缓存三大问题及解决方案

**1. 缓存来由**

随着互联网系统发展的逐步完善，提高系统的qps，目前的绝大部分系统都增加了缓存机制从而避免请求过多的直接与数据库操作从而造成系统瓶颈，极大的提升了用户体验和系统稳定性。

**2. 缓存问题**

虽然使用缓存给系统带来了一定的质的提升，但同时也带来了一些需要注意的问题。

2.1 缓存穿透

缓存穿透是指查询一个一定不存在的数据，因为缓存中也无该数据的信息，则会直接去数据库层进行查询，从系统层面来看像是穿透了缓存层直接达到db，从而称为缓存穿透，没有了缓存层的保护，这种查询一定不存在的数据对系统来说可能是一种危险，如果有人恶意用这种一定不存在的数据来频繁请求系统，不，准确的说是攻击系统，请求都会到达数据库层导致db瘫痪从而引起系统故障。

2.2 解决方案

缓存穿透业内的解决方案已经比较成熟，主要常用的有以下几种：

bloom filter：类似于哈希表的一种算法，用所有可能的查询条件生成一个bitmap，在进行数据库查询之前会使用这个bitmap进行过滤，如果不在其中则直接过滤，从而减轻数据库层面的压力。guava中有实现BloomFilter算法

空值缓存：一种比较简单的解决办法，在第一次查询完不存在的数据后，将该key与对应的空值也放入缓存中，只不过设定为较短的失效时间，例如几分钟，这样则可以应对短时间的大量的该key攻击，设置为较短的失效时间是因为该值可能业务无关，存在意义不大，且该次的查询也未必是攻击者发起，无过久存储的必要，故可以早点失效。

2.3 缓存雪崩

在普通的缓存系统中一般例如redis、memcache等中，我们会给缓存设置一个失效时间，但是如果所有的缓存的失效时间相同，那么在同一时间失效时，所有系统的请求都会发送到数据库层，db可能无法承受如此大的压力导致系统崩溃。

2.4 解决方案

线程互斥：只让一个线程构建缓存，其他线程等待构建缓存的线程执行完，重新从缓存获取数据才可以，每个时刻只有一个线程在执行请求，减轻了db的压力，但缺点也很明显，降低了系统的qps。

交错失效时间：这种方法时间比较简单粗暴，既然在同一时间失效会造成请求过多雪崩，那我们错开不同的失效时间即可从一定长度上避免这种问题，在缓存进行失效时间设置的时候，从某个适当的值域中随机一个时间作为失效时间即可。关注Java技术栈微信公众号，在后台回复关键字：*缓存*，可以获取更多栈长整理的缓存系列技术干货。

2.5 缓存击穿

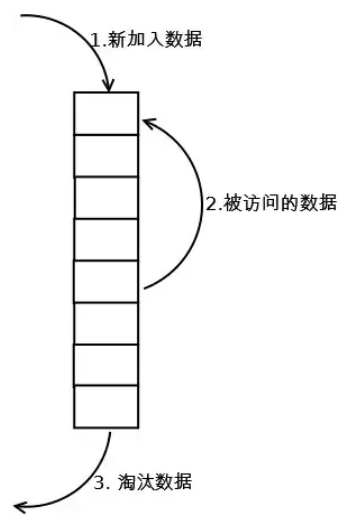
缓存击穿实际上是缓存雪崩的一个特例，大家使用过微博的应该都知道，微博有一个热门话题的功能，用户对于热门话题的搜索量往往在一些时刻会大大的高于其他话题，这种我们成为系统的“热点“，由于系统中对这些热点的数据缓存也存在失效时间，在热点的缓存到达失效时间时，此时可能依然会有大量的请求到达系统，没有了缓存层的保护，这些请求同样的会到达db从而可能引起故障。击穿与雪崩的区别即在于击穿是对于特定的热点数据来说，而雪崩是全部数据。

2.6 解决方案

二级缓存：对于热点数据进行二级缓存，并对于不同级别的缓存设定不同的失效时间，则请求不会直接击穿缓存层到达数据库。

这里参考了阿里双11万亿流量的缓存击穿解决方案，解决此问题的关键在于热点访问。由于热点可能随着时间的变化而变化，针对固定的数据进行特殊缓存是不能起到治本作用的，结合LRU算法能够较好的帮我们解决这个问题。那么LRU是什么，下面粗略的介绍一下，有兴趣的可以点击上面的链接查看.

