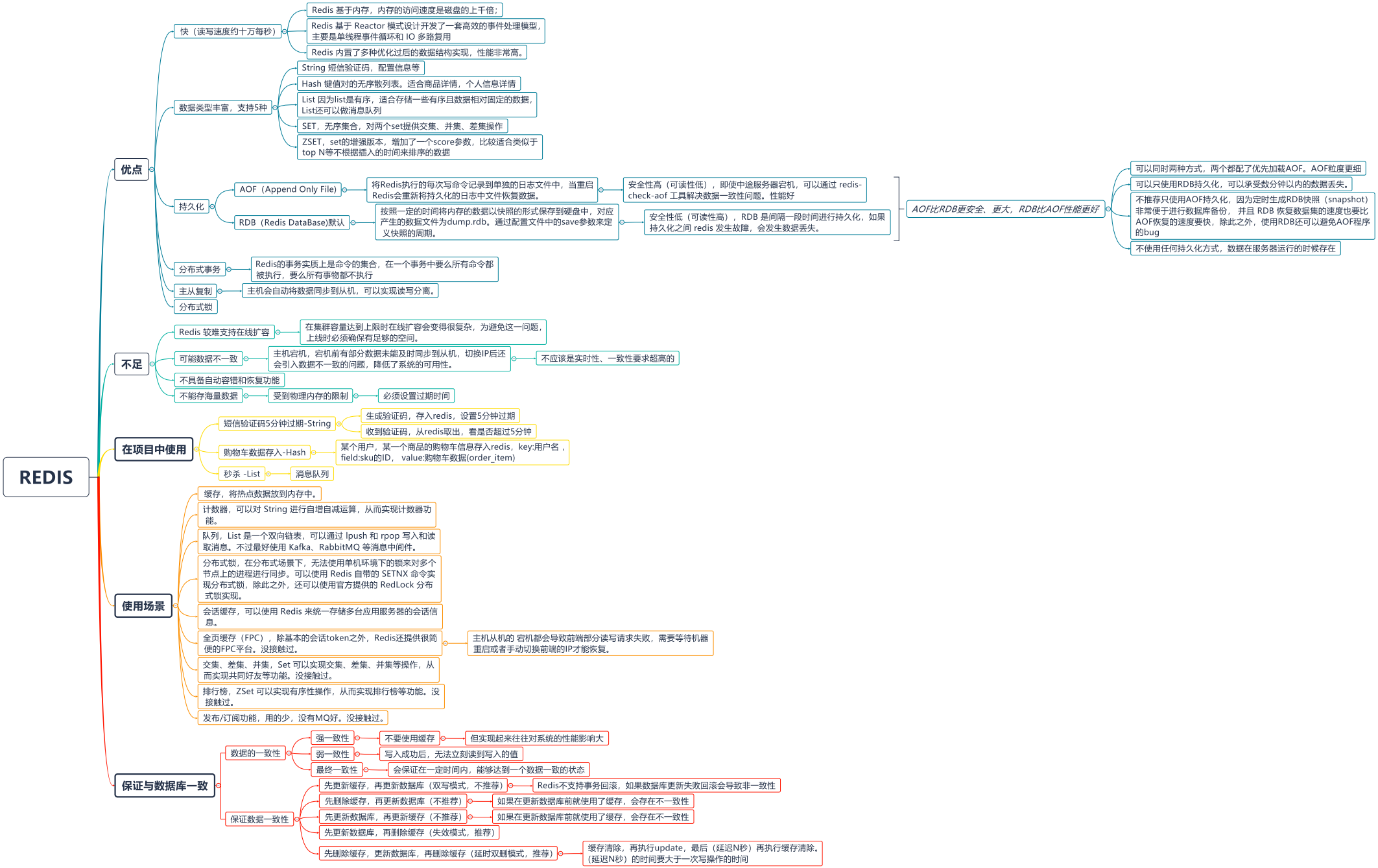
REDIS
优点
快（读写速度约十万每秒）
Redis 基于内存，内存的访问速度是磁盘的上千倍；
Redis 基于 Reactor 模式设计开发了一套高效的事件处理模型，主要是单线程事件循环和 IO 多路复用
Redis 内置了多种优化过后的数据结构实现，性能非常高。
数据类型丰富，支持5种
String 短信验证码，配置信息等
Hash 键值对的无序散列表。适合商品详情，个人信息详情
List 因为list是有序，适合存储一些有序且数据相对固定的数据，List还可以做消息队列
SET，无序集合，对两个set提供交集、并集、差集操作
ZSET，set的增强版本，增加了一个score参数，比较适合类似于top N等不根据插入的时间来排序的数据
持久化
AOF（Append Only File）
将Redis执行的每次写命令记录到单独的日志文件中，当重启Redis会重新将持久化的日志中文件恢复数据。
安全性高（可读性低），即使中途服务器宕机，可以通过 redis-check-aof 工具解决数据一致性问题。性能好
AOF比RDB更安全、更大，RDB比AOF性能更好
可以同时两种方式，两个都配了优先加载AOF。AOF粒度更细
可以只使用RDB持久化，可以承受数分钟以内的数据丢失。
不推荐只使用AOF持久化，因为定时生成RDB快照（snapshot）非常便于进行数据库备份， 并且 RDB 恢复数据集的速度也要比AOF恢复的速度要快，除此之外，使用RDB还可以避免AOF程序的bug
不使用任何持久化方式，数据在服务器运行的时候存在
RDB（Redis DataBase)默认
按照一定的时间将内存的数据以快照的形式保存到硬盘中，对应产生的数据文件为dump.rdb。通过配置文件中的save参数来定义快照的周期。
安全性低（可读性高），RDB 是间隔一段时间进行持久化，如果持久化之间 redis 发生故障，会发生数据丢失。
分布式事务
Redis的事务实质上是命令的集合，在一个事务中要么所有命令都被执行，要么所有事物都不执行
主从复制
主机会自动将数据同步到从机，可以实现读写分离。
分布式锁
不足
Redis 较难支持在线扩容
在集群容量达到上限时在线扩容会变得很复杂，为避免这一问题，上线时必须确保有足够的空间。
可能数据不一致
主机宕机，宕机前有部分数据未能及时同步到从机，切换IP后还会引入数据不一致的问题，降低了系统的可用性。
