理论

Redis 缓存问题

在高并发的业务场景下，数据库大多数情况下都是用户并发访问最薄弱的环节。所以，就需要使用 Redis 做一个缓存操作，让请求先访问到 Redis ，而不是直接访问 MySQL 等数据库。这样可以大大缓解数据库的压力。

- 缓存穿透
- 缓存击穿
- 缓存雪崩
- 缓存污染（或者满了）
- 缓存和数据库一致性

缓存穿透

问题来源

缓存穿透是指**缓存和数据库中都没有的数据**，**而用户不断发起请求。**由于缓存是不命中时被动写的，而且出于容错考虑，如果从 数据库 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 数据库 去查询，失去了缓存的意义。

在流量大的时候，可能数据库就挂掉了，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。

案例

发起 id 为“-1”的数据 或者 id 为特别大不存在的数据，这时的用户很可能是攻击者，攻击会导致数据库压力过大。

解决方案

- 接口层增加校验。如用户鉴权校验，id 做基础校验，id <= 0 的直接拦截。
- 从缓存取不到的数据，在数据库中也没有取到，这时也可以将 key-value对 写为 key-null ，缓存有效时间可以设置 短点 ，如 30秒（设置太长会导致正常情况下也没法使用）。这样就可以防止攻击用户反复用一个 id 暴力攻击。
- 布隆过滤器。bloomfilter 就类似于一个 hash set ，用户快速判断某个元素是否存在于集合中，其典型的应用场景就是快速判断一个key是否存在于某容器，不存在就直接返回。布隆过滤器的关键就在于 hash算法 和 容器大小。

缓存击穿

问题来源

缓存击穿是指 **缓存中没有但数据库中有的数据（一般是缓存时间到期），这时由于并发用户特别多，同时读缓存没读到数据，又同时去数据库取数据，引起数据库压力瞬间增大，造成压力过大。**

解决方案

- **设置热点数据永不过期。**
- **接口限流与熔断、降级。**重要的接口一定要做好限流策略，防止用户恶意刷接口，同时要降级准备，当接口中的某些服务不可用时，进行熔断，失败快速返回机制。
- **加互斥锁**（redis 分布式锁 或 ReentrantLock），第一个请求的线程可以拿到锁，拿到锁的线程查询到了数据之后设置缓存，其他线程获取锁失败会等待 50ms ，然后重新到缓存拿数据，这样就可以避免大量请求落到数据库。

缓存雪崩

问题来源

缓存雪崩是指 **缓存中大批量数据到过期时间，而查询数据量巨大，引起数据库压力过大甚至down机。**和缓存击穿不同的是，缓存击穿指并发查同一条数据，缓存雪崩是不同数据都过期了，很多数据都查不到从而查数据库。

解决方案

1. 缓存数据的**过期时间设置随机**，防止同一时间大量数据过期现象的发生。
2. 如果缓存数据库是分布式部署，**将热点数据均匀分布在不同的缓存数据库中。**
3. **设置热点数据永不过期。**

缓存污染（或满了）

问题来源

缓存区,存储了太多数据.导致空间不够用.

