链滴

guava 布隆过滤器解决缓存穿透

作者：XieWeiZM

原文链接：https://ld246.com/article/1567144815067

来源网站：链滴

前言

在博客系统中，为了提升响应速度，加入了 Redis 缓存，把文章主键 ID 作为 key 值去缓存查询，如不存在对应的 value，就去数据库中查找 。这个时候，如果请求的并发量很大，就会对后端的数据库务造成很大的压力。

造成原因

- 业务自身代码或数据出现问题
- 恶意攻击、爬虫造成大量空的命中，会对数据库造成很大压力

案例分析

由于文章的地址是这样子的：

```
https://blog.52itstyle.top/49.html
```

大家很容易猜出，是不是还有 50、51、52 甚至是十万+？如果是正儿八经的爬虫，可能会读取你的页数。但是有些不正经的爬虫或者人，还真以为你有十万+博文，然后就写了这么一个脚本。

```
for num in range(1,1000000):
//爬死你，开100个线程
```

解决方案

设置布隆过滤器，预先将所有文章的主键 ID 哈希到一个足够大的 BitMap 中，每次请求都会经过 Bitap 的拦截，如果 Key 不存在，直接返回异常。这样就避免了对 Redis 缓存以及底层数据库的查询压。

这里我们使用谷歌开源的第三方工具类来实现：

```
<!-- 使用guava -->
<dependency>
<groupId>com.google.guava</groupId>
<artifactId>guava</artifactId>
<version>25.1-jre</version>
</dependency>
```

编写布隆过滤器：

```
/**
 * 布隆缓存过滤器
 */
@Component
public class BloomCacheFilter {

    public static BloomFilter<Integer> bloomFilter = null;

    @Autowired
    private DynamicQuery dynamicQuery;
    /**
```

```
     * 初始化
     */
    @PostConstruct
    public void init(){
        String nativeSql = "SELECT id FROM blog";
        List<Object> list = dynamicQuery.query(nativeSql,new Object[]{});
        bloomFilter = BloomFilter.create(Funnels.integerFunnel(), list.size());
        list.forEach(blog ->bloomFilter.put(Integer.parseInt(blog.toString())));
    }

    /**
     * 判断key是否存在
     * @param key
     * @return
     */
    public static boolean mightContain(long key){
        return bloomFilter.mightContain((int)key);
    }
}
```

然后，每一次查询之前做一次 Key 值校验：

```
/**
 * 博文
 */
@RequestMapping("{id}.shtml")
public String page(@PathVariable("id") Long id, ModelMap model) {
    if(BloomCacheFilter.mightContain(id)){
        Blog blog = blogService.getById(id);
        model.addAttribute("blog",blog);
        return  "article";
    }else{
        return  "error";
    }
}
```

效率

那么，在数据量很大的情况下，效率如何呢？我们来做个实验，以 100W 为基数。

```
public static void main(String[] args) {
        int capacity = 1000000;
        int key = 6666;
        BloomFilter<Integer> bloomFilter = BloomFilter.create(Funnels.integerFunnel(), capacity)

        for (int i = 0; i < capacity; i++) {
            bloomFilter.put(i);
        }
        /**返回计算机最精确的时间，单位纳妙 */
        long start = System.nanoTime();
        if (bloomFilter.mightContain(key)) {
            System.out.println("成功过滤到" + key);
        }
        long end = System.nanoTime();
```

```
        System.out.println("布隆过滤器消耗时间:" + (end - start));

}
```

布隆过滤器消耗时间:281299，约等于 0.28 毫秒，匹配速度是不是很快?

错判率

万事万物都有所均衡，既然效率如此之高，肯定其它方面定有所牺牲，通过测试我们发现，过滤器有 % 的错判率，也就是说，本没有的文章，有可能会访问到!

```
 public static void main(String[] args) {
        int capacity = 1000000;
        BloomFilter<Integer> bloomFilter = BloomFilter.create(Funnels.integerFunnel(), capacity)

        for (int i = 0; i < capacity; i++) {
            bloomFilter.put(i);
        }
        int sum = 0;
        for (int i = capacity + 20000; i < capacity + 30000; i++) {
            if (bloomFilter.mightContain(i)) {
                sum ++;
            }
        }
        //0.03
        DecimalFormat df=new DecimalFormat("0.00");//设置保留位数
        System.out.println("错判率为:" + df.format((float)sum/10000));
}
```

通过源码阅读，发现 3% 的错判率是系统写死的。

```
public  static  <T>  BloomFilter<T> create(Funnel<?  super T> funnel,  long expectedInsertion
)  {
    return create(funnel, expectedInsertions,  0.03D);
 }
```

当然我们也可以通过传参，降低错判率。测试了一下，查询速度稍微有一丢丢降低，但也只是零点几秒级的而已。

```
BloomFilter<Integer> bloomFilter =  BloomFilter.create(Funnels.integerFunnel(), capacity,0.01)
```

那么如何做到零错判率呢？答案是不可能的，布隆过滤器，错判率必须大于零。为了保证文章 100% 访问率，正常情况下，我们可以关闭布隆校验，只有才突发情况下开启。比如，可以通过阿里的动态数配置 Nacos 实现。

```
@NacosValue(value = "${bloomCache:false}", autoRefreshed = true)
private boolean bloomCache;
//省略部分代码
if(bloomCache||BloomCacheFilter.mightContain(id)){
     Blog blog = blogService.getById(id);
     model.addAttribute("blog",blog);
     return  "article";
  }else{
```

```
    return "error";
}
```

小结

缓存穿透大多数情况下都是恶意攻击导致的空命中率。虽然十万博客还没有被百度收录，每天也就寥的几十个IP，但是梦想还是有的，万一实现了呢？所以，还是要做好准备的！