

属性是spu级别的比如：电脑有内存、cpu、硬盘

规格是sku级别的比如：多大内存、什么颜色的、什么尺寸的（价格也有不同 比如iphone土豪金的配置就要比其他颜色相同的配置要高 那这个颜色就是属于规格）

缓存：

后台问题描述

访问数据库查询商品信息：

读多写少

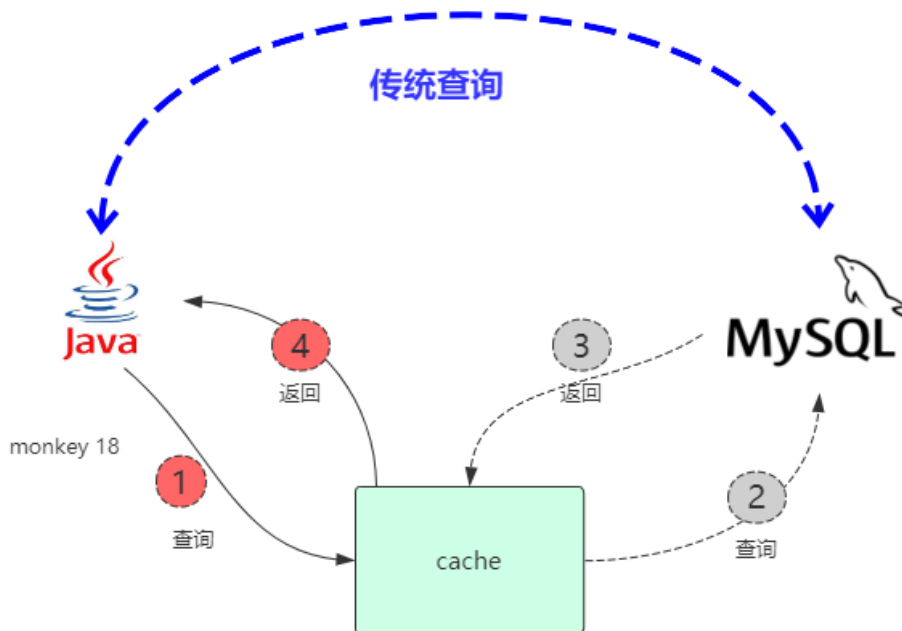
```
1  /**
2   * 获取商品详情信息
3   *
4   * @param id 产品ID
5   */
6  public PmsProductParam getProductInfo(Long id) {
7      PmsProductParam productInfo = null;
8      productInfo = portalProductDao.getProductInfo(id);
9      if (null == productInfo) {
10         return productInfo;
11     }
12     FlashPromotionParam promotion =
flashPromotionProductDao.getFlashPromotion(id);
13     if (!ObjectUtils.isEmpty(promotion)) {
14         productInfo.setFlashPromotionCount(promotion.getRelation().get(0).getFlashP
romotionCount());
15         productInfo.setFlashPromotionLimit(promotion.getRelation().get(0).getFlashP
romotionLimit());
16         productInfo.setFlashPromotionPrice(promotion.getRelation().get(0).getFlashP
romotionPrice());
17         productInfo.setFlashPromotionRelationId(promotion.getRelation().get(0).getI
d());
18         productInfo.setFlashPromotionEndDate(promotion.getEndDate());
19         productInfo.setFlashPromotionStartDate(promotion.getStartDate());
20         productInfo.setFlashPromotionStatus(promotion.getStatus());
21     }
22     return productInfo;
23 }
```

前端静态化页面：

我们发现上节课通过对数据进行静态化，也是有很多问题的，比如我们商品如果过多，freemarker模板一定修改之后，我们所有的商品都需要重新再次生产静态化，这个工作量实在是太大了。

引入缓存:

缓存作用:



实战:

```
1 /**
2  * 获取商品详情信息
3  *
4  * @param id 产品ID
5  */
6 public PmsProductParam getProductInfo(Long id) {
7     PmsProductParam productInfo = null;
8
9     productInfo = redisOpsUtil.get(RedisKeyPrefixConst.PRODUCT_DETAIL_CACHE +
10 id, PmsProductParam.class);
11     if (productInfo != null) {
12         return productInfo;
13     }
14     productInfo = portalProductDao.getProductInfo(id);
15     if (null == productInfo) {
16         return null;
17     }
18 }
```

```

17  FlashPromotionParam promotion =
flashPromotionProductDao.getFlashPromotion(id);
18  if (!ObjectUtils.isEmpty(promotion)) {
19      productInfo.setFlashPromotionCount(promotion.getRelation().get(0).getFlashP
romotionCount());
20      productInfo.setFlashPromotionLimit(promotion.getRelation().get(0).getFlashP
romotionLimit());
21      productInfo.setFlashPromotionPrice(promotion.getRelation().get(0).getFlashP
romotionPrice());
22      productInfo.setFlashPromotionRelationId(promotion.getRelation().get(0).getI
d());
23      productInfo.setFlashPromotionEndDate(promotion.getEndDate());
24      productInfo.setFlashPromotionStartDate(promotion.getStartDate());
25      productInfo.setFlashPromotionStatus(promotion.getStatus());
26  }
27  redisOpsUtil.set(RedisKeyPrefixConst.PRODUCT_DETAIL_CACHE + id,
productInfo, 3600, TimeUnit.SECONDS);
28  return productInfo;
29  }

```

有两种：

1、最终一致性方案：

设置超时时间来解决

```

1  redisOpsUtil.set(RedisKeyPrefixConst.PRODUCT_DETAIL_CACHE+id,productInfo,360,
imeUnit.SECONDS);

```

2、实时一致性方案：

课程讲到 交易canal binlog

两个问题(高并发)、压缩的问题》减少内存

名称：

聚合报告

注释：

所有数据写入一个文件

文件名

浏览...

显示日志内容：

☐ 仅错误日志

☐ 仅成功日志

Label	# 样本	平均值	最小值	最大值	标准偏差	异常 %	吞吐量	接收 KBlsec	发送 KBlsec	平均字节
产品详情多级缓存优化	1000	658	474	1191	167.79	0.00%	630.5/sec	2233.29	112.06	
总体	1000	658	474	1191	167.79	0.00%	630.5/sec	2233.29	112.06	

现在有什么问题？

```

: ==> Parameters: 20(Long)
: ==> Parameters: 26(Long)
: <==      Total: 0
: <==      Total: 0
: set productId:PmsProductParam [Hash = 10265616, id=26, bran
: set productId:PmsProductParam [Hash = 836310919, id=26, bra
: <==      Total: 0
: set productId:PmsProductParam [Hash = 505299489, id=26, bra
tInfo : ==> Parameters: 26(Long)
: ==> Preparing: SELECT sp.*, sfp.id r_id, sfp.flash_promoti
tInfo : ==> Preparing: SELECT *,l.id ladder_id,l.product_id ladder
tInfo : ==> Parameters: 26(Long)
: ==> Parameters: 26(Long)
: set productId:PmsProductParam [Hash = 1782115467, id=26, br
: <==      Total: 0
: set productId:PmsProductParam [Hash = 1351528149, id=26, br
tInfo : <==      Total: 12
: ==> Preparing: SELECT sp.*, sfp.id r_id, sfp.flash_promoti
: ==> Parameters: 26(Long)
tInfo : <==      Total: 12
tInfo : ==> Preparing: SELECT *,l.id ladder_id,l.product_id ladder
tInfo : ==> Parameters: 26(Long)

```

跟我们预期只set一次redis 是有出入，为何会这样子了？**并发问题**

当我第二次再去访问，此时此刻没有日志输出，说明全部走了缓存：

并发问题：《并发编程》《并发问题》锁的方式来实现 java并发 加锁方式（不适合《特殊》分布式）

分布式锁：redis、zookeeper

图灵微商城-Jmeter压测.jmx (C:\Users\ym\Desktop\图灵微商城-Jmeter压测.jmx) - Apache JMeter (5.3)

文件 编辑 查找 运行 选项 工具 帮助

产品详情页压测-多级缓存优化
产品详情页压测-无需登录
产品详情多级缓存优化
聚合报告
查看结果树
汇总报告
订单业务接口压测-需要登录

汇总报告
名称: 聚合报告
注释:
所有数据写入一个文件
文件名: 确定 显示日志内容: ☐ 仅错误日志 ☐ 仅成功日志

Label	# 样本	平均值	最小值	最大值	标准偏差	异常 %	吞吐量	接收 KB/sec	发送 KB/sec
产品详情多级...	1000	29	14	91	7.78	0.00%	862.8/sec	3056.08	153.35
总计	1000	29	14	91	7.78	0.00%	862.8/sec	3056.08	153.35