缓存污染问题说的是缓存中一些只会被访问一次或者几次的数据，被访问完后，再也不会被访问到，但这部分数据依然留存在缓存中，消耗缓存空间。

解决方案

建议把缓存容量设置为总数据量的 15% 到 30% ，兼顾访问性能和内存空间开销。

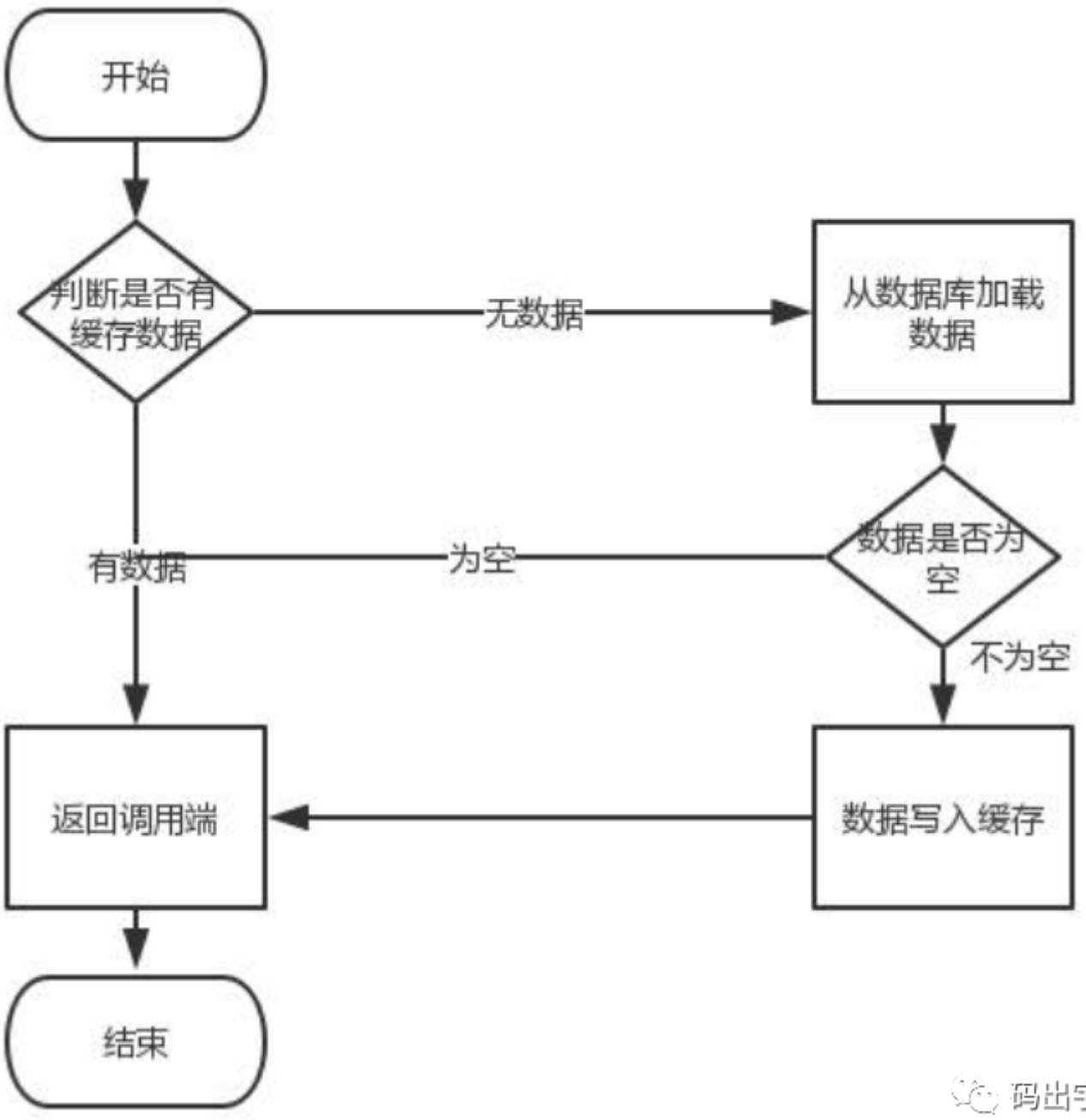
需要缓存淘汰策略,根据淘汰策略去选择要淘汰的数据，然后进行删除操作。

数据库和缓存一致性

问题来源

使用 Redis 做一个缓冲操作，让请求先访问到 Redis ，而不是直接访问 MySQL 数据库：

读取缓存步骤一般没有什么问题，但是一旦涉及到数据更新：**数据库和缓存更新**，就容易出现缓存（Redis）和数据库（MySQL）间的数据一致性问题。



不管是先写 MySQL 数据库，再删除 Redis 缓存；还是先删除缓存，再写库，都有可能出现数据不一致的情况。

案例

1. 如果删除缓存 Redis，但还没来得及 写库 MySQL，另一个线程就来读取，发现缓存为空，则去数据库中读取数据写入缓存，此时缓存中为脏数据。
2. 如果先写了库，在删除缓存前，写库的线程宕机了，没有删除掉缓存，则也会出现数据不一致的情况。

