Redis缓存三剑客：穿透、雪崩、击穿—手把手教你解决

原创 码上知新 码上知新 2025年03月14日 14:56 江苏

菜小弟：表哥，我最近在做一个电商系统，用了Redis做缓存，但遇到了很多问题，比如缓存穿透、缓存雪崩、缓存击穿，我都快被搞疯了！能帮我详细解释一下这些问题，以及如何解决吗？

表哥：没问题，这些问题确实是使用Redis时常见的挑战。我们先从最基本的概念开始，再一步步深入每个问题的原理和解决方案。

缓存穿透

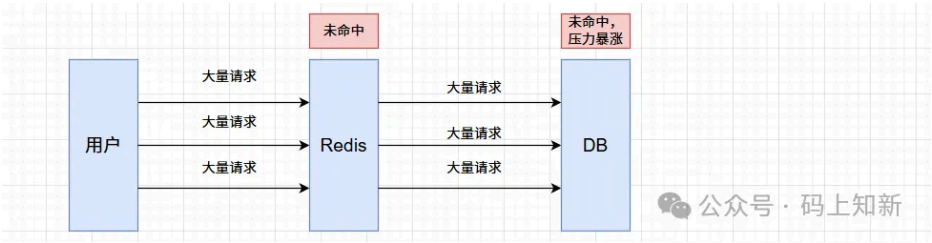
菜小弟：我先问问什么是缓存穿透？我听说是缓存查不到，直接去查数据库了。

表哥：没错。缓存穿透是指**查询一个缓存中不存在且数据库中也不存在的数据**，导致每次请求都直接访问数据库的行为。这种行为会让缓存完全失效，所有的请求都绕过缓存，直接打到数据库上。

菜小弟：那为什么会发生缓存穿透呢？

表哥：通常有两种情况：

1. **恶意攻击**：攻击者构造大量不存在的数据进行查询。比如使用随机生成的ID或者一些非法参数，试图让系统无法从缓存中获取数据，从而直接访问数据库。
2. **业务逻辑问题**：在某些情况下，用户可能会输入一些无效的查询参数（比如错误的关键字、不存在的ID等）。如果系统没有对这些参数进行校验，就会导致大量无效查询直接打到数据库。



表哥：你看，如果很多这样的请求同时发生，数据库的压力会很大。

菜小弟：那怎么解决呢？

表哥：有几个常见的解决方案：

缓存空值

当查询一个数据发现不存在时，将空值（比如 `Null` 或 `{}`）缓存起来，并设置一个较短的过期时间（比如30秒）。这样下次再查询相同数据时，可以直接从缓存中获取空值，而不需要再去查数据库。这种适合处理短期内的高频无效查询。 缓存空值会占用一定的缓存空间，如果空值过多，可能会导致缓存性能下降。

布隆过滤器

在缓存之前加一层布隆过滤器，用来判断数据是否存在。如果布隆过滤器判断数据不存在，就立即返回，不查缓存和数据库。布隆过滤器是一种概率型数据结构，用于判断一个元素是否在集合中。它的特点是**占用空间小**、**查询速度快**，但有一定的误判率。

布隆过滤器的核心思想

通过多个哈希函数将一个元素映射到位数组中的多个位置上，并将其标记为1。查询时，通过检查这些位置是否都为1来判断元素是否存在。

布隆过滤器主要由以下两部分组成

1. 位数组（Bit Array）一个长度为 `m` 的二进制数组，初始时所有位都为0。位数组的长度 `m` 决定了布隆过滤器的空间大小。
2. 多个哈希函数（Hash Functions）一组独立的哈希函数，用于将元素映射到位数组的不同位置上。哈希函数的数量 `k` 需要根据需求进行选择。

布隆过滤器操作原理

添加元素

- 使用 `k` 个哈希函数对元素进行哈希计算，得到 `k` 个哈希值。
- 将这 `k` 个哈希值映射到位数组的 `k` 个位置上。
- 将这些位置的值标记为1。

举例：假设布隆过滤器的位数组长度为 `m=10`，哈希函数数量为 `k=3`。添加元素 `A` 时，哈希函数计算出 `A` 的哈希值为 `2`、`5`、`8`，则将位数组的第2、5、8位置标记为1。

查询元素

- 使用 `k` 个哈希函数对元素进行哈希计算，得到 `k` 个哈希值。
- 检查这k个哈希值对应的位数组位置是否都为1。
    - 如果所有位置都为1，则认为元素可能存在（但存在误判率）。
    - 如果有任何一个位置为0，则元素一定不存在。