QPS立马就提高了很多。

加入分布式锁:

```

1 <!--加入redisson-->
2 <dependency>
3   <groupId>org.redisson</groupId>
4   <artifactId>redisson</artifactId>
5   <version>3.6.5</version>
6 </dependency>

```

缓存应用场景：

- 1、访问量大、QPS高、更新频率不是很高的业务
- 2、数据一致性要求不高

缓存不足：

缓存击穿问题（热点数据单个key）：

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，需要考虑一个问题：缓存被“击穿”的问题，这个和缓存雪崩的区别在于这里针对某一key缓存，前者则是很多key。



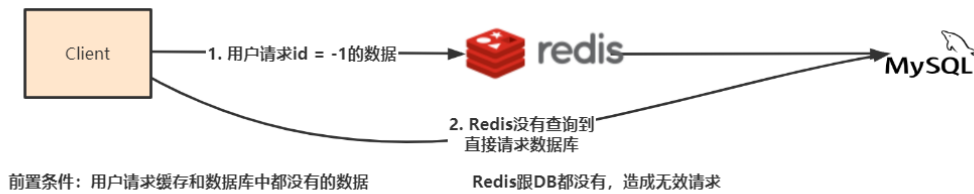
解决方案：

- 1.加锁，在未命中缓存时，通过加锁避免大量请求访问数据库
- 2.不允许过期。物理不过期，也就是不设置过期时间。而是逻辑上定时在后台异步的更新数据。
- 3.采用二级缓存。L1缓存失效时间短，L2缓存失效时间长。请求优先从L1缓存获取数据，如果未命中，则加锁，保证只有一个线程去数据库中读取数据然后再更新到L1和L2中。然后其他线程依然在L2缓存获取数据。

缓存穿透问题（恶意攻击、访问不存在数据）：

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的，并且出于容错考虑，如果从存储层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存储层去查询，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。

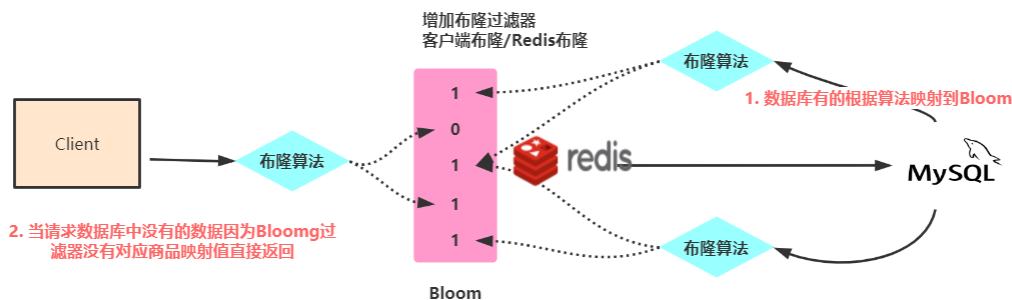
缓存穿透



解决方案：有很多种方法可以有效地解决缓存穿透问题

1、最常见的则是采用布隆过滤器，将所有可能存在的数据哈希到一个足够大的bitmap中，一个一定不存在的数据会被这个bitmap拦截掉，从而避免了对底层存储系统的查询压力。

2、另外也有一个更为简单粗暴的方法（我们采用的就是这种），如果一个查询返回的数据为空（不管是数据不存在，还是系统故障），我们仍然把这个空结果进行缓存，但它的过期时间会很短，最长不超过五分钟。