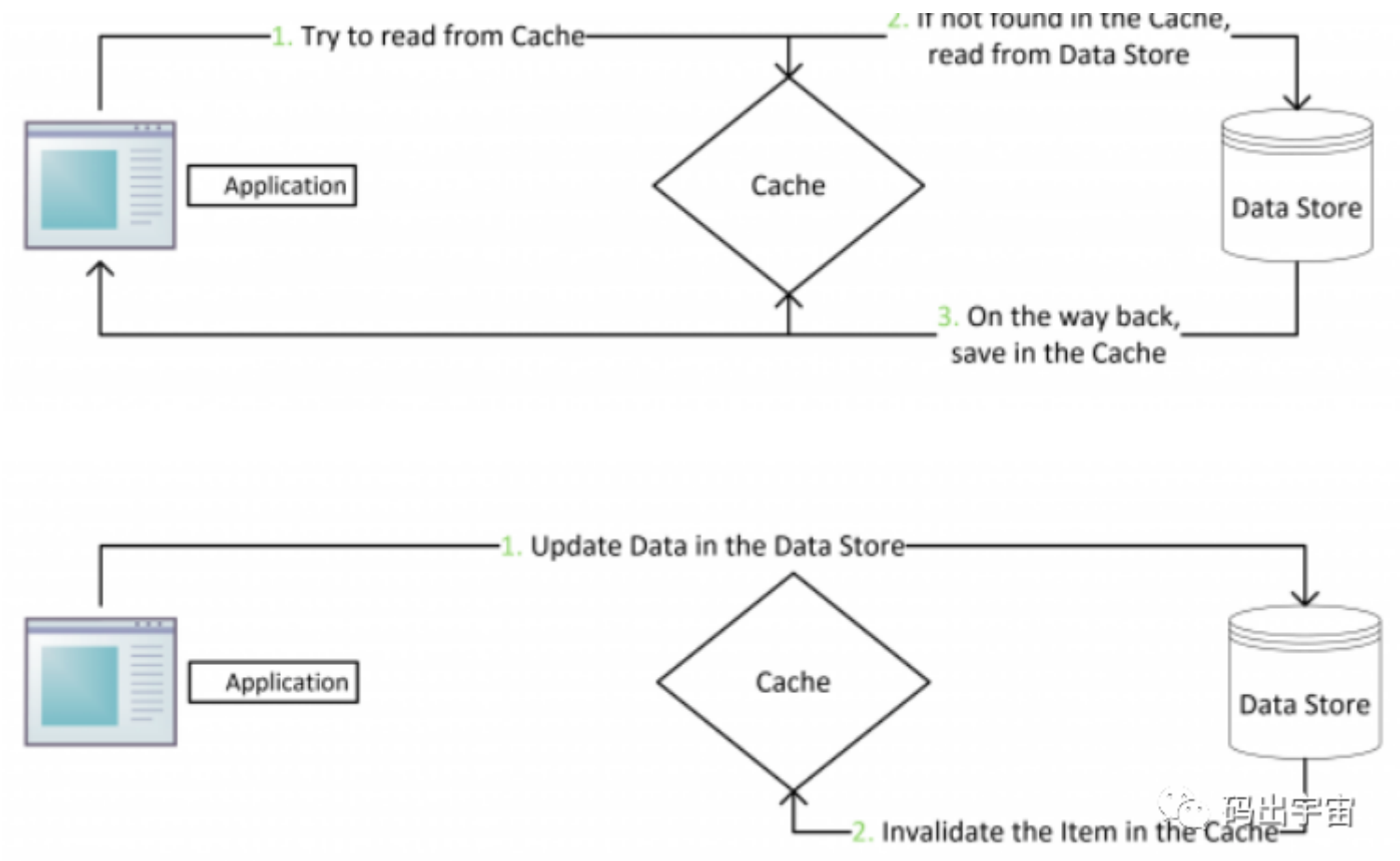
更新缓存的 Design Pattern

Cache Aside Pattern

- **读的时候**：先读缓存，缓存没有的话，就读数据库，然后取出数据后放入缓存，同时返回响应。
- **更新的时候**：先更新数据库，然后再删除缓存。

其具体逻辑如下：

- **失效：**应用程序先从 cache 取数据，没有的到，则从数据库中取数据，成功后，放到缓存中。
- **命中**：应用程序从 cache 中取数据，取到后返回。
- **更新**：先把数据存到数据库中，成功后，再让缓存淘汰。



Redis是什么

数据类型、数据一致、效率、安全

Redis是一个key-value存储系统，它支持存储的value类型相对更多，包括string、list、set、zset和hash。为了保证数据一致，Redis中数据的操作都是原子性的。为了保证效率，数据都是缓存在内存中。为保证安全，Redis会周期性的把更新的数据写入磁盘或者把修改操作写入追加的记录文件，并且在此基础上实现了master-slave（主从）同步。

Redis有哪些功能?

1.基于本机内存的缓存

当调用api访问数据库时，假如此过程需要2秒，如果每次请求都要访问数据库，那将对服务器造成巨大的压力，如果将此sql的查询结果存到Redis中，再次请求时，直接从Redis中取得，而不是访问数据库，效率将得到巨大的提升，Redis可以定时去更新数据（比如1分钟）。

2.持久化的功能

如果电脑重启，写入内存的数据是不是就失效了呢，这时Redis还提供了持久化的功能。

3、哨兵（Sentinel）和复制

Sentinel可以管理多个Redis服务器，它提供了监控、提醒以及自动的故障转移功能；

复制则是让Redis服务器可以配备备份的服务器；

Redis也是通过这两个功能保证Redis的高可用；

4、集群（Cluster）

单台服务器资源总是有上限的，CPU和IO资源可以通过主从复制，进行读写分离，把一部分CPU和IO的压力转移到从服务器上，但是内存资源怎么办，主从模式只是数据的备份，并不能扩充内存；

现在我们可以横向扩展，让每台服务器只负责一部分任务，然后将这些服务器构成一个整体，对外界来说，这一组服务器就像是集群一样。

Redis为什么是单线程的?

