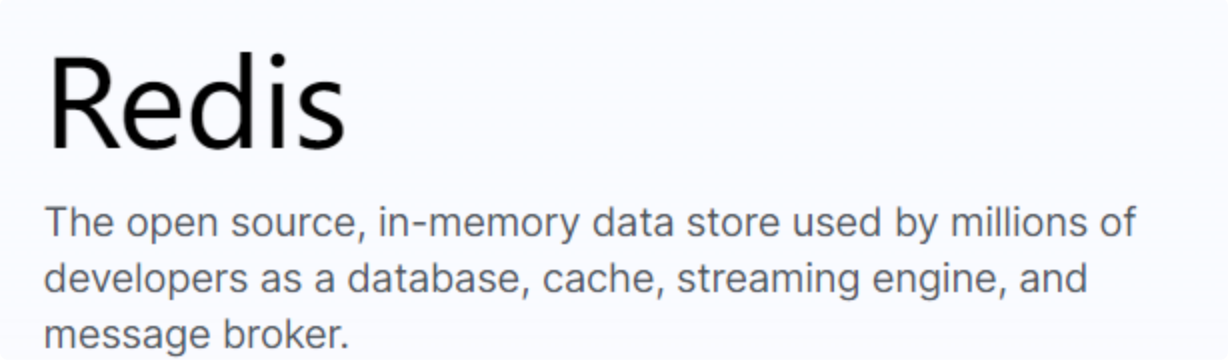
缓存利器-Redis

目录
1.什么是缓存
1.1 缓存常见使用目的
1.2 常见缓存方案
2.Redis优秀的底层设计
2.1 redis单线程处理请求
2.2 高效存储的数据类型
2.3 内存淘汰策略和内存过期删除策略

