【缓存】面试题

以下面试题,基于网络整理,和自己编辑。具体参考的文章,会在文末给出所有的链接。

如果胖友有自己的疑问,欢迎在星球提问,我们一起整理吊吊的【缓存】面试题的大保健。

而题目的难度,尽量按照从容易到困难的顺序,逐步下去。

另外,本文只分享通用的【缓存】的面试题,关于 Redis、MemCache 会单独分享。

再另外,本文【缓存】指的更多的是分布式缓存服务,而不是 HTTP 缓存等等。

什么是缓存?

: 这个问题, 理解即可。

缓存,就是数据交换的缓冲区,针对服务对象的不同(本质就是不同的硬件)都可以构建缓存。

目的是,**把读写速度慢的介质的数据保存在读写速度快的介质中,从而提高读写速度,减少时间消耗**。例如:

- CPU 高速缓存: 高速缓存的读写速度远高于内存。
 - 。 CPU 读数据时,如果在高速缓存中找到所需数据,就不需要读内存
 - 。 CPU 写数据时, 先写到高速缓存, 再回写到内存。
- 磁盘缓存:磁盘缓存其实就把常用的磁盘数据保存在内存中,内存读写速度也是远高于磁盘的。
 - 。 读数据, 时从内存读取。
 - 写数据时,可先写到内存,定时或定量回写到磁盘,或者是同步回写。

为什么要用缓存?

正如在 <u>「什么是缓存?」</u> 问题中所看到的,使用缓存的目的,就是提升读写性能。而实际业务场景下,更多的是为了提升**读性能**,带来更好的性能,更高的并发量。

日常业务中,我们使用比较多的数据库是 MySQL ,缓存是 Redis。一起来看看,阿里云提供的性能规格:

- Redis 性能规格,<u>https://help.aliyun.com/document_detail/26350.html</u> 。打底 8W QPS ,最高可达干万QPS 。
- MySQL 性能规格 https://help.aliyun.com/document_detail/53637.html。 打底 1.4K QPS ,最高 7W QPS

如此一比较,Redis 比 MySQL 的读写性能好很多。那么,我们将 MySQL 的热点数据,缓存到 Redis 中,提升读取性能,也减小 MySQL 的读取压力。例如说:

- 论坛帖子的访问频率比较高,且要实时更新阅读量,使用 Redis 记录帖子的阅读量,可以提升性能和并发。
- 商品信息,数据更新的频率不高,但是读取的频率很高,特别是热门商品。

请说说有哪些缓存算法? 是否能手写一下 LRU 代码的实现?

缓存算法,比较常见的是三种:

- LRU (least recently used,最近最少使用)
- LFU (Least Frequently used, 最不经常使用)
- FIFO (first in first out, 先进先出)

完整的话,胖友可以看看《缓存、缓存算法和缓存框架简介》的「缓存算法」部分。

塚 手写 LRU 代码的实现

手写 LRU 代码的实现,有多种方式。其中,最简单的是基于 LinkedHashMap 来实现,代码如下:

```
class LRUCache<K, V> extends LinkedHashMap<K, V> {
   private final int CACHE_SIZE;
   /**
    * 传递进来最多能缓存多少数据
    * @param cacheSize 缓存大小
   public LRUCache(int cacheSize) {
       // true 表示让 LinkedHashMap 按照访问顺序来进行排序,最近访问的放在头部,最老访问的放在尾
部。
       super((int) Math.ceil(cacheSize / 0.75) + 1, 0.75f, true);
       CACHE_SIZE = cacheSize;
   }
   @override
   protected boolean removeEldestEntry(Map.Entry<K, V> eldest) {
       // 当 map 中的数据量大于指定的缓存个数的时候,就自动删除最老的数据。
       return size() > CACHE_SIZE;
   }
}
```

其它更复杂,更能体现个人编码能力的 LRU 实现方式,可以看看如下两篇文章:

- <u>《动手实现一个 LRU Cache》</u>
- 《缓存、缓存算法和缓存框架简介》 文末,并且还提供了 FIFO、LFU 的代码实现。

常见的常见的缓存工具和框架有哪些?

在 Java 后端开发中,常见的缓存工具和框架列举如下:

- 本地缓存: Guava LocalCache、Ehcache、Caffeine。
 - 。 Ehcache 的功能更加丰富,Caffeine 的性能要比 Guava LocalCache 好。
- 分布式缓存: Redis、MemCache、Tair。
 - o Redis 最为主流和常用。

用了缓存之后,有哪些常见问题?