1. 代码更清晰，处理逻辑更简单；
2. 不用考虑各种锁的问题，不存在加锁和释放锁的操作，没有因为可能出现死锁而导致的性能问题；
3. 不存在多线程切换而消耗CPU；
4. 无法发挥多核CPU的优势，但可以采用多开几个Redis实例来完善；

Redis真的是单线程的吗?

Redis6.0之前是单线程的，Redis6.0之后开始支持多线程；
redis内部使用了基于epoll的多路服用，也可以多部署几个redis服务器解决单线程的问题；
redis主要的性能瓶颈是内存和网络；
内存好说，加内存条就行了，而网络才是大麻烦，所以redis6内存好说，加内存条就行了；
而**网络**才是大麻烦，所以redis6.0引入了多线程的概念，
redis6.0在网络IO处理方面引入了多线程，如网络数据的读写和协议解析等，需要注意的是，执行命令的核心模块还是单线程的。

Redis持久化有几种方式?

redis提供了两种持久化的方式，分别是RDB（Redis DataBase）和AOF（Append Only File）。

RDB，简而言之，就是在不同的时间点，将redis存储的数据生成快照并存储到磁盘等介质上；

AOF，则是换了一个角度来实现持久化，那就是将redis执行过的所有写指令记录下来，在下次redis重新启动时，只要把这些写指令从前到后再重复执行一遍，就可以实现数据恢复了。

其实RDB和AOF两种方式也可以同时使用，在这种情况下，如果redis重启的话，则会优先采用AOF方式来进行数据恢复，这是因为AOF方式的数据恢复完整度更高。

Redis支持的 java 客户端都有哪些?

Redisson、Jedis、lettuce 等等，官方推荐使用 Redisson。

Redis怎么实现分布式锁?

使用Redis实现分布式锁

redis命令：set users 10 nx ex 12 原子性命令

```
//使用uuid，解决锁释放的问题
@GetMapping
public void testLock() throws InterruptedException {
    String uuid = UUID.randomUUID().toString();
    Boolean b_lock = redisTemplate.opsForValue().setIfAbsent("lock", uuid, 10,
TimeUnit.SECONDS);
    if(b_lock){
        Object value = redisTemplate.opsForValue().get("num");
        if(StringUtils.isEmpty(value)){
            return;
        }
        int num = Integer.parseInt(value + "");
        redisTemplate.opsForValue().set("num",++num);
        Object lockUUID = redisTemplate.opsForValue().get("lock");
        if(uuid.equals(lockUUID.toString())){
            redisTemplate.delete("lock");
        }
    }else{
        Thread.sleep(100);
        testLock();
    }
}
```

备注：可以通过lua脚本，保证分布式锁的原子性。

Redis分布式锁有什么缺陷?

Redis 分布式锁不能解决超时的问题，分布式锁有一个超时时间，程序的执行如果超出了锁的超时时间就会出现问题。

Redis容易产生的几个问题：

1. 锁未被释放
2. B锁被A锁释放了
3. 数据库事务超时
4. 锁过期了，业务还没执行完
5. Redis主从复制的问题

Redis如何做内存优化?

1、缩短键值的长度

1. 缩短值的长度才是关键，如果值是一个大的业务对象，可以将对象序列化成二进制数组；
2. 首先应该在业务上进行精简，去掉不必要的属性，避免存储一些没用的数据；
3. 其次是序列化的工具选择上，应该选择更高效的序列化工具来降低字节数组大小；
4. 以JAVA为例，内置的序列化方式无论从速度还是压缩比都不尽如人意，这时可以选择更高效的序列化工具，如: protostuff，kryo等

2、共享对象池

对象共享池指Redis内部维护[0-9999]的整数对象池。创建大量的整数类型redisObject存在内存开销，每个redisObject内部结构至少占16字节，甚至超过了整数自身空间消耗。所以Redis内存维护一个[0-9999]的整数对象池，用于节约内存。 除了整数值对象，其他类型如list,hash,set,zset内部元素也可以使用整数对象池。因此开发中在满足需求的前提下，尽量使用整数对象以节省内存。

3、字符串优化

4、编码优化

5、控制key的数量