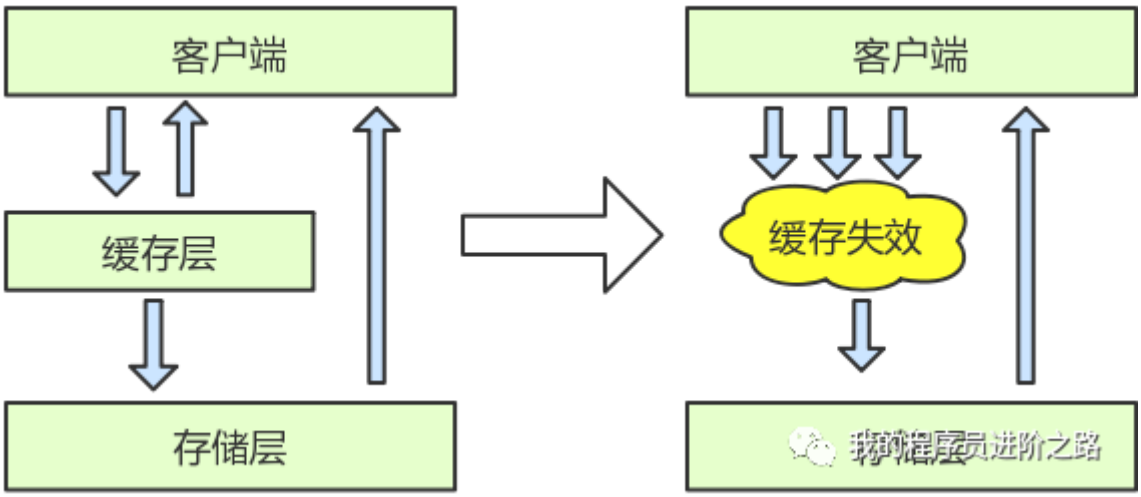
通过双key解决缓存并发问题

IT人的职场进阶 2019-10-29

我们在使用缓存的时候，不管Redis或者是Memcached，基本上都会遇到以下3个问题：缓存穿透、缓存并发、缓存集中失效。这篇文章主要针对【缓存并发】问题展开讨论，并给出具体的解决方案。

1.什么是缓存并发？

在高并发的访问下，当某个缓存处于过期失效的时间点时，极有可能出现多个进程同时查询该缓存（该缓存是业务场景中非常"热点"的数据，比如首页的缓存数据）。因为查询DB并重新缓存需要一定的时间，而瞬时并发非常高，如果此时缓存失效了，这些并发请求都会直接访问DB，从而导致DB服务器的CPU或者内存负载过高，服务能力下降甚至宕机，此问题即缓存并发问题。



缓存并发问题在微服务架构下凸显更加严重，比如某个基础服务A因为上述问题出现不可用，进而导致依赖A服务的B、C服务也不可用，而B服务的不可用又导致服务E、F不可用，不可用的服务就像滚雪球一样越滚越大，最终导致系统出现严重故障，此现象我们称之为雪崩效应。

注意缓存并发和缓存集中失效的区别在于：缓存并发指的是某一个热点key的失效，而缓存集中失效则是一批key同时失效，两者都可能导致雪崩问题。

2.如何解决？

针对该问题，存在以下三种解决方案：

1. 加锁：在缓存失效后，通过加锁的方式只允许一个线程查询数据和写缓存，其他线程如果发现有锁就等待，等解锁后再返回数据。该方案会造成部分请求等待。

2. 二级缓存：A1为原始缓存，A2为拷贝缓存。A1失效时，可以访问A2，其中A1的缓存失效时间设置为短期（比如5min），A2的缓存失效时间设置为长期（比如1天）。如果缓存value很大，此方案的缓存空间利用率低。

3. 双key：思路和方案2类似，不同的是双key分别缓存过期时间（key-time)和缓存数据（key-data），其中(key-time)的缓存失效时间设置为短期（比如5min），(key-data)的缓存失效时间设置为长期（比如1天）。当第一个线程发现 key-time 过期不存在时，则先更新key-time，然后去查询数据库并更新key-data 的值；当其他线程来获取数据时，虽然第一个线程还没有从数据库查询完毕并更新缓存，但发现key-time存在，会直接读取缓存的旧数据返回。和二级缓存的方案对比，该方案的缓存空间利用率高。

3.双key方案的示例代码

1. 写缓存的示例代码

```
public static boolean set(String key, String value, int seconds) {
   Jedis jedis = null;
   try {
      jedis = jedisPool.getResource();
      if (seconds > 0){
         // 添加数据缓存，缓存有效时间 = 真实时间 + 1 天
         jedis.set(key, seconds + 60 * 60 * 24, value);

         // 添加过期时间缓存，缓存有效时间 = 真实时间
         jedis.set("lock_" + key, seconds, System.currentTimeMillis() + "");
      } else {
         jedis.set(key, value);
         jedis.set("lock_" + key, System.currentTimeMillis() + "");
       }

       return true;
   } catch (JedisException e) {
      if (jedis != null) {
         returnBrokenResource(jedis);
         jedis = null;
      }
      throw e;
   } finally {
      if (jedis != null) {
         returnResource(jedis);
      }
   }
}
```

2. 读缓存的示例代码

```
public static String get(String key) {
   Jedis jedis = null;
   try {
      jedis = jedisPool.getResource();
```

```
5
6         // 缓存过期 && 获取锁成功，setnx:原子操作
7         if (jedis.setnx("lock_" + key, System.currentTimeMillis() + "") == 1) {
8             /**
9              * 将锁的失效时间设为60s，在60s内若查询数据库成功，则更新锁的失效时间=缓存时间
10             * 如果60s内出现异常，则60s后第一个请求又会去访问数据库
11             * 返回null表示没有查询到数据 ，外层代码会通过数据库获取数据并设置缓存
12             */
13            jedis.expire("lock_" + key, 60);
14            return null;
15        } else{
16            // 缓存未过期或者缓存过期但获取锁失败，则返回旧数据
17            return jedis.get(key);
18        }
19    } catch (JedisException e) {
20        if (jedis != null) {
21            returnBrokenResource(jedis);
22            jedis = null;
23        }
24        throw e;
25    } finally {
26        if (jedis != null) {
27            returnResource(jedis);
28        }
29    }
30 }
```

文章已于2020-07-14修改