常见的问题,可列举如下:

- 缓存何时写入? 并且写时如何避免并发重复写入?
- 缓存如何失效?
- 缓存和 DB 的一致性如何保证?
- 如何避免缓存穿透的问题?
- 如何避免缓存击穿的问题?
- 如果避免缓存雪崩的问题?

: 重点可以去"记"加粗的五个词。

下面,我们会对每个问题,逐步解析。

当查询缓存报错,怎么提高可用性?

缓存可以极大的提高查询性能,但是缓存数据丢失和缓存不可用不能影响应用的正常工作。

因此,一般情况下,如果缓存出现异常,需要手动捕获这个异常,并且记录日志,并且从数据库查询数据返回给用户,而不应该导致业务不可用。

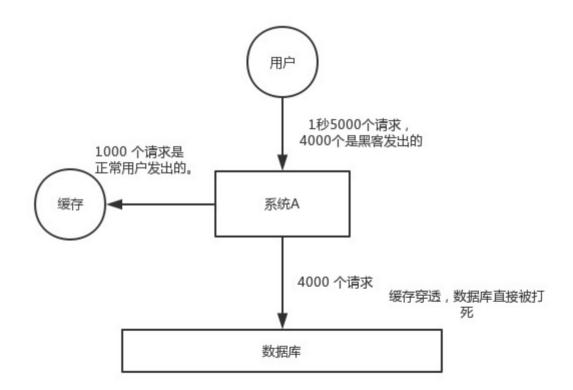
当然,这样做可能会带来缓存雪崩的问题。具体怎么解决,可以看看本文 「如何避免缓存"雪崩"的问题? | 问题。

如果避免缓存"穿透"的问题?

烬 缓存穿透

缓存穿透,是指查询一个一定不存在的数据,由于缓存是不命中时被动写(*被动写,指的是从 DB 查询到数据,则更新到缓存中*)的,并且处于容错考虑,如果从 DB 查不到数据则不写入缓存,这将导致这个不存在的数据每次请求都要到 DB 去查询,失去了缓存的意义。

在流量大时,可能 DB 就挂掉了,要是有人利用不存在的 key 频繁攻击我们的应用,这就是漏洞。如下图:



• 在 <u>「为什么要用缓存?」</u>中,我们已经看到,MySQL 的性能是远不如 Redis 的,如果大量的请求直接打到 MySQL ,则会直接打挂 MySQL 。

塚 如何解决

有两种方案可以解决:

- 方案一,缓存空对象: 当从 DB 查询数据为空,我们仍然将这个空结果进行缓存,具体的值需要使用特殊的标识,能和真正缓存的数据区分开。另外,需要设置较短的过期时间,一般建议不要超过 5 分钟。
- 方案二, BloomFilter 布隆过滤器:在缓存服务的基础上,构建 BloomFilter 数据结构,在 BloomFilter 中存储对应的 KEY 是否存在,如果存在,说明该 KEY 对应的值为空。那么整个逻辑的如下:
 - 1、根据 KEY 查询缓存。如果存在对应的值,直接返回;如果不存在,继续向下执行。
 - 2、根据 KEY 查询在缓存 BloomFilter 的值。如果存在值,说明该 KEY 不存在对应的值,直接返回空;如果不存在值,继续向下执行。
 - 3、查询 DB 对应的值,如果存在,则更新到缓存,并返回该值。如果不存在值,更新到 缓存 BloomFilter 中,并返回空。

ሜ 选择

这两个方案, 各有其优缺点。

	缓存空对象	BloomFilter 布隆过滤器
适用场景	1、数据命中不高 2、保证一致性	1、数据命中不高 2、数据相对固定、实时性低
维护成 本	1、代码维护简单 2、需要过多的缓存空间 3、数据 不一致	1、代码维护复杂 2、缓存空间占用小

• 对于 BloomFilter 布隆过滤器,使用场景中的"数据相对固定、实时性低", TODO 不支持删除?

实际情况下,使用方案二比较多。因为,相比方案一来说,更加节省内容,对缓存的负荷更小。并且,常用的缓存 Redis 支持构建 BloomFilter 数据结构,具体可以看看如下文章:

- 《Google Guava之BloomFilter 源码分析及基于 Redis 的重构》
- 《基于 Redis 的 BloomFilter 实现》
- 开源项目: https://github.com/erikdubbelboer/Redis-Lua-scaling-bloom-filter

另外,推荐看下<u>《Redis架构之防雪崩设计:网站不宕机背后的兵法》</u>文章的<u>「一、缓存穿透预防及优化」</u>,大神解释的更好,且提供相应的图和伪代码。

如何避免缓存"雪崩"的问题?

