下 Spring 注解缓存

Spring 3.1 版本之后，加入了注解缓存操作技术，对缓存使用进行了具体的定义操作，添加了部分自定义的 annotation，能够非常便捷的获取缓存对象以及释放缓存对象。

常用的注解：

1. **@Cacheable：** 针对方法配置，根据方法的请求参数对其结果进行缓存
2. **@CachePut：** 针对方法配置，能够根据方法的请求参数对其结果进行缓存，和 @Cacheable 不同的是，它每次都会触发真实方法的调用
3. **@CacheEvict：** 主要针对方法配置，能够根据一定的条件对缓存进行清空

读者分享





加管理小姐姐 vx: mf97532 或扫描二维码，可以免费领取
验证消息回答：“Java 资料”否在不予通过好友，后续资料也会不断更新














Java 架构进阶资料展示



 docker学习思维笔记 XMind Workbook 150 KB	 Git基础学习笔记 XMind Workbook 12.4 KB	 Java并发体系学习思维笔记 XMind Workbook 327 KB
 java基础笔记 XMind Workbook 5.32 KB	 JVM和性能优化学习思维笔记 XMind Workbook 309 KB	 JVM知识点 XMind Workbook 32.5 KB
 kafka知识导图笔记 XMind Workbook 10.2 KB	 mybatis学习笔记 XMind Workbook 1.32 MB	 MySQL优化学习思维笔记 XMind Workbook 301 KB
 Redis设计与实现 XMind Workbook 55.6 KB	 springboot学习思维笔记 XMind Workbook 225 KB	 SpringCloud学习笔记 XMind Workbook 14.4 KB
 Spring学习思维笔记 XMind Workbook 425 KB	 设计模式学习笔记 XMind Workbook 1.09 MB	 算法和数据结构 XMind Workbook 14.4 KB

有面试复习资料还有整理了面试高频问题的视频解析和大咖架构进阶笔记

 【Mark老师】面试必备—服务器推送技术.mp4	2019-05-11 12:00	972.72MB
 【James老师】面试必备—轻松搞定AOP面试从Spring热插件实战开始.m...	2019-05-11 12:00	793.46MB
 【James老师】面试必备—快速搞定RabbitMq中间件.mp4	2019-05-11 12:00	786.27MB
 【James老师】面试必备—手写SpringMVC框架，从害怕到喜欢只需1小...	2019-05-11 12:00	771.43MB
 【Peter老师】面试必备—API接口安全.mp4	2019-05-11 13:55	756.11MB
 【King老师】面试必备—JVM性能优化.mp4	2019-05-11 11:59	633.64MB
 【King老师】面试必备—深入理解JVM.mp4	2019-05-11 11:59	601.06MB
 【Lison老师】面试必备—匠心独运手写MyBatis框架.mp4	2019-05-11 11:59	577.78MB
 【Deer老师】面试必备—zk分布式锁.mp4	2019-05-11 11:58	554.66MB
 【Lison老师】面试必备—Spring AOP源码讲解.mp4	2019-05-11 11:58	513.67MB
 【Deer老师】面试必备—Redis进阶问题讲解.mp4	2019-05-11 11:59	487.04MB

	Dubbo服务框架知识点PDF文档整理
	Java架构进阶150道面试题PDF文档整理
	Java架构师面试技术点整理及学习笔记
	Java面试高频试题汇总（整理答案适合1-5年开发）
	Java面试知识点体系PDF文档整理
	JVM及性能优化知识点笔记PDF文档整理
	Redis知识点笔记PDF文档整理
	Spring全家桶知识点PDF文档整理
	分布式架构知识点笔记PDF文档整理