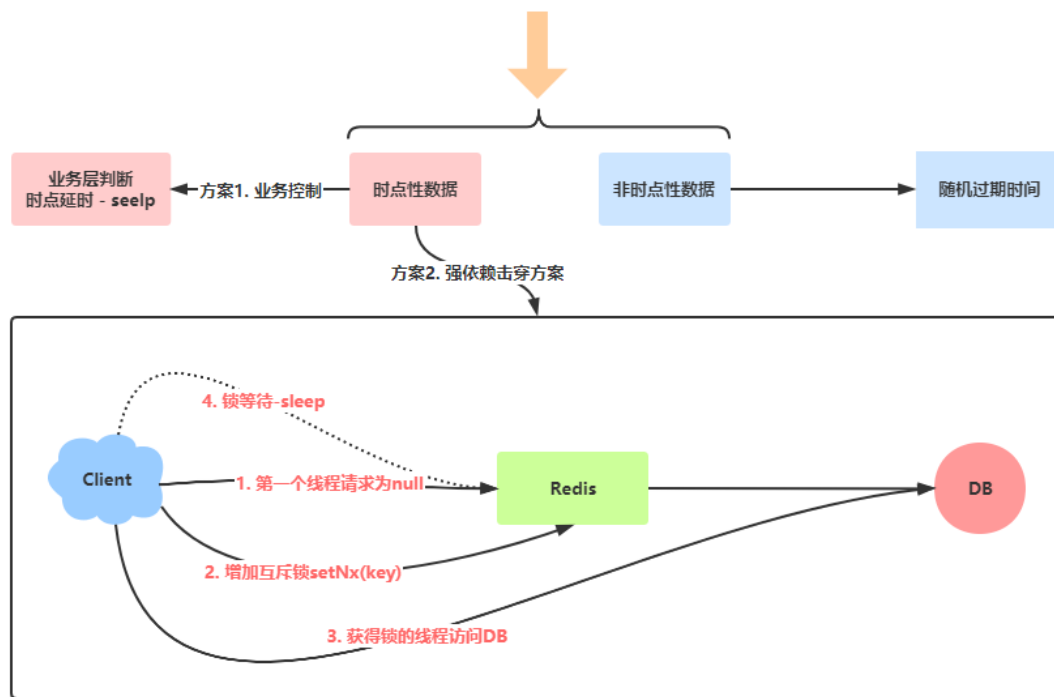


缓存雪崩（同一时间失效，并发量大）：

缓存雪崩是指在我们设置缓存时采用了相同的过期时间，导致缓存在某一时刻同时失效，请求全部转发到DB，DB瞬时压力过重雪崩。

缓存雪崩





解决方案：

1、缓存失效时的雪崩效应对底层系统的冲击非常可怕。大多数系统设计者考虑用加锁或者队列的方式保证缓存的单线程（进程）写，从而避免失效时大量的并发请求落到底层存储系统上。这里分享一个简单方案就时讲缓存失效时间分散开，比如我们可以在原有的失效时间基础上增加一个随机值，比如1-5分钟随机，这样每一个缓存的过期时间的重复率就会降低，就很难引发集体失效的事件。

2、**事前**：这种方案就是在发生雪崩前对缓存集群实现高可用，如果是使用 Redis，可以使用 主从+哨兵，Redis Cluster 来避免 Redis 全盘崩溃的情况。

3、**事中**：使用 Hystrix进行限流 & 降级，比如一秒来了5000个请求，我们可以设置假设只能有一秒 2000个请求能通过这个组件，那么其他剩余的 3000 请求就会走限流逻辑。然后去调用我们自己开发的降级组件（降级），比如设置的一些默认值呀之类的。以此来保护最后的 MySQL 不会被大量的请求给打死。

4、**事后**：开启Redis持久化机制，尽快恢复缓存集群

缓存和数据库双写一致性问题：

一致性问题分布式常见问题，还可以再分为最终一致性和强一致性。数据库和缓存双写，就必然会存在不一致的问题。

答这个问题，先明白一个前提。就是如果对数据有强一致性要求，不能放缓存。

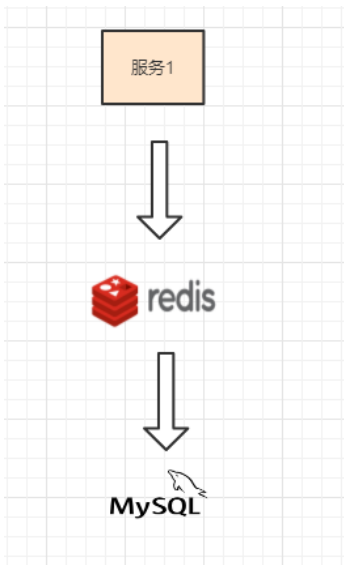
我们所做的一切，只能保证最终一致性。另外，我们所做的方案其实从根本上来说，只能说降低不一致发生的概率，无法完全避免。因此，有强一致性要求的数据，不能放缓存。