% 缓存雪崩

缓存雪崩,是指缓存由于某些原因无法提供服务(例如,缓存挂掉了),所有请求全部达到 DB 中,导致 DB 负荷大增,最终挂掉的情况。

塚 如何解决

预防和解决缓存雪崩的问题,可以从以下多个方面进行共同着手。

1) 缓存高可用

通过搭建缓存的高可用,避免缓存挂掉导致无法提供服务的情况,从而降低出现缓存雪崩的情况。

假设我们使用 Redis 作为缓存,则可以使用 Redis Sentinel 或 Redis Cluster 实现高可用。

2) 本地缓存

如果使用本地缓存时,即使分布式缓存挂了,也可以将 DB 查询到的结果缓存到本地,避免后续请求全部到达 DB 中。

当然,引入本地缓存也会有相应的问题,例如说:

- 本地缓存的实时性怎么保证?
 - 方案一,可以引入消息队列。在数据更新时,发布数据更新的消息;而进程中有相应的消费者消费该消息,从而更新本地缓存。

也可以使用 Redis Pub / Sub 取代消息队列来实现,但是 TT 此时 Redis 可能已经挂了,所以也不一定合适。

- 方案二,设置较短的过期时间,请求时从 DB 重新拉取。
- 方案三,使用「如果避免缓存"击穿"的问题? | 问题的【方案二】,手动过期。
- 每个进程可能会本地缓存相同的数据,导致数据浪费?
 - 方案一,需要配置本地缓存的过期策略和缓存数量上限。

: 上述的几个方案写的有点笼统,如果有不理解的地方,请在星球给留言。

如果我们使用 JVM ,则可以使用 Ehcache、Guava Cache 实现本地缓存的功能。

3) 请求 DB 限流

通过限制 DB 的每秒请求数,避免把 DB 也打挂了。这样至少能有两个好处:

- 1. 可能有一部分用户,还可以使用,系统还没死透。
- 2. 未来缓存服务恢复后,系统立即就已经恢复,无需在处理 DB 也挂掉的情况。

当然,被限流的请求,我们最好也要有相应的处理,走4)服务降级/。

如果我们使用 Java ,则可以使用 Guava RateLimiter、Sentinel 实现限流的功能。

4) 服务降级

如果请求被限流,或者请求 DB 超时,我们可以服务降级,提供一些默认的值,或者友情提示,甚至空白的值也行。

如果我们使用 Java ,则可以使用 Hystrix、Sentinel 实现限流的功能。

5) 提前演练

在项目上线前,演练缓存宕掉后,应用以及后端的负载情况以及可能出现的问题,在此基础上做一些预案设定。

另外,推荐看下<u>《Redis架构之防雪崩设计:网站不宕机背后的兵法》</u>文章的<u>「二、缓存雪崩问题优化」</u>,大神解释的更好,且提供相应的图和伪代码。

如果避免缓存"击穿"的问题?

塚 缓存击穿

缓存击穿,是指某个**极度"热点"**数据在某个时间点过期时,恰好在这个时间点对这个 KEY 有大量的并发请求过来,这些请求发现缓存过期一般都会从 DB 加载数据并回设到缓存,但是这个时候大并发的请求可能会瞬间 DB 压垮。

- 对于一些设置了过期时间的 KEY , 如果这些 KEY 可能会在某些时间点被超高并发地访问 , 是一种非常"热点"的数据。这个时候 , 需要考虑这个个问题。
- 缓存被"击穿"的问题,和缓存"雪崩""的区别在于,前者针对某一 KEY 缓存,后者则是很多 KEY。
- 缓存被"击穿"的问题,和缓存"穿透""的区别在于,这个 KEY 是真实存在对应的值的。

ሜ 如何解决

有两种方案可以解决:

- 方案一,使用互斥锁:请求发现缓存不存在后,去查询 DB 前,使用分布式锁,保证有且只有一个线程去查询 DB ,并更新到缓存。流程如下:
 - 1、获取分布式锁,直到成功或超时。如果超时,则抛出异常,返回。如果成功,继续向下执行。
 - 2、再去缓存中。如果存在值,则直接返回;如果不存在,则继续往下执行。 ② 因为,获得到锁,可能已经被"那个"线程去查询过 DB ,并更新到缓存中了。
 - 。 3、查询 DB , 并更新到缓存中, 返回值。
- 方案二,手动过期:缓存上从不设置过期时间,功能上将过期时间存在 KEY 对应的 VALUE 里,如果发现要过期,通过一个后台的异步线程进行缓存的构建,也就是"手动"过期。通过后台的异步线程,保证有且只有一个线程去查询 DB。

