

绝大部分写业务的程序员，在实际开发中使用 Redis 的时候，只会 Set Value 和 Get Value 两个操作，对 Redis 整体缺乏一个认知。这里以面试题的形式对 Redis 常见问题做一个总结，解决大家的知识盲点。

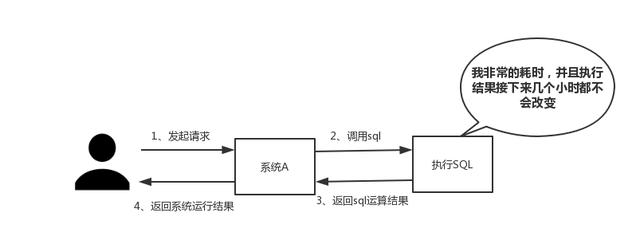
1、为什么使用 Redis？

在项目中使用 Redis，主要考虑两个角度：性能和并发。如果只是为了分布式锁这些其他功能，还有其他中间件 Zookpeer 等代替，并非一定要使用 Redis。

**性能：**

如下图所示，我们在碰到需要执行耗时特别久，且结果不频繁变动的 SQL，就特别适合将运行结果放入缓存。这样，后面的请求就去缓存中读取，使得请求能够迅速响应。

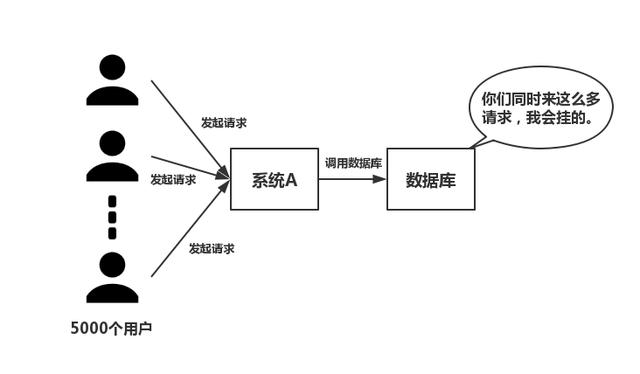
特别是在秒杀系统，在同一时间，几乎所有人都在点，都在下单。。。执行的是同一操作———向数据库查数据。



根据交互效果的不同，响应时间没有固定标准。在理想状态下，我们的页面跳转需要在瞬间解决，对于页内操作则需要在刹那间解决。

**并发：**

如下图所示，在大并发的情况下，所有的请求直接访问数据库，数据库会出现连接异常。这个时候，就需要使用 Redis 做一个缓冲操作，让请求先访问到 Redis，而不是直接访问数据库。



使用 Redis 的常见问题

* 缓存和数据库双写一致性问题
* 缓存雪崩问题
* 缓存击穿问题
* 缓存的并发竞争问题

2、单线程的 Redis 为什么这么快

这个问题是对 Redis 内部机制的一个考察。很多人都不知道 Redis 是单线程工作模型。

原因主要是以下三点：

* 纯内存操作
* 单线程操作，避免了频繁的上下文切换
* 采用了非阻塞 I/O 多路复用机制

仔细说一说 I/O 多路复用机制，打一个比方：小名在 A 城开了一家快餐店店，负责同城快餐服务。小明因为资金限制，雇佣了一批配送员，然后小曲发现资金不够了，只够买一辆车送快递。

**经营方式一**

客户每下一份订单，小明就让一个配送员盯着，然后让人开车去送。慢慢的小曲就发现了这种经营方式存在下述问题：

* 时间都花在了抢车上了，大部分配送员都处在闲置状态，抢到车才能去送。
* 随着下单的增多，配送员也越来越多，小明发现快递店里越来越挤，没办法雇佣新的配送员了。
* 配送员之间的协调很花时间。

综合上述缺点，小明痛定思痛，提出了经营方式二。

**经营方式二**

小明只雇佣一个配送员。当客户下单，小明按送达地点标注好，依次放在一个地方。最后，让配送员依次开着车去送，送好了就回来拿下一个。上述两种经营方式对比，很明显第二种效率更高。

