Redis 缓存穿透问题及解决方案

上周在工作中遇到了一个问题场景，即查询商品的配件信息时（商品：配件为 1：N 的关系），如若商品并未配置配件信息，则查数据库为空，且不会加入缓存，这就会导致，下次在查询同样商品的配件时，由于缓存未命中，则仍旧会查底层数据库，所以缓存就一直未起到应有的作用，当并发流量大时，会很容易把 DB 打垮。

缓存穿透问题

缓存穿透是指**查询一个根本不存在的数据**，缓存层和存储层都不会命中，通常出于容错的考虑，如果从存储层查不到数据则不写入缓存层。
一般对于未命中的数据我们是按照如下方式进行处理的：

1.缓存层不命中。
2.存储层不命中，不将空结果写回缓存。
3.返回空结果。

```
/**
 * 缓存穿透问题：
 * 在数据库层没有查到数据，未存入缓存，
 * 则下次查询同样的数据时，还会查库。
 *
 * @param id
 * @return
 */
private Object getObjectById(Integer id) {
    // 从缓存中获取数据
    Object cacheValue = cache.get(id);
    if (cacheValue != null) {
        return cacheValue;
    }
    // 从数据库中获取
    Object storageValue = storage.get(id);
    // 如果这里按照id查询DB为空，那么便会出现缓存穿透
    if (storageValue != null) {
        cache.set(id, storageValue);
    }
    return storageValue;
}
```

缓存穿透将导致不存在的数据每次请求都要到存储层去查询，失去了缓存保护后端存储的意义。
缓存穿透问题可能会使后端存储负载加大，由于很多后端存储不具备高并发性，甚至可能造成后端存储宕掉。

方案一：缓存空对象

```java
/**
 * 缓存空对象：
 * 此种方式存在漏洞，不经过判断就直接将Null对象存入到缓存中，
 * 如果恶意制造不存在的id那么，缓存中的键值就会很多，恶意攻击时，很可能会被打爆，所以需设置较短的过期时间。
 *
 * @param id
 * @return
 */
public Object getObjectInclNullById(Integer id) {
    // 从缓存中获取数据
    Object cacheValue = cache.get(id);
    // 缓存为空
    if (cacheValue != null) {
        // 从数据库中获取
        Object storageValue = storage.get(key);
        // 缓存空对象
        cache.set(key, storageValue);
        // 如果存储数据为空，需要设置一个过期时间(300秒)
        if (storageValue == null) {
            // 必须设置过期时间，否则有被攻击的风险
            cache.expire(key, 60 * 5);
        }
        return storageValue;
    }
    return cacheValue;
}
```

缓存空对象会有一个必须考虑的问题：

空值做了缓存，意味着缓存层中存了更多的键，需要更多的内存空间（如果是攻击，问题更严重），比较有效的方法是针对这类数据设置一个较短的过期时间，让其自动剔除。

方案二：布隆过滤器拦截

布隆过滤器介绍

概念：

布隆过滤器（英语：Bloom Filter）是 1970 年由布隆提出的。它实际上是一个很长的二进制向量和一系列随机映射函数。布隆过滤器可以用于检索一个元素是否在一个集合中。它的优点是空间效率和查询时间都远远超过一般的算法，**缺点是有一定的误识别率和删除困难。**

如果想判断一个元素是不是在一个集合里，一般想到的是将集合中所有元素保存起来，然后通过比较确定。链表、树、散列表（又叫哈希表，Hash table）等等数据结构都是这种思路。但是随着集合中元素的增加，我们需要的存储空间越来越大。同时检索速度也越来越慢，上述三种结构的检索时间复杂度分别为 O(n),O(log n),O(n/k)

布隆过滤器的原理是，当一个元素被加入集合时，通过 K 个散列函数将这个元素映射成一个位数组中的 K 个点，把它们置为 1。检索时，我们只要看看这些点是不是都是 1 就（大约）知道集合中有没有它了：如果这些点有任何一个 0，则被检元素一定不在；如果都是 1，则被检元素很可能在。这就是布隆过滤器的基本思想。

示例：

google guava 包下有对布隆过滤器的封装，BloomFilter。

```
import com.google.common.hash.BloomFilter;
import com.google.common.hash.Funnels;

public class BloomFilterTest {

    // 初始化一个能够容纳10000个元素且容错率为0.01布隆过滤器
    private static final BloomFilter<Integer> bloomFilter = BloomFilter.create(Funnels.integerFunnel(), 10000, 0.01);

    /**
     * 初始化布隆过滤器
     */
    private static void initLegalIdsBloomFilter() {
        // 初始化10000个合法Id并加入到过滤器中
        for (int legalId = 0; legalId < 10000; legalId++) {
            bloomFilter.put(legalId);
        }
    }

    /**
     * id是否合法有效，即是否在过滤器中
     *
     * @param id
     * @return
     */
    public static boolean validateIdInBloomFilter(Integer id) {
        return bloomFilter.mightContain(id);
    }

    public static void main(String[] args) {
        // 初始化过滤器
        initLegalIdsBloomFilter();
        // 误判个数
        int errorNum=0;
        // 验证从10000个非法id是否有效
        for (int id = 10000; id < 20000; id++) {
            if (validateIdInBloomFilter(id)){
                // 误判数
                errorNum++;
            }
        }
        System.out.println("judge error num is : " + errorNum);
    }
}
```

布隆过滤器拦截

设置过期时间，让其自动过期失效，这种在很多时候不是最佳的实践方案。

我们可以提前将真实正确的商品 Id，在添加完成之后便加入到过滤器当中，每次再进行查询时，先确认要查询的 Id 是否在过滤器当中，如果不在，则说明 Id 为非法 Id，则不需要进行后续的查询步骤了。

```java
/**
 * 防缓存穿透的：布隆过滤器
 *
 * @param id
 * @return
 */
public Object getObjectByBloom(Integer id) {
    // 判断是否为合法id
    if (!bloomFilter.mightContain(id)) {
        // 非法id,则不允许继续查库
        return null;
    } else {
        // 从缓存中获取数据
        Object cacheValue = cache.get(id);
        // 缓存为空
        if (cacheValue == null) {
            // 从数据库中获取
            Object storageValue = storage.get(id);
            // 缓存空对象
            cache.set(id, storageValue);
        }
        return cacheValue;
    }
}
```

文字学习太枯燥？光看不练学不懂？一个人学习没有氛围？

今晚八点，咕泡学院在腾讯课堂有一个技术分享讲座，主题就是【Redis 缓存穿透解决方案与原理】，由咕泡学院金牌讲师【青山老师】倾情奉献。

内容提要：

1.为什么会发生缓存穿透？

2.解决缓存穿透的思路

3.布隆过滤器原理与项目实战

4.布隆过滤器的不足与升级