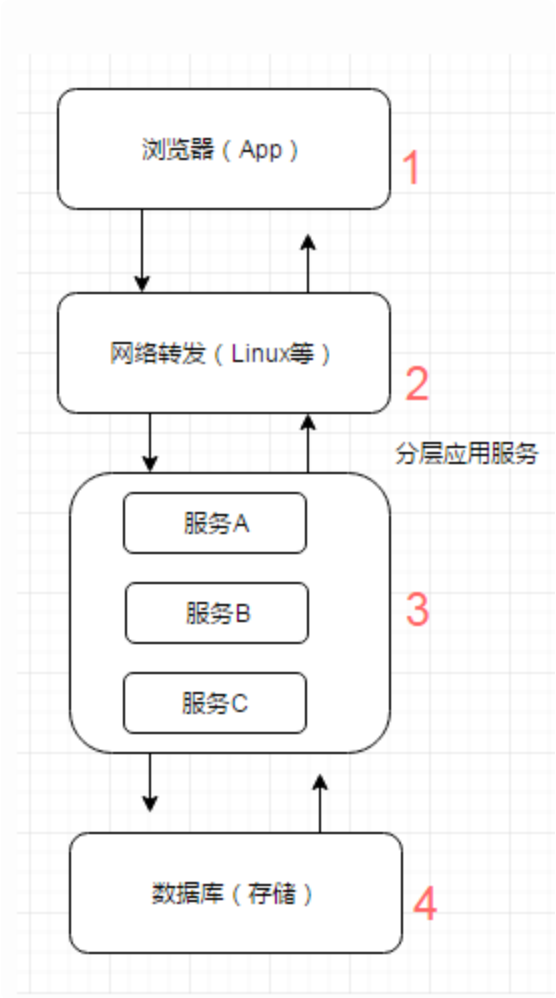


目录

1. 什么是缓存
2. Redis优秀的底层设计
3. 常见的Redis缓存使用问题
4. 其他--Redis作为分布式锁
5. 总结

1.什么是缓存

互联网应用一般流程



缓存可以在任何一个步骤使用；

缓存使用指标：命中率；最大空间；过期策略；缓存介质

1.1 缓存常见使用目的

1.2 常见缓存方案

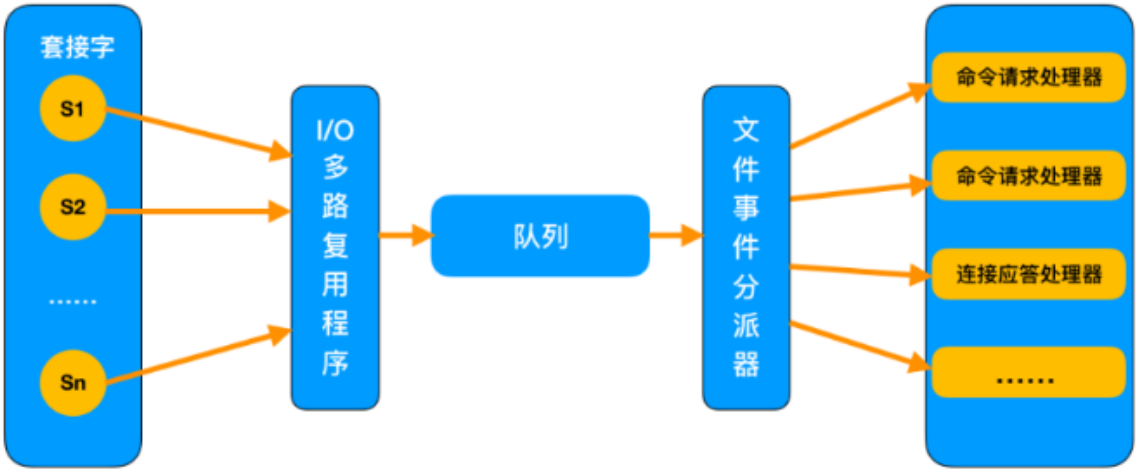
本地缓存：Guava Cache; Spring Cache；自定义使用HashMap作为缓存

分布式缓存： Redis; MemCache

比较项	GuavaCache	MemCache	Redis
支持分布式	否	半支持	支持水平拓展
是否支持持久化	否	否	支持（RDB和AOF）
缓存字段大小限制	一般无限制	一般key为250KB，value为1M	可到1GB
支持的数据类型	简单的key-value	简单的key-value	支持String，HashMap，List，Set，SortedSet，BitMap等丰富基础类型
过期策略	惰性删除和定期删除结合	惰性删除	惰性删除和定期删除结合
易用性	易用	易用	易用
多语言客户端	java	支持多种语言	支持多种语言
延时	亚毫秒级别	亚毫秒级别	亚毫秒级别