举例：继续上面的例子，查询元素 `B` 时，哈希函数计算出 `B` 的哈希值为 `2`、`5`、`9`。检查位数组的第2、5、9位：

- 第2位为1，第5位为1，第9位为0。
- 因为有位置为0，所以 `B` 一定不存在。

查询元素 `C` 时，哈希函数计算出 `C` 的哈希值为 `2`、`5`、`8`。检查位数组的第2、5、8位：

- 所有位置都为1。
- 所以 `C` 可能存在（但需要进一步确认）。

请求校验

- 检查商品ID是否符合格式要求。
- 检查关键字是否在允许的范围内。
- 校验用户权限，避免非法查询。

缓存雪崩

菜小弟：那缓存雪崩又是什么？

表哥：缓存雪崩是指**在某个时间点，缓存中的数据大面积同时失效（或缓存服务不可用）**，导致大量请求直接打到数据库，从而引发数据库压力骤增，甚至崩溃的现象。

菜小弟：为什么会发生缓存雪崩呢？

表哥：通常是因为以下两点：

1. **缓存数据同时过期**如果大量的缓存数据设置了相同的过期时间，那么这些数据将在同一时间失效，导致所有相关的查询直接访问数据库。
2. **缓存服务不可用**如果缓存服务本身出现问题（如宕机、网络故障等），所有缓存数据都无法访问，请求会直接打到数据库。

菜小弟：这确实很危险！那怎么解决呢？

表哥：有以下几个解决方案：

分散缓存过期时间

- 在设置缓存过期时间时，增加一个随机值。例如，设置过期时间为30分钟加上一个0到10分钟的随机数。
- 对于不同的业务数据，设置不同的过期时间。

缓存服务高可用

- 使用分布式缓存（如Redis Cluster），避免单点故障。
- 配置缓存服务的监控和自动恢复机制，及时发现和处理故障。

限流和降级

- 配置限流策略，限制同时访问数据库的请求数量。
- 设置降级方案，当数据库负载过高时，返回默认值或错误页面，避免进一步加重负载。

菜小弟：如果数据库真的扛不住了，会怎么样？

表哥：数据库可能会崩溃，导致整个系统不可用。所以缓存雪崩的后果非常严重，必须提前预防。

表哥：数据库可能会崩溃，导致整个系统不可用。所以缓存雪崩的后果非常严重，必须提前预防。

缓存击穿

菜小弟：缓存击穿到底是啥？

表哥：缓存击穿是指某个热点数据突然失效，导致大量请求同时查询数据库。和缓存雪崩类似，但缓存击穿是针对单个热点数据，而不是大量数据。

菜小弟：听起来好像挺严重的，那它会带来什么后果呢？

表哥：如果这个数据特别热点，可能会有成千上万的请求同时打到数据库，导致数据库瞬间过载，甚至崩溃。这样一来，整个系统的性能都会受到严重影响。

菜小弟：为什么会发生缓存击穿呢？

表哥：缓存击穿指的是**某个热点数据在缓存中过期的那一刻，突然有大量并发请求访问这个数据**，导致请求直接打到数据库，从而对数据库造成巨大压力。

菜小弟：那怎么解决呢？

表哥：有以下几个解决方案：

互斥锁

互斥锁是一种并发控制机制，用于防止多个线程同时访问共享资源。在缓存击穿的场景中，可以通过互斥锁确保**只有一个请求去访问数据库**，其他请求等待查询结果。

步骤解析：

1. **缓存未命中**：第一个请求发现缓存中没有数据。
2. **获取锁**：该请求尝试获取锁，成功后访问数据库。
3. **创建缓存**：从数据库中获取数据后，将数据写入缓存。
4. **释放锁**：释放锁，其他请求从缓存中获取数据。

永不过期

对于需要频繁访问的热点数据，可以设置永不过期，通过后台异步线程定期更新缓存，保证缓存的实时性。

缓存降级

降级处理是指**在某些极端情况下，暂时屏蔽对热点数据的访问**，直接返回默认值或错误提示，避免数据库被打爆。

菜小弟：互斥锁听起来很复杂，会不会影响性能？

表哥：确实会有一点性能损失，但相比于数据库崩溃，这点损失是可以接受的。

总结

表哥：现在我们来总结一下：

1. **缓存穿透**：查询不存在的数据。解决方案包括缓存空值、布隆过滤器、请求校验。
2. **缓存雪崩**：大量数据同时失效。解决方案包括设置不同的过期时间、限流和降级、使用缓存集群。
3. **缓存击穿**：热点数据突然失效。解决方案包括互斥锁、永不过期、缓存降级。

菜小弟：听起来这些解决方案都很实用，但实际项目中该怎么选择呢？

表哥：根据实际情况来选择：

- 如果系统容易被恶意攻击，优先考虑布隆过滤器和请求校验。
- 如果数据量很大，且存在热点数据，优先考虑互斥锁和缓存预热。
- 如果系统对数据实时性要求不高，可以考虑缓存降级。

表哥：另外，还可以结合监控和报警系统，及时发现和解决缓存问题。

参考资料：【Redis核心技术与实战】

Redis · 目录

上一篇
Redis哨兵机制解析：掌握高可用守护者的核心功能

下一篇
临界区守护者：解剖原生Redis锁的不凡之路