不应该是实时性、一致性要求超高的
不具备自动容错和恢复功能
不能存海量数据
受到物理内存的限制
必须设置过期时间
在项目中使用
短信验证码5分钟过期-String
生成验证码，存入redis，设置5分钟过期
收到验证码，从redis取出，看是否超过5分钟
购物车数据存入-Hash
某个用户，某一个商品的购物车信息存入redis，key:用户名，field:sku的ID， value:购物车数据(order_item)
秒杀 -List
消息队列
使用场景
缓存，将热点数据放到内存中。
计数器，可以对 String 进行自增自减运算，从而实现计数器功能。
队列，List 是一个双向链表，可以通过 lpush 和 rpop 写入和读取消息。不过最好使用 Kafka、RabbitMQ 等消息中间件。
分布式锁，在分布式场景下，无法使用单机环境下的锁来对多个节点上的进程进行同步。可以使用 Redis 自带的 SETNX 命令实现分布式锁，除此之外，还可以使用官方提供的 RedLock 分布式锁实现。
会话缓存，可以使用 Redis 来统一存储多台应用服务器的会话信息。
全页缓存（FPC），除基本的会话token之外，Redis还提供很简便的FPC平台。没接触过。
主机从机的 宕机都会导致前端部分读写请求失败，需要等待机器重启或者手动切换前端的IP才能恢复。
交集、差集、并集，Set 可以实现交集、差集、并集等操作，从而实现共同好友等功能。没接触过。
排行榜，ZSet 可以实现有序性操作，从而实现排行榜等功能。没接触过。
发布/订阅功能，用的少，没有MQ好。没接触过。
保证与数据库一致
数据的一致性
强一致性
不要使用缓存
但实现起来往往对系统的性能影响大
弱一致性
写入成功后，无法立刻读到写入的值
最终一致性
会保证在一定时间内，能够达到一个数据一致的状态
保证数据一致性
先更新缓存，再更新数据库（双写模式，不推荐）
Redis不支持事务回滚，如果数据库更新失败回滚会导致非一致性
先删除缓存，再更新数据库（不推荐）
如果在更新数据库前就使用了缓存，会存在不一致性
先更新数据库，再更新缓存（不推荐）
如果在更新数据库前就使用了缓存，会存在不一致性
先更新数据库，再删除缓存（失效模式，推荐）
先删除缓存，更新数据库，再删除缓存（延时双删模式，推荐）
缓存清除，再执行update，最后（延迟N秒）再执行缓存清除。（延迟N秒）的时间要大于一次写操作的时间