zk>临时顺序节点》原子性 线程创建如果可以创建成功，是否第一个 拿到了锁

引入分布式锁：

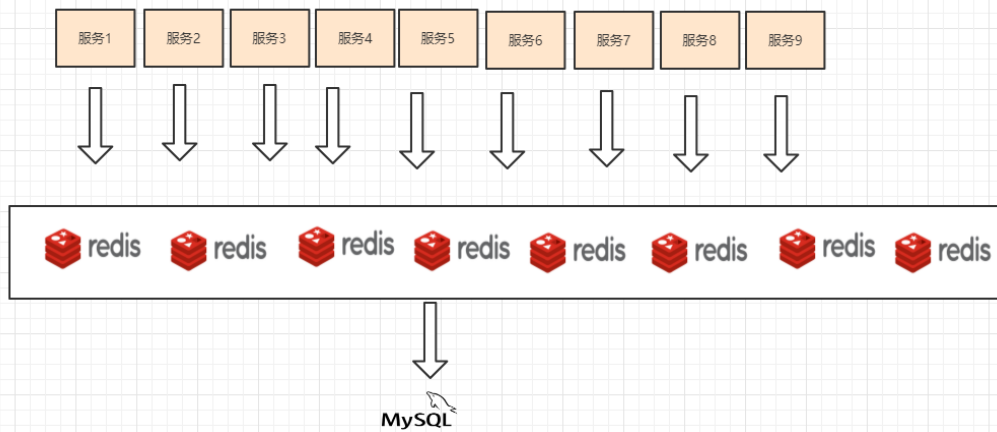
```
1 <!--加入redisson-->
2 <dependency>
3   <groupId>org.redisson</groupId>
4   <artifactId>redisson</artifactId>
5   <version>3.6.5</version>
6 </dependency>
7
8 @Bean
9 public RedissonClient redissonClient(){
10   Config config = new Config();
11   config.useSingleServer().setAddress("redis://tlshop.com:6379").setPassword("123456").setDatabase(1);
12   return Redisson.create(config);
13 }
```

单机加锁：



分布式加锁：

同一时间同一个数据请求过来，比如set 100 value: (1 2 3)



```
1  /**
2   * 获取商品详情信息
3   *
4   * @param id 产品ID
5   */
6  public PmsProductParam getProductInfo(Long id) {
7      PmsProductParam productInfo = null;
8      productInfo = redisOpsUtil.get(RedisKeyPrefixConst.PRODUCT_DETAIL_CACHE +
9      id, PmsProductParam.class);
10     if (productInfo != null) {
11         log.info("get redis productId:" + productInfo);
12         return productInfo;
13     }
14     try {
15         if (zkLock.lock(lockPath + "_" + id)) {
16             productInfo = portalProductDao.getProductInfo(id);
17             if (null == productInfo) {
18                 return null;
19             }
20             checkFlash(id, productInfo);
21             log.info("set db productId:" + productInfo);
22             //缓存失效时间随机
23             redisOpsUtil.set(RedisKeyPrefixConst.PRODUCT_DETAIL_CACHE + id,
24             productInfo, 3600, TimeUnit.SECONDS);
25         } else {
26             //问题: 返回为null
27             productInfo = redisOpsUtil.get(RedisKeyPrefixConst.PRODUCT_DETAIL_CACHE + i
28             d, PmsProductParam.class);
29         }
30     } finally {
31         log.info("unlock :" + productInfo);
32     }
33 }
```

```

29  zkLock.unlock(lockPath + "_" + id);
30  }
31  return productInfo;
32  }
33
34  private void checkFlash(Long id, PmsProductParam productInfo) {
35      FlashPromotionParam promotion =
flashPromotionProductDao.getFlashPromotion(id);
36      if (!ObjectUtils.isEmpty(promotion)) {
37          productInfo.setFlashPromotionCount(promotion.getRelation().get(0).getFlashP
romotionCount());
38          productInfo.setFlashPromotionLimit(promotion.getRelation().get(0).getFlashP
romotionLimit());
39          productInfo.setFlashPromotionPrice(promotion.getRelation().get(0).getFlashP
romotionPrice());
40          productInfo.setFlashPromotionRelationId(promotion.getRelation().get(0).getI
d());
41          productInfo.setFlashPromotionEndDate(promotion.getEndDate());
42          productInfo.setFlashPromotionStartDate(promotion.getStartDate());
43          productInfo.setFlashPromotionStatus(promotion.getStatus());
44      }

```

我们发现在高并发下增加了分布式锁可以解决刚才那个问题，但是也降低了qps，所以这儿还是要根据需求而定

分布式锁原理：

zk图

<https://www.processon.com/view/link/6044dcb85653bb620cda200c>

文档：03秒杀系统-商品详情页多级缓存实战二.....

链接：http://note.youdao.com/noteshare?

id=98ceab0cd09e3d2115e711a43cf8eba5&sub=BE3726C2F6324C40B02EB3BAE679FD08