2.Redis优秀的底层设计

2.1 redis单线程处理请求

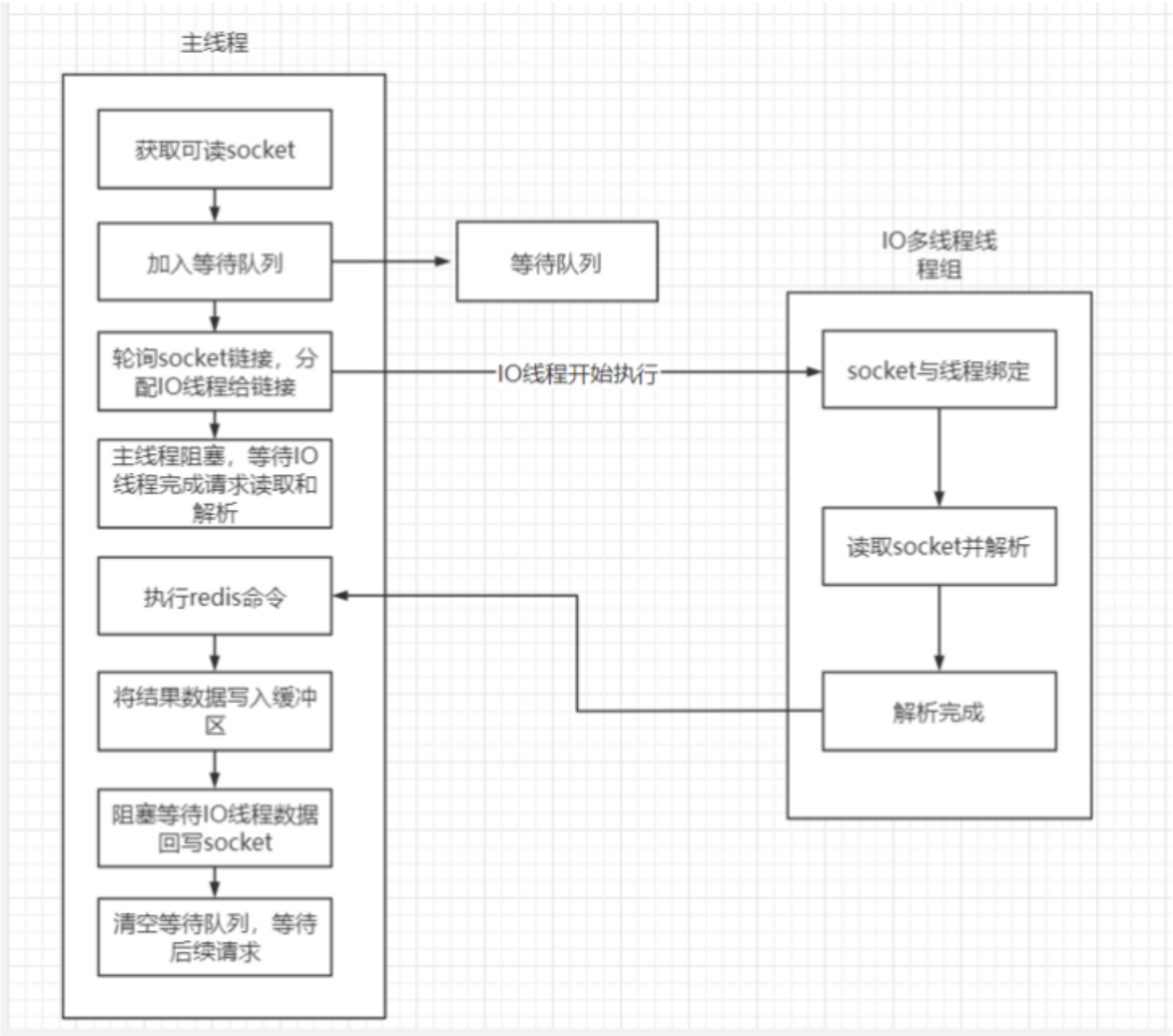


整个请求在一个线程里完成，避免处理多线程问题，redis性能瓶颈在内存，不在cpu，所以单线程已经满足

前提：Redis6.0以前版本

Redis6.0以后, 采用多个 IO 线程来处理网络请求, 后续的读写命令执行仍然是单线程

多线程能解决key或者value很大的情况导致计算资源紧张



2.2 高效存储的数据类型

redis使用RedisObject对象来存储所有的key-value

数据类型	编码方式	转换规则
REDIS_STRING	REDIS_ENCODING_INT	如果长度小于20并且可以value可以转换为整形
	REDIS_ENCODING_EMBSTR	如果长度小于44，使用此编码
	REDIS_ENCODING_RAW	默认编码
REDIS_LIST	OBJ_ENCODING_QUICKLIST	默认编码，6.0.0版本无其他编码
REDIS_HASH	REDIS_ENCODING_ZIPLIST	初始化默认编码
	REDIS_ENCODING_HT	当元素超过配置：hash-max-ziplist-entries时使用此编码，默认值是512
REDIS_SET	REDIS_ENCODING_INTSET	如果member可以转换为整形，则使用REDIS_ENCODING_INTSET编码，否则使用REDIS_ENCODING_HT编码
	REDIS_ENCODING_HT	
REDIS_ZSET	REDIS_ENCODING_ZIPLIST	如果 zset-max-ziplist-entries 配置为0并且，元素长度小于 zset-max-ziplist-value 配置(默认64)，则使用REDIS_ENCODING_SKIPLIST编码，否则使用REDIS_ENCODING_ZIPLIST编码
	REDIS_ENCODING_SKIPLIST	

String：简单动态字符串的应用，预分配内存，减少内存碎片化，提升响应时间

2.3 内存淘汰策略和内存过期删除策略

策略	流程	优缺点

定时删除	设置过期字典，设置定期删除事件到时间点后直接删除key 1 typedef struct redisDb { 2 dict *dict; /* 数据库键空间，存放着所有的键值对 */ 3 dict *expires; /* 键的过期时间 */ 4 .... 5 } redisDb;	优点：对内存友好 缺点：对cpu不友好
惰性删除	在过期时间到达之后，再去访问该key，才启动删除key	优点：对cpu友好 缺点：对内存不友好
定期删除	每隔一段时间「随机」从数据库中取出一定数量的 key 进行检查，并删除其中的过期key	优点：对内存和cpu有适中处理 缺点：抽取数量不好确定，难以确定执行时长和频率

Redis 选择「惰性删除+定期删除」这两种策略配和使用

其中定期删除是一个动态策略，依赖预上一次执行删除的数量比例确定要不要循环进行定期删除