ሜ 选择

这两个方案, 各有其优缺点。

	使用互斥锁	手动过期
优点	1、思路简单 2、保证一致性	1、性价最佳,用户无需等待
缺点	1、代码复杂度增大 2、存在死锁的风险	1、无法保证缓存一致性

具体使用哪一种方案,胖友可以根据自己的业务场景去做选择。

• 有一点要注意,上述的两个方案,都是建立在**极度"热点"**数据存在的情况,所以实际场景下,需要结合 <u>「如果避免缓存"穿透"的问题?</u> 的方案,一起使用。

另外,推荐看下<u>《Redis 架构之防雪崩设计:网站不宕机背后的兵法》</u>文章的<u>「三、缓存热点 key 重建优化」</u>,大神解释的更好,且提供相应的图和伪代码。

缓存和 DB 的一致性如何保证?

% 产生原因

主要有两种情况, 会导致缓存和 DB 的一致性问题:

- 1. 并发的场景下,导致读取老的 DB 数据,更新到缓存中。
- 2. 缓存和 DB 的操作,不在一个事务中,可能只有一个操作成功,而另一个操作失败,导致不一致。

当然,有一点我们要注意,缓存和 DB 的一致性,我们指的更多的是最终一致性。我们使用缓存只要是提高读操作的性能,真正在写操作的业务逻辑,还是以数据库为准。例如说,我们可能缓存用户钱包的余额在缓存中,在前端查询钱包余额时,读取缓存,在使用钱包余额时,读取数据库。

ሜ 解决方案

在开始说解决方案之前,胖友先看看如下几篇文章,可能有一丢丢多,保持耐心。

- 左耳朵耗子
 - 《缓存更新的套路》
- 沈剑
 - · <u>《缓存架构设计细节二三事》</u>
 - 《缓存与数据库一致性优化》

下面,我们就来看看几种方案。当然无论哪种方案,比较重要的就是解决两个问题:

- 1、将缓存可能存在的并行写,实现串行写。
- 2、实现数据的最终一致性。
- 1) 先淘汰缓存, 再写数据库

因为先淘汰缓存,所以数据的最终一致性是可以得到有效的保证的。为什么呢?先淘汰缓存,即使写数据库发生异常,也就是下次缓存读取时,多读取一次数据库。

但是,这种方案会存在缓存和 DB 的数据会不一致的情况,《缓存与数据库一致性优化》已经说了。

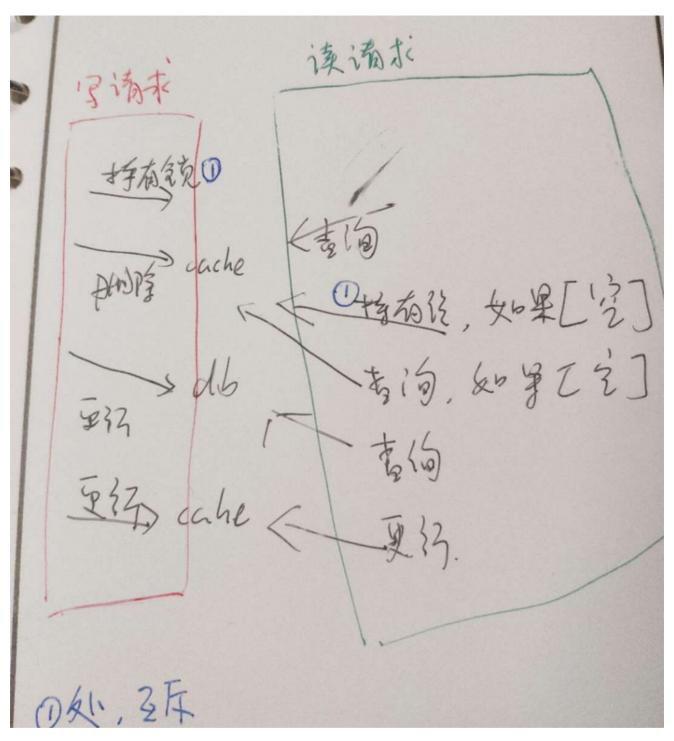
那么,我们需要解决缓存并行写,实现串行写。比较简单的方式,引入分布式锁。

- 在写请求时,先淘汰缓存之前,获取该分布式锁。
- 在读请求时,发现缓存不存在时,先获取分布式锁。

这样,缓存的并行写就成功的变成串行写落。实际上,就是<u>「如果避免缓存"击穿"的问题?</u>的【方案一】互斥锁的加强版。

整体执行,如下草图:

