Redis高频面试题（精简版）

Redis支持哪些数据结构？

Redis支持以下主要数据结构：

- **String**：字符串类型，可存储文本或二进制数据
- **Hash**：字段-值映射，适合存储对象
- **List**：有序字符串元素集合，可重复
- **Set**：无序字符串元素集合，不可重复
- **ZSet**：有序集合，每个元素有一个分数(score)用于排序
- **高级数据结构**：Bitmap、HyperLogLog、Geo、Stream等

Redis的持久化机制有哪些？RDB和AOF的区别是什么？

Redis提供两种持久化机制：

- **RDB（快照）**：按指定时间间隔生成内存数据的快照文件(.rdb)
- **AOF（追加文件）**：记录每一条写命令到日志文件(.aof)

区别：

- **性能**：RDB对性能影响小，AOF可能影响性能
- **数据安全**：AOF更安全，可配置刷盘策略；RDB可能丢失最后一次快照后的所有数据
- **文件大小**：RDB文件小，AOF文件较大
- **恢复速度**：RDB恢复快，AOF恢复慢

Redis的过期键删除策略是什么？为什么采用这种策略？

Redis采用**"定期删除+惰性删除"**的混合策略：

- **定期删除**：每隔一段时间检查部分过期键并删除
- **惰性删除**：访问键时才检查是否过期

原因：单一策略有缺点

- **定时删除**：CPU开销大
- **惰性删除**：可能浪费内存
- **混合策略**：平衡了CPU和内存开销

Redis的内存淘汰策略有哪些？如何选择？

当内存达到`maxmemory`限制时，Redis会触发淘汰策略：

- **allkeys-lru**：淘汰所有键中最近最少使用的
- **volatile-lru**：淘汰设置了过期时间的键中最近最少使用的
- **allkeys-random**：随机淘汰所有键
- **volatile-random**：随机淘汰设置了过期时间的键
- **volatile-ttl**：淘汰设置了过期时间的键中剩余时间最短的
- **noeviction**：不淘汰任何键，写入操作返回错误

选择建议：

- 缓存场景常用`allkeys-lru`
- 有部分键需长期保留时用`volatile-lru`
- 数据访问分布均匀时可用`random`策略

Redis如何实现分布式锁？有什么注意事项？

实现方式：使用`SET key value NX EX timeout`命令

- **NX**：确保只有一个客户端能加锁
- **EX**：自动释放锁，避免死锁

注意事项：

- 锁的过期时间需大于业务执行时间
- 释放锁需使用Lua脚本确保原子性
- 考虑锁的可重入性
- Redis集群下可能存在锁丢失问题

Redis主从复制的原理是什么？有什么作用？

原理：从节点连接主节点，先进行全量复制，再通过增量复制保持数据一致

作用：

- 读写分离，提高读性能
- 数据备份，提高数据安全性
- 故障转移，提高系统可用性

Redis哨兵模式的工作原理是什么？

哨兵模式通过哨兵进程监控主从节点健康状态：

- **监控**：定期向所有节点发送PING命令检查健康状态
- **通知**：当主节点宕机时，通知客户端和其他哨兵
- **自动故障转移**：选举新的主节点，指挥其他从节点切换

Redis如何处理缓存穿透问题？

缓存穿透是指查询不存在的数据，导致请求直达数据库：

- **解决方案**：
  - 缓存空值：对不存在的key缓存空值并设置短期过期
  - 布隆过滤器：在请求到达Redis前拦截不存在的key
  - 业务校验：提前过滤无效请求

Redis如何处理缓存击穿问题？

缓存击穿是指热点key过期瞬间，大量请求直达数据库：

- **解决方案**：
  - 热点key永不过期
  - 互斥锁：只有一个请求去数据库更新缓存
  - 提前刷新：定时任务提前更新即将过期的热点key

Redis如何处理缓存雪崩问题？

缓存雪崩是指大量key同时过期或Redis集群故障：

- **解决方案**：
  - 过期时间加随机值，避免集中过期
  - 多级缓存：本地缓存+Redis+数据库
  - 限流降级：保护后端系统
  - 高可用架构：Redis集群+哨兵模式

如何保证Redis和数据库数据一致性？

保证Redis和数据库数据一致性的常用方案：

- **Cache Aside Pattern**：读时先查缓存，无则查数据库并回写缓存；写时先更新数据库，再删除缓存
- **延时双删**：先删除缓存，再更新数据库，延迟一段时间后再次删除缓存，解决并发问题

- **异步更新缓存**：通过消息队列异步更新缓存，提高性能和可靠性
- **事务/分布式事务**：使用Redis事务或分布式事务保证操作原子性
- **版本号控制**：为数据添加版本号，防止旧数据覆盖新数据

Redis为什么是单线程的？单线程为什么还能高性能？

Redis采用单线程模型主要是因为：

- 避免了多线程的上下文切换开销
- 减少了锁竞争和同步问题

单线程高性能的原因：

- **内存操作**：所有数据都在内存中，读写速度快
- **非阻塞I/O**：使用epoll/kqueue等高效I/O多路复用机制
- **事件驱动**：基于事件驱动模型处理请求，减少等待时间
- **高效数据结构**：针对不同场景优化的数据结构（如跳表、压缩列表）

Redis的事务机制是怎样的？有什么局限性？

Redis事务通过MULTI、EXEC、WATCH等命令实现：

- **MULTI**：开始事务
- **EXEC**：执行事务中的所有命令
- **WATCH**：监视一个或多个key，若被修改则事务取消

局限性：

- 没有回滚机制：即使命令失败，已执行的命令也不会回滚
- 弱隔离性：事务执行期间，其他客户端的命令可能会插入执行
- 不支持复杂的事务逻辑：无法实现类似关系数据库的复杂事务

Redis的发布订阅机制是什么？有什么应用场景？

Redis的发布订阅(Pub/Sub)是一种消息通信模式：

- **发布者(Publisher)**：发送消息到频道(Channel)
- **订阅者(Subscriber)**：接收订阅频道的消息

应用场景：

- 实时聊天系统
- 实时数据推送
- 事件通知系统
- 消息分发系统

Redis的Bitmap是什么？有什么应用场景？

Bitmap是Redis提供的位操作数据结构，本质是字符串，但可以按位进行操作：

应用场景：

- 用户签到统计
- 活跃用户统计
- 在线状态标记
- 大规模数据的布尔表示