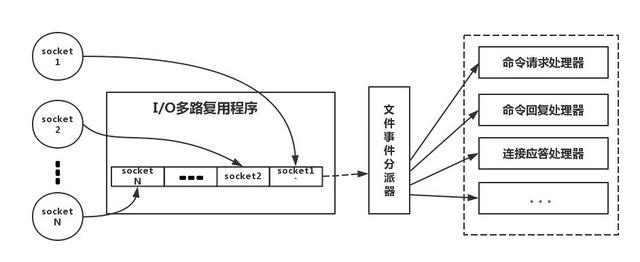
在上述比喻中：

* 每个配送员→每个线程
* 每个订单→每个 Socket(I/O 流)
* 订单的送达地点→Socket 的不同状态
* 客户送餐请求→来自客户端的请求
* 明曲的经营方式→服务端运行的代码
* 一辆车→CPU 的核数

于是有了如下结论：

* 经营方式一就是传统的并发模型，每个 I/O 流(订单)都有一个新的线程(配送员)管理。
* 经营方式二就是 I/O 多路复用。只有单个线程(一个配送员)，通过跟踪每个 I/O 流的状态(每个配送员的送达地点)，来管理多个 I/O 流。

下面类比到真实的 Redis 线程模型，如图所示：

-

Redis-client 在操作的时候，会产生具有不同事件类型的 Socket。在服务端，有一段 I/O 多路复用程序，将其置入队列之中。然后，文件事件分派器，依次去队列中取，转发到不同的事件处理器中。

3、Redis 的数据类型及使用场景

一个合格的程序员，这五种类型都会用到。

**String**

最常规的 set/get 操作，Value 可以是 String 也可以是数字。一般做一些复杂的计数功能的缓存。

**Hash**

这里 Value 存放的是结构化的对象，比较方便的就是操作其中的某个字段。我在做单点登录的时候，就是用这种数据结构存储用户信息，以 CookieId 作为 Key，设置 30 分钟为缓存过期时间，能很好的模拟出类似 Session 的效果。

**List**

使用 List 的数据结构，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。

**Set**

因为 Set 堆放的是一堆不重复值的集合。所以可以做全局去重的功能。我们的系统一般都是集群部署，使用 JVM 自带的 Set 比较麻烦。另外，就是利用交集、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。

**Sorted Set**

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。Sorted Set 可以用来做延时任务。

4、Redis 的过期策略和内存淘汰机制

Redis 是否用到家，从这就能看出来。比如你 Redis 只能存 5G 数据，可是你写了 10G，那会删 5G 的数据。怎么删的，这个问题思考过么？

正解：Redis 采用的是定期删除+惰性删除策略。

为什么不用定时删除策略

定时删除，用一个定时器来负责监视 Key，过期则自动删除。虽然内存及时释放，但是十分消耗 CPU 资源。在大并发请求下，CPU 要将时间应用在处理请求，而不是删除 Key，因此没有采用这一策略。

定期删除+惰性删除如何工作

定期删除，Redis 默认每个 100ms 检查，有过期 Key 则删除。需要说明的是，Redis 不是每个 100ms 将所有的 Key 检查一次，而是随机抽取进行检查。如果只采用定期删除策略，会导致很多 Key 到时间没有删除。于是，惰性删除派上用场。

采用定期删除+惰性删除就没其他问题了么

不是的，如果定期删除没删除掉 Key。并且你也没及时去请求 Key，也就是说惰性删除也没生效。这样，Redis 的内存会越来越高。那么就应该采用内存淘汰机制。

在 redis.conf 中有一行配置：

**# maxmemory-policy volatile-lru**

该配置就是配内存淘汰策略的：

* 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。（推荐使用，目前项目在用这种）(最近最久使用算法)
* 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。（应该也没人用吧，你不删最少使用 Key，去随机删）
* 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。（不推荐）
* 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。（依然不推荐）
* 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。（不推荐）

5、Redis 和数据库双写一致性问题

一致性问题还可以再分为最终一致性和强一致性。数据库和缓存双写，就必然会存在不一致的问题。前提是如果对数据有强一致性要求，不能放缓存。我们所做的一切，只能保证最终一致性。

