# 说说 Redis 都有哪些应用场景？

* **缓存**：这应该是 Redis 最主要的功能了，也是大型网站必备机制，合理地使用缓存不仅可以加 快数据的访问速度，而且能够有效地降低后端数据源的压力。
* **共享Session**：对于一些依赖 session 功能的服务来说，如果需要从单机变成集群的话，可以选择 redis 来统一管理 session。
* **消息队列系统**：消息队列系统可以说是一个大型网站的必备基础组件，因为其具有业务 解耦、非实时业务削峰等特性。Redis提供了发布订阅功能和阻塞队列的功 能，虽然和专业的消息队列比还不够足够强大，但是对于一般的消息队列功 能基本可以满足。比如在分布式爬虫系统中，使用 redis 来统一管理 url队列。
* **分布式锁**：在分布式服务中。可以利用Redis的setnx功能来编写分布式的锁，虽然这个可能不是太常用。

当然还有诸如排行榜、点赞功能都可以使用 Redis 来实现，但是 Redis 也不是什么都可以做，比如**数据量特别大时，不适合 Redis**，我们知道 Redis 是基于内存的，虽然内存很便宜，但是如果你每天的数据量特别大，比如几亿条的用户行为日志数据，用 Redis 来存储的话，成本相当的高。

# 单线程的 Redis 为什么这么快？

Redis 有多快？官方给出的答案是读写速度 10万/秒，这个数字不让人意外，但是 Redis 是单线程的。为什么单线程的 Redis 速度这么快？原因有以下三点：

* **纯内存操作**：Redis 是完全基于内存的，所以读写效率非常的高，当然 Redis 存在持久化操作，在持久化操作是都是 fork 子进程和利用 Linux 系统的页缓存技术来完成，并不会影响 Redis 的性能。
* **单线程操作**：单线程并不是坏事，单线程可以避免了频繁的上下文切换，频繁的上下文切换也会影响性能的。
* **合理高效的数据结构**
* **采用了非阻塞 I/O 多路复用机制**：多路I/O复用模型是利用 select、poll、epoll 可以同时监察多个流的 I/O 事件的能力，在空闲的时候，会把当前线程阻塞掉，当有一个或多个流有 I/O 事件时，就从阻塞态中唤醒，于是程序就会轮询一遍所有的流（epoll 是只轮询那些真正发出了事件的流），并且只依次顺序的处理就绪的流，这种做法就避免了大量的无用操作。

# 说说 Redis 的数据结构及使用场景

Redis 提供了 5种数据结构，每一种数据结构有各种的使用场景。

1. String 字符串

字符串类型是 Redis 最基础的数据结构，首先键都是字符串类型，而且 其他几种数据结构都是在字符串类型基础上构建的，我们常使用的 set key value 命令就是字符串。常用在缓存、计数、共享Session、限速等。

1. Hash 哈希

在Redis中，哈希类型是指键值本身又是一个键值对 结构，形如value={{field1，value1}，...{fieldN，valueN}}，添加命令：hset key field value。哈希可以用来存放用户信息，比如实现购物车

1. List 列表

列表（list）类型是用来存储多个有序的字符串。可以做简单的消息队列的功能。另外，可以利用 lrange 命令，做基于 Redis的分页功能，性能极佳，用户体验好。

1. Set 集合

集合（set）类型也是用来保存多个的字符串元素，但和列表类型不一 样的是，集合中不允许有重复元素，并且集合中的元素是无序的，不能通过 索引下标获取元素。利用 Set 的交集、并集、差集等操作，可以计算共同喜好，全部的喜好，自己独有的喜好等功能。

1. Sorted Set 有序集合

Sorted Set 多了一个权重参数 Score，集合中的元素能够按 Score 进行排列。可以做排行榜应用，取 TOP N 操作

# 说一说 Redis 的数据过期策略

先给大家一个结论，Redis 中数据过期策略采用定期删除+惰性删除策略。

**1、定期删除、惰性删除策略是什么？**

* **定期删除策略**：Redis 启用一个定时器定时监视所有的 key，判断key是否过期，过期的话就删除。这种策略可以保证过期的 key 最终都会被删除，但是也存在严重的缺点：每次都遍历内存中所有的数据，非常消耗 CPU 资源，并且当 key 已过期，但是定时器还处于未唤起状态，这段时间内 key 仍然可以用。
* **惰性删除策略**：在获取 key 时，先判断 key 是否过期，如果过期则删除。这种方式存在一个缺点：如果这个 key 一直未被使用，那么它一直在内存中，其实它已经过期了，会浪费大量的空间。

**2、定期删除+惰性删除策略是如何工作的？**

这两种策略天然的互补，结合起来之后，定时删除策略就发生了一些改变，不在是每次扫描全部的 key 了，而是随机抽取一部分 key 进行检查，这样就降低了对 CPU 资源的损耗，惰性删除策略互补了为检查到的key，基本上满足了所有要求。**但是有时候就是那么的巧，既没有被定时器抽取到，又没有被使用，这些数据又如何从内存中消失**？没关系，还有内存淘汰机制，当内存不够用时，内存淘汰机制就会上场。Redis 内存淘汰机制有以下几种策略：

* noeviction：当内存不足以容纳新写入数据时，新写入操作会报错。（Redis 默认策略）
* 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。（推荐使用）
* 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。
* 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。这种情况一般是把 Redis 既当缓存，又做持久化存储的时候才用。
* 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。

配置内存淘汰机制只需要在 redis.conf 配置文件中配置 maxmemory-policy 参数即可。

# 如何解决 Redis 缓存穿透和缓存雪崩问题

**缓存雪崩：** 由于缓存层承载着大量请求，有效地 保护了存储层，但是如果缓存层由于某些原因不能提供服务，比如 Redis 节点挂掉了，热点 key 全部失效了，在这些情况下，所有的请求都会直接请求到数据库，可能会造成数据库宕机的情况。

**预防和解决缓存雪崩问题**，可以从以下三个方面进行着手：

* **1、使用 Redis 高可用架构**：使用 Redis 集群来保证 Redis 服务不会挂掉
* **2、缓存时间不一致：** 给缓存的失效时间，加上一个随机值，避免集体失效
* **3、限流降级策略**：有一定的备案，比如个性推荐服务不可用了，换成热点数据推荐服务

**缓存穿透：** 缓存穿透是指查询一个根本不存在的数据，这样的数据肯定不在缓存中，这会导致请求全部落到数据库上，有可能出现数据库宕机的情况。

**预防和解决缓存穿透问题**，可以考虑以下两种方法：

* **1、缓存空对象：** 将空值缓存起来，但是这样就有一个问题，大量无效的空值将占用空间，非常浪费。
* **2、布隆过滤器拦截：** 将所有可能的查询key 先映射到布隆过滤器中，查询时先判断key是否存在布隆过滤器中，存在才继续向下执行，如果不存在，则直接返回。布隆过滤器有一定的误判，所以需要你的业务允许一定的容错性。