: 临时手绘, 不要打我。字很丑, 哈哈哈哈。



• 写请求时,是否主动更新缓存,根据自己业务的需要,是否有,都没问题。

2) 先写数据库, 再更新缓存

按照"先写数据库,再更新缓存",我们要保证 DB 和缓存的操作,能够在"同一个事务"中,从而实现最终一致性。

基于定时任务来实现

- 首先,写入数据库。
- 然后,在写入数据库所在的事务中,插入一条记录到任务表。该记录会存储需要更新的缓存 KEY 和 VALUE。
- 【异步】最后, 定时任务每秒扫描任务表, 更新到缓存中, 之后删除该记录。

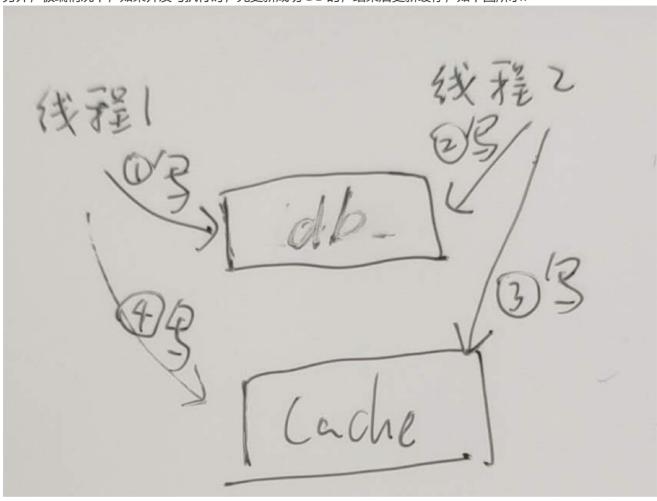
基于消息队列来实现

• 首先,写入数据库。

- 然后,发送带有缓存 KEY 和 VALUE 的事务消息。此时,需要有支持事务消息特性的消息队列,或者我们自己 封装消息队列,支持事务消息。
- 【异步】最后,消费者消费该消息,更新到缓存中。

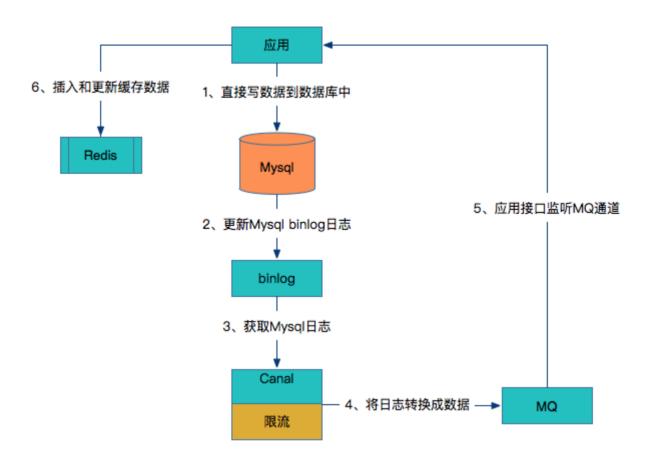
这两种方式,可以进一步优化,可以先尝试更新缓存,如果失败,则插入任务表,或者事务消息。

另外, 极端情况下, 如果并发写执行时, 先更新成功 DB 的, 结果后更新缓存, 如下图所示:



- : 灵魂画手, 哈哈哈哈。
- 理论来说,希望的更新缓存顺序是,线程1快于线程2,但是实际线程1晚于线程2,导致数据不一致。
- 可能胖友会说,图中不是基于定时任务或消息队列来实现异步更新缓存啊?答案是一直的,如果网络抖动,导致【插入任务表,或者事务消息】的顺序不一致。
- 那么怎么解决呢?需要做如下三件事情:
 - o 1、在缓存值中,拼接上数据版本号或者时间戳。例如说: value = {value: 原值, version: xxx}。
 - 2、在任务表的记录,或者事务消息中,增加上数据版本号或者时间戳的字段。
 - 3、在定时任务或消息队列执行更新缓存时,先读取缓存,对比版本号或时间戳,大于才进行更新。 ③ 当然,此处也会有并发问题,所以还是得引入分布式锁或 CAS 操作。
 - 关于 Redis 分布式锁,可以看看 <u>《精尽 Redis 面试题》</u>的 <u>「如何使用 Redis 实现分布式锁?」</u>问题
 - 关于 Redis CAS 操作,可以看看 <u>《精尽 Redis 面试题》</u> 的 <u>「什么是 Redis 事务?」</u> 问题。
- 3) 基于数据库的 binlog 日志