另外，我们所做的方案从根本上来说，只能降低不一致发生的概率。因此，有强一致性要求的数据，不能放缓存。首先，采取正确更新策略，先更新数据库，再删缓存。其次，因为可能存在删除缓存失败的问题，提供一个补偿措施即可，例如利用消息队列。

6、如何应对缓存穿透和缓存雪崩问题

这两个问题，一般中小型传统软件企业很难碰到。如果有大并发的项目，流量有几百万左右，这两个问题一定要深刻考虑。缓存穿透，即黑客故意去请求缓存中不存在的数据，导致所有的请求都怼到数据库上，从而数据库连接异常。

缓存穿透解决方案：

* 利用互斥锁，缓存失效的时候，先去获得锁，得到锁了，再去请求数据库。没得到锁，则休眠一段时间重试。
* 采用异步更新策略，无论 Key 是否取到值，都直接返回。Value 值中维护一个缓存失效时间，缓存如果过期，异步起一个线程去读数据库，更新缓存。需要做缓存预热(项目启动前，先加载缓存)操作。
* 提供一个能迅速判断请求是否有效的拦截机制，比如，利用布隆过滤器，内部维护一系列合法有效的 Key。迅速判断出，请求所携带的 Key 是否合法有效。如果不合法，则直接返回。

缓存雪崩，即缓存同一时间大面积的失效，这个时候又来了一波请求，结果请求都怼到数据库上，从而导致数据库连接异常。

缓存雪崩解决方案：

* 给缓存的失效时间，加上一个随机值，避免集体失效。
* 使用互斥锁，但是该方案吞吐量明显下降了。
* 双缓存。我们有两个缓存，缓存 A 和缓存 B。缓存 A 的失效时间为 20 分钟，缓存 B 不设失效时间。自己做缓存预热操作。
* 然后细分以下几个小点：从缓存 A 读数据库，有则直接返回；A 没有数据，直接从 B 读数据，直接返回，并且异步启动一个更新线程，更新线程同时更新缓存 A 和缓存 B。

7、如何解决 Redis 的并发竞争 Key 问题

这个问题大致就是，同时有多个子系统去 Set 一个 Key。这个时候要注意什么呢？大家基本都是推荐用 Redis 事务机制。

但是我并不推荐使用 Redis 的事务机制。因为我们的生产环境，基本都是 Redis 集群环境，做了数据分片操作。你一个事务中有涉及到多个 Key 操作的时候，这多个 Key 不一定都存储在同一个 redis-server 上。因此，Redis 的事务机制，十分鸡肋。

如果对这个 Key 操作，不要求顺序

这种情况下，准备一个分布式锁，大家去抢锁，抢到锁就做 set 操作即可，比较简单。

如果对这个 Key 操作，要求顺序

假设有一个 key1，系统 A 需要将 key1 设置为 valueA，系统 B 需要将 key1 设置为 valueB，系统 C 需要将 key1 设置为 valueC。

期望按照 key1 的 value 值按照 valueA > valueB > valueC 的顺序变化。这种时候我们在数据写入数据库的时候，需要保存一个时间戳。

假设时间戳如下：

系统 A key 1 {valueA 3:00}

系统 B key 1 {valueB 3:05}

系统 C key 1 {valueC 3:10}

那么，假设系统 B 先抢到锁，将 key1 设置为{valueB 3:05}。接下来系统 A 抢到锁，发现自己的 valueA 的时间戳早于缓存中的时间戳，那就不做 set 操作了，以此类推。其他方法，比如利用队列，将 set 方法变成串行访问也可以。

8、总结

Redis 在国内各大公司都能看到其身影，比如我们熟悉的新浪，阿里，腾讯，百度，美团，小米等。学习 Redis，这几方面尤其重要：Redis 客户端、Redis 高级功能、Redis 持久化和开发运维常用问题探讨、Redis 复制的原理和优化策略、Redis 分布式解决方案等。

<https://segmentfault.com/a/1190000019910205>