LRU（Least recently used，最近最少使用）算法根据数据的历史访问记录来进行淘汰数据，其核心思想是“如果数据最近被访问过，那么将来被访问的几率也更高”。最常见的实现是使用一个链表保存缓存数据，如下图所示



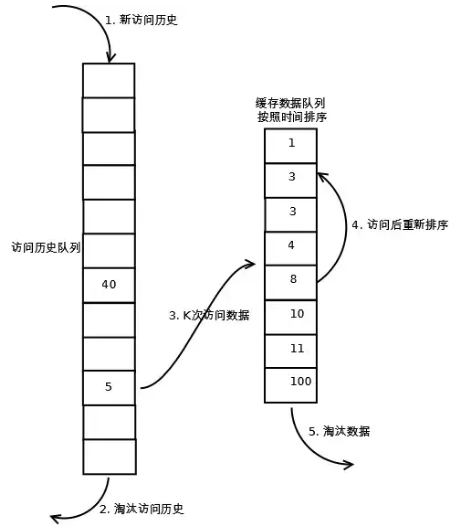
这个链表即是我们的缓存结构，缓存处理步骤为

首先将新数据放入链表的头部

在进行数据插入的过程中，如果检测到链表中有数据被再次访问也就是有请求再次访问这些数据，那么就其插入的链表的头部，因为它们相对其他数据来说可能是热点数据，具有保留时间更久的意义

最后当链表数据放满时将底部的数据淘汰，也就是不常访问的数据

 LRU-K算法 ，其实上面的算法也是该算法的特例情况即LRU-1，上面的算法存在较多的不合理性，在实际的应用过程中采用该算法进行了改进，例如偶然的数据影响会造成命中率较低，比如某个数据即将到达底部即将被淘汰，但由于一次的请求又放入了头部，此后再无该数据的请求，那么该数据的继续存在其实是不合理的，针对这类情况LRU-K算法拥有更好的解决措施。结构图如下所示：



LRU-K需要多维护一个队列或者更多，用于记录所有缓存数据被访问的历史。只有当数据的访问次数达到K次的时候，才将数据放入缓存。当需要淘汰数据时，LRU-K会淘汰第K次访问时间距当前时间最大的数据。关注Java技术栈微信公众号，在后台回复关键字：*缓存*，可以获取更多栈长整理的缓存系列技术干货。

第一步添加数据照样放入第一个队列的头部

如果数据在该队列里访问没有达到K次（该数值根据具体系统qps来定）则会继续到达链表底部直至淘汰；如果该数据在队列中时访问次数达到了K次，那么它会被加入到接下来的2级(具体需要几级结构也同样结合系统分析)链表中，按照时间顺序在2级链表中排列

接下来2级链表中的操作与上面算法相同，链表中的数据如果再次被访问则移到头部，链表满时，底部数据淘汰

相比LRU，LRU-K需要多维护一个队列，用于记录所有缓存数据被访问的历史，所以需要更多的内存空间来用来构建缓存，但优点也很明显，较好的降低了数据的污染率提高了缓存的命中率，对于系统来说可以用一定的硬件成本来换取系统性能也不失为一种办法。

**3.为什么要在分布式系统中使用缓存**

在提高应用程序速度和性能上，每一毫秒都很重要。根据谷歌的一项研究，假如一个网站在3秒钟或更短时间内没有加载成功，会有 53% 的手机用户会离开。

缓存是让分布式应用程序加速的重要技术之一。存储的信息越接近 CPU，访问速度就越快。从 CPU 缓存中加载数据比从 RAM 中加载要快得多，比从硬盘或网络上加载要快得多得多。

要存储经常访问的数据，分布式应用程序需要在多台机器中维护缓存。分布式缓存是降低分布式应用程序延迟、提高并发性和可伸缩性的一种重要策略。

Redis 是一种流行的开源内存数据存储，可用作数据库、缓存或消息代理。由于是从内存而非磁盘加载数据，Redis 比许多传统的数据库解决方案更快。

然而，对开发者来说让 Redis 分布式缓存正确工作是一个巨大挑战。例如，必须谨慎处理本地缓存失效，即替换或删除缓存条目。每次更新或删除存储计算机本地缓存中的信息时，必须更新分布式缓存系统所有计算机内存中的缓存。

好消息是，有一些类似 Redisson 这样的 Redis 框架，可以帮助构建应用程序所需的分布式缓存。

**Redisson 中分布式缓存的三个重要实现：Maps、Spring Cache 和 JCache。**

**3.1. Redisson 分布式缓存**

Redisson 是一个基于 Redis 的框架，用 Java 实现了一个 [Redis](http://mp.weixin.qq.com/s?__biz=MzI3ODcxMzQzMw==&mid=2247488079&idx=1&sn=1ecf7c491e9275dda8bfe0a52376cfe5&chksm=eb539779dc241e6fff288cd248f6d99c9456dea607fa7dfb624fc1597065e01bf4d2f92974fe&scene=21#wechat_redirect) 包装器（wrapper）和接口。Redisson 包含许多常见的 Java 类，例如分布式对象、分布式服务、分布式锁和同步器，以及分布式集合。正如下面即将介绍的，其中一些接口同时支持分布式缓存和本地缓存。