:如下内容,引用自<u>《技术专题讨论第五期:论系统架构设计中缓存的重要性》</u>文章,超哥对这个问题的回答。



- 应用直接写数据到数据库中。
- 数据库更新binlog日志。
- 利用Canal中间件读取binlog日志。
- Canal借助于限流组件按频率将数据发到MQ中。
- 应用监控MQ通道,将MQ的数据更新到Redis缓存中。

可以看到这种方案对研发人员来说比较轻量,不用关心缓存层面,而且这个方案虽然比较重,但是却容易形成统一的解决方案。

当然,以上种种方案,各有其复杂性,如果胖友心里没底,还是仅仅使用如下任一方案:

- "先淘汰缓存,再写数据库"的方案,并且无需引入分布式锁。
- "先写数据库,再更新缓存"的方案,并且无需引入定时任务或者消息队列。

原因如下:

FROM 基友老梁的总结

使用缓存过程中,经常会遇到缓存数据的不一致性和脏读现象。一般情况下,采取缓存双淘汰机制,在更新数据库的**前**淘汰缓存。此外,设定超时时间,例如三十分钟。

极端场景下,即使有脏数据进入缓存,这个脏数据也最存在一段时间后自动销毁。

- 重点,是最后一句话哟。
- 真的,和几个朋友沟通了下,真的出现不一致的情况,靠缓存过期后,重新从 DB 中读取即可。

另外,在 DB 主从架构下,方案会更加复杂。详细可以看看《主从 DB 与 cache 一致性优化》。

: 这是一道相对复杂的问题,重点在于理解为什么产生不一致的原因,然后针对这个原因去解决。

什么是缓存预热?如何实现缓存预热?

ሜ 缓存预热

在刚启动的缓存系统中,如果缓存中没有任何数据,如果依靠用户请求的方式重建缓存数据,那么对数据库的压力 非常大,而且系统的性能开销也是巨大的。

此时,最好的策略是启动时就把热点数据加载好。这样,用户请求时,直接读取的就是缓存的数据,而无需去读取 DB 重建缓存数据。

举个例子,热门的或者推荐的商品,需要提前预热到缓存中。

% 如何实现

一般来说, 有如下几种方式来实现:

- 1. 数据量不大时,项目启动时,自动进行初始化。
- 2. 写个修复数据脚本, 手动执行该脚本。
- 3. 写个管理界面,可以手动点击,预热对应的数据到缓存中。

缓存数据的淘汰策略有哪些?

除了缓存服务器自带的缓存**自动**失效策略之外,我们还可以根据具体的业务需求进行自定义的**"手动"**缓存淘汰,常见的策略有两种:

- 1、定时去清理过期的缓存。
- 2、当有用户请求过来时,再判断这个请求所用到的缓存是否过期,过期的话就去底层系统得到新数据并更新 缓存。

两者各有优劣,第一种的缺点是维护大量缓存的 key 是比较麻烦的,第二种的缺点就是每次用户请求过来都要判断缓存失效,逻辑相对比较复杂! 具体用哪种方案,大家可以根据自己的应用场景来权衡。

缓存如何存储 POJO 对象?

实际场景下,缓存值可能是一个 POJO 对象,就需要考虑如何 POJO 对象存储的问题。目前有两种方式:

- 方案一,将 POJO 对象序列化进行存储,适合 Redis 和 Memcached。
 - 可参考 《Redis 序列化方式StringRedisSerializer、FastJsonRedisSerializer和KryoRedisSerializer》文章。
- 方案二,使用 Hash 数据结构,适合 Redis。
 - o 可参考《Redis 之序列化 POJO》 文章。

666. 彩蛋

参考与推荐如下文章:

- _痕迹 _《缓存那些事 (二) 什么是缓存以及缓存的作用》
- yanglbme 《在项目中缓存是如何使用的?缓存如果使用不当会造成什么后果?》
- boothsun <u>《缓存常见问题》</u>

• 超神杀戮 《缓存穿透与缓存雪崩》