Redis 简介

作者：dql
原文链接：https://ld246.com/article/1681311074176
来源网站：链滴

<h3 id="Redis-有哪些优缺点">Redis 有哪些优缺点</h3>
优点
• 读写性能优异， Redis 能读的速度是 110000 次/s，写的速度是 81000 次/s。
• 支持数据持久化，支持 AOF 和 RDB 两种持久化方式。
• 支持事务，Redis 的所有操作都是原子性的，同时 Redis 还支持对几个操
作合并后的原子性执行。
• 数据结构丰富，除了支持 string 类型的 value 外还支持 hash、set、
zset、list 等数据结构。
• 支持主从复制，主机会自动将数据同步到从机，可以进行读写分离。
<h4 id="缺点">缺点</h4>
• 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。
• Redis 不具备自动容错和恢复功能，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。
• 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。
• 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，运维人员在系统上线时必须确保有足够的空间，这
对资源造成了很大的浪费。
<h6 id="--RDB--快照并不是很可靠-如果你的电脑突然宕机了-或者电源断了-又或者不小心杀掉了程-那么最新的数据就会丢失-除了-RDB-持久化功能之外-Redis-还提供了-AOF-Append-Only-File-持久化功能-">==RDB==快照并不是很可靠。如果你的电脑突然宕机了，或者电源断了，又或者不心杀掉了进程，那么最新的数据就会丢失。除了 RDB 持久化功能之外，Redis 还提供了 AOF(Append Only File) 持久化功能。</h6>
<h6 id="----AOF----Append-Only-File---持久化默认是关闭的-通过将-redis-conf-中将-appendonly-no-修改为-appendonly-yes-来开启AOF-持久化功能-如果服务器开始了-AOF-持久化功能-服务会优先使用-AOF-文件来还原数据库状态-只有在-AOF-持久化功能处于关闭状态时-服务器才会使用-DB-文件来还原数据库状态-">== AOF== (Append Only File) 持久化默认关闭的，通过将 redis.conf 中将 appendonly no，修改为 appendonly yes 来开启 AOF 持久化功，如果服务器开始了 AOF 持久化功能，服务器会优先使用 AOF 文件来还原数据库状态。只有在 AOF 持久化功能处于关闭状态时，服务器才会使用 RDB 文件来还原数据库状态.</h6>
<h4 id="为什么要用-Redis--为什么要用缓存">为什么要用 Redis /为什么要用缓存</h4>
主要从“高性能”和“高并发”这两点来看待这个问题
<h6 id="高性能-">高性能：</h6>
假如用户第一次访问数据库中的某些数据。这个过程会比较慢，因为是从硬盘上
读取的。将该用户访问的数据存在数缓存中，这样下一次再访问这些数据的时候
就可以直接从缓存中获取了。操作缓存就是直接操作内存，所以速度相当快。如
果数据库中的对应数据改变的之后，同步改变缓存中相应的数据即可！
<h6 id="高并发-">高并发：</h6>
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，所以我们可以考虑
把数据库中的部分数据转移到缓存中去，这样用户的一部分请求会直接到缓存这
里而不用经过数据库
<h4 id="为什么要用-Redis-而不用-map-guava-做缓存-">为什么要用 Redis 而不用 map/guava 缓存?</h4>
<pre><code class="highlight-chroma">缓存分为==本地缓存==和==分布式缓存==，
</code></pre>
使用自带的 map 或者 guava 实现的是==本地缓存==，特点是轻量以及快速，生命周期随着 jv 的销毁而结束。多个实例下，每个缓存都需要保存一份实例，不具有一致性。
redis 或 memcached 之类的称为==分布式缓存，多实例情况下，==各实例共用一份缓存数据缓存具有一致性
<h3 id="Redis-为什么这么快">Redis 为什么这么快</h3>
1.完全基于内存，绝大部分请求是纯粹的内存操作，非常快速
2、数据结构简单，对数据操作也简单，Redis 中的数据结构是专门进行设计的；

3、采用单线程，避免了不必要的上下文切换和竞争条件，也不存在多进程或者
多线程导致的切换而消耗 CPU，不用去考虑各种锁的问题，不存在加锁释放锁操
作，没有因为可能出现死锁而导致的性能消耗；
4、使用多路 I/O 复用模型，非阻塞 IO；
5、使用底层模型不同，它们之间底层实现方式以及与客户端之间通信的应用协
议不一样，Redis 直接自己构建了 VM 机制
==多路 I/O 复用模型==：首先都会对一组文件描述符进行相关事件的注
，然后阻塞等待某些事件的发生或等待超时
<h5 id="Redis-主要有-5-种数据类型-">Redis 主要有 5 种数据类型：</h5>
包括 String，List，Set，Zset，Hash，满足大部分的使用要求

<h3 id="过期键的删除策略">过期键的删除策略</h3>
<h5 id="Redis-的过期键的删除策略">Redis 的过期键的删除策略</h5>
• ==定时过期==：每个设置过期时间的 key 都需要创建一个定时器，到过期时间
就会立即清除。对内存很友好；但是会占用大量的 CPU 资源去处理过期的数据，影响缓存的响
时间和吞吐
量。
•==惰性过期==：只有当访问一个 key 时，才会判断该 key 是否已过期，过期则
清除。该策略可以最大化地节省 CPU 资源，却对内存非常不友好。极端情
况可能出现大量的过期 key 没有再次被访问，从而不会被清除，占用大量
内存。
• ==定期过期==：每隔一定的时间，会扫描一定数量的数据库的 expires 字典中
一定数量的 key，并清除其中已过期的 key。该策略是前两者的一个折中
方案。通过调整定时扫描的时间间隔和每次扫描的限定耗时，可以在不同
情况下使得 CPU 和内存资源达到最优的平衡效果。
<h3 id="内存相关">内存相关</h3>
<h5 id="MySQL-里有-2000w-数据-redis-中只存-20w-的数据-如何保证">MySQL 里有 2000w 数
，redis 中只存 20w 的数据，如何保证</h5>
<h5 id="redis-中的数据都是热点数据">redis 中的数据都是热点数据</h5>
redis 内存数据集大小上升到一定大小的时候，就会施行数据淘汰策略。
<h4 id="Redis-的内存淘汰策略有哪些">Redis 的内存淘汰策略有哪些</h4>
Redis 的内存淘汰策略是指在 Redis 的用于缓存的内存不足时，怎么处理需要新
写入且需要申请额外空间的数据。
全局的键空间选择性移除
• noeviction：当内存不足以容纳新写入数据时，新写入操作会报错。
• allkeys-lru：当内存不足以容纳新写入数据时，在键空间中，移除最近
最少使用的 key。（这个是最常用的）
• allkeys-random：当内存不足以容纳新写入数据时，在键空间中，随机移
除某个 key。
设置过期时间的键空间选择性移除
• volatile-lru：当内存不足以容纳新写入数据时，在设置了过期时间的键
空间中，移除最近最少使用的 key。
• volatile-random：当内存不足以容纳新写入数据时，在设置了过期时间
的键空间中，随机移除某个 key。
• volatile-ttl：当内存不足以容纳新写入数据时，在设置了过期时间的键
空间中，有更早过期时间的 key 优先移除。