**3.2. Map**

Map 是 Java 最有用的集合之一。Redisson 提供了一个名为 RMap 的 Java Map 实现，支持本地缓存。

如果希望执行多个读操作或网络环回（roundtrip），应使用支持本地缓存的 RMap。通过本地存储 [Map](http://mp.weixin.qq.com/s?__biz=MzI3ODcxMzQzMw==&mid=2247484146&idx=1&sn=4aed1eab1106e0cc6aee09db84f275d7&chksm=eb5387c4dc240ed22b87c0bd6f4aef0ccc653a49758317ec84545ae20871d10b8acd40475261&scene=21#wechat_redirect)数据，RMap 比不启用本地缓存时快45倍。通用分布式缓存使用 RMapCache，本地缓存使用 RLocalCachedMap。

Redis 引擎自身能够执行缓存，不需要在客户端执行代码。然而，虽然本地缓存能显著提高读取速度，但需要由开发人员维护，并且可能需要一些开发工作。Redisson 为开发人员提供了 RLocalCachedMap 对象，让本地缓存实现起来更容易。

下面的代码展示了如何初始化 RMapCache 对象：

RMapCache<String, SomeObject> map = redisson.getMapCache("anyMap");  
map.put("key1", new SomeObject(), 10, TimeUnit.MINUTES, 10, TimeUnit.SECONDS);

上面的代码将字符串 "key1" 放到 RMapCache 中，并与 SomeObject() 关联。然后它指定了两个参数，TTL设为10分钟、最大空闲时间10秒。

当不再需要时，应销毁 RMapCache 对象：

map.destroy();

Redisson 关闭后不用再做销毁操作。

**3.3. Spring Cache**

Spring 是一个用于构建企业级 Web 应用程序的 Java 框架，也提供了缓存支持。

Redisson 包含了 Spring 缓存功能，提供两个对象： RedissonSpringCacheManager 和 RedissonSpringLocalCachedCacheManager。 RedissonSpringLocalCachedCacheManager 支持本地缓存。

下面是一个 RedissonSpringLocalCachedCacheManager 对象的示例配置：

@Configuration  
@ComponentScan  
@EnableCaching  
public static class Application {  
    @Bean(destroyMethod="shutdown")  
    RedissonClient redisson() throws IOException {  
        Config config = new Config();  
        config.useClusterServers()  
                .addNodeAddress("127.0.0.1:7004", "127.0.0.1:7001");  
        return Redisson.create(config);  
    }  
    @Bean  
    CacheManager cacheManager(RedissonClient redissonClient) {  
        Map<String, CacheConfig> config = new HashMap<String, CacheConfig>();  
        *// 新建 "testMap" 缓存：ttl=24分钟，maxIdleTime=12分钟*  
        config.put("testMap", new CacheConfig(24\*60\*1000, 12\*60\*1000));  
        return new RedissonSpringCacheManager(redissonClient, config);  
    }  
}

此外，还可以读取 JSON 或 YAML 文件配置 RedissonSpringCacheManager。

与 RMaps 一样，每个 RedissonSpringCacheManager 实例都有两个重要参数: ttl（生存时间）和 maxIdleTime。如果这些参数设为0或者没有定义，那么数据将无限期地保留在缓存中。

**3.4. JCache**

JCache 是一个 Java 缓存 API，允许开发人员从缓存临时存储、检索、更新和删除对象。

Redisson 提供了 Redis 的 JCache API 实现。下面是在 Redisson 中使用默认配置调用 JCache API 的示例：

MutableConfiguration<String, String> config = new MutableConfiguration<>();  
CacheManager manager = Caching.getCachingProvider().getCacheManager();  
Cache<String, String> cache = manager.createCache("namedCache", config);

此外，还可以使用自定义配置文件、Redisson Config 对象或 Redisson RedissonClient 对象配置 JCache。例如，下面的代码使用自定义 Redisson 配置来调用 JCache：

MutableConfiguration<String, String> jcacheConfig = new MutableConfiguration<>();  
Config redissonCfg = ...  
Configuration<String, String> config = RedissonConfiguration.fromConfig(redissonCfg, jcacheConfig);  
CacheManager manager = Caching.getCachingProvider().getCacheManager();  
Cache<String, String> cache = manager.createCache("namedCache", config);