Redis4-雪崩穿透+解决方案

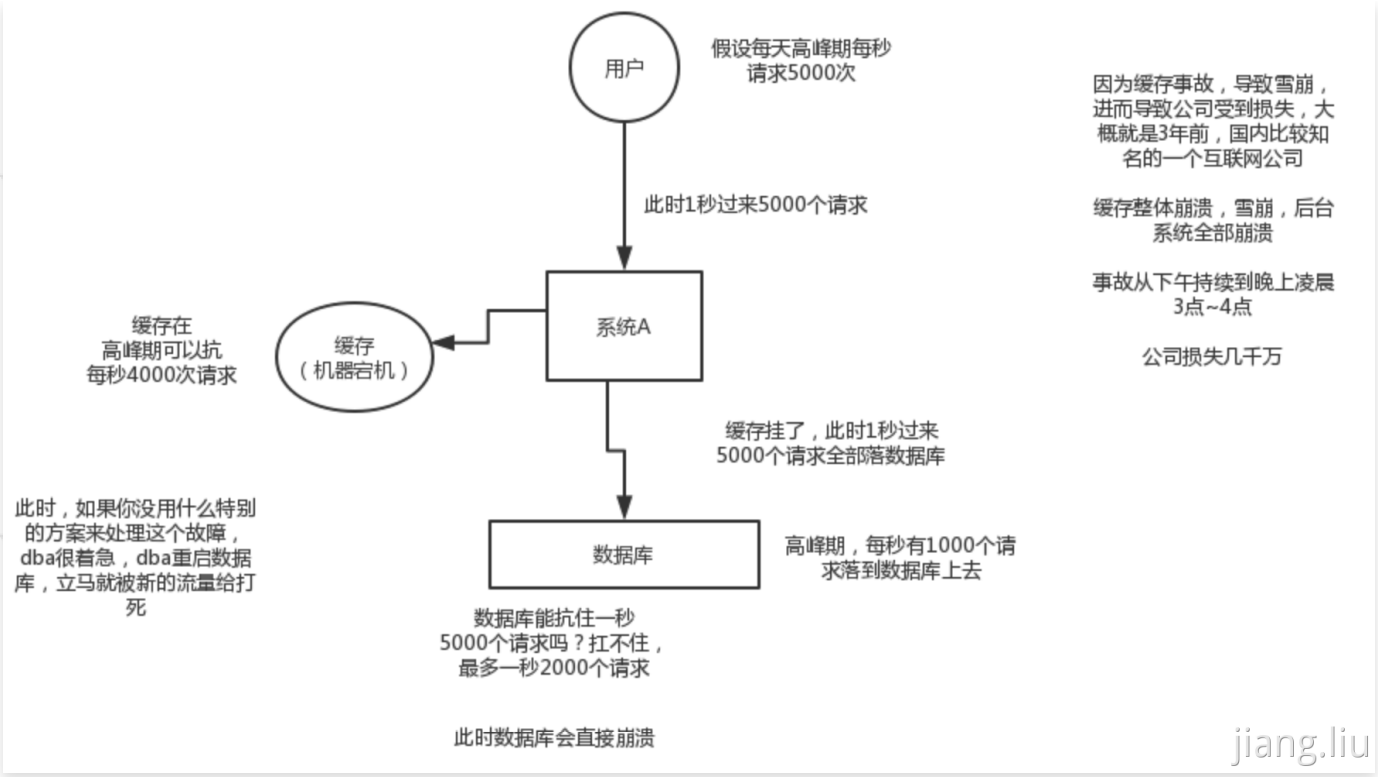
1.什么是redis的雪崩和穿透?

缓存雪崩:

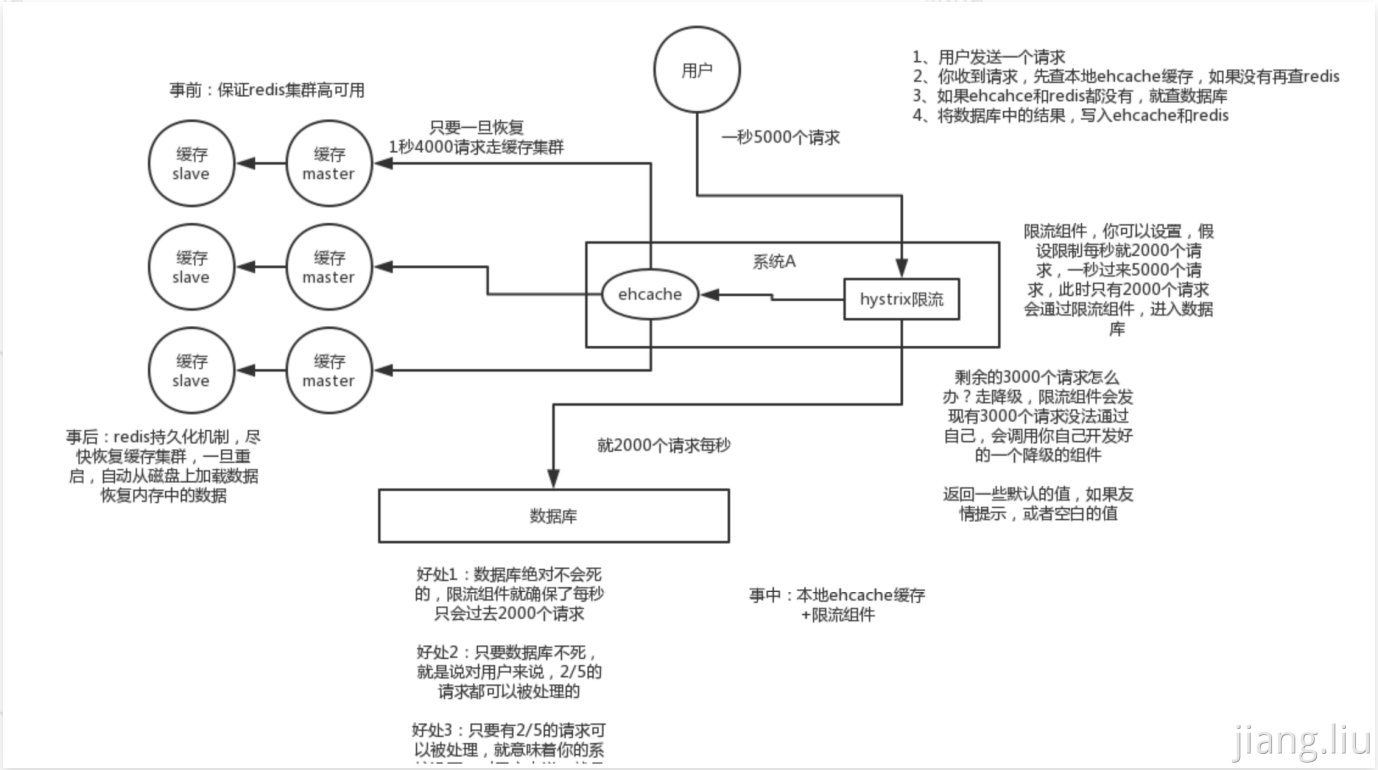
所谓缓存雪崩就是在某一个时刻，缓存集大量失效（比如集体过期）。所有流量直接打到数据库上，对数据库造成巨大压力；甚至宕机。

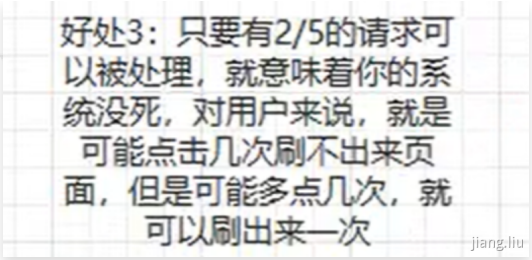
解决方案:

- 增加本地encache缓存+hystrix限流组件
- 避免缓存设置相近的有效期，为有效期增加随机值；防止同一时间大量数据过期现象发生


用户
假设每天高峰期每秒
请求5000次
此时1秒过来5000个请求
系统A
缓存
（机器宕机）
缓存在
高峰期可以抗
每秒4000次请求
缓存挂了，此时1秒过来
5000个请求全部落数据库
数据库
高峰期，每秒有1000个请
求落到数据库上去
此时，如果你没用什么特别
的方案来处理这个故障，
dba很着急，dba重启数据
库，立马就被新的流量给打
死
数据库能抗住一秒
5000个请求吗？扛不住，
最多一秒2000个请求
此时数据库会直接崩溃
因为缓存事故，导致雪崩，
进而导致公司受到损失，大
概就是3年前，国内比较知
名的一个互联网公司
缓存整体崩溃，雪崩，后台
系统全部崩溃
事故从下午持续到晚上凌晨
3点~4点
公司损失几千万
jiang.liu

如何解决缓存雪崩?


用户
1、用户发送一个请求
2、你收到请求，先查本地ehcache缓存，如果没有再查redis
3、如果ehcahce和redis都没有，就查数据库
4、将数据库中的结果，写入ehcache和redis
事前：保证redis集群高可用
缓存
slave
缓存
master
只要一旦恢复
1秒4000请求走缓存集群
一秒5000个请求
系统A
ehcache
hystrix限流
限流组件，你可以设置，假
设限制每秒就2000个请
求，一秒过来5000个请
求，此时只有2000个请求
会通过限流组件，进入数据
库
剩余的3000个请求怎么
办？走降级，限流组件会发
现有3000个请求没法通过
自己，会调用你自己开发好
的一个降级的组件
返回一些默认的值，如果友
情提示，或者空白的值
事后：redis持久化机制，尽
快恢复缓存集群，一旦重
启，自动从磁盘上加载数据
恢复内存中的数据
就2000个请求每秒
数据库
好处1：数据库绝对不会死
的，限流组件就确保了每秒
只会过去2000个请求
好处2：只要数据库不死，
就是说对用户来说，2/5的
请求都可以被处理的
好处3：只要有2/5的请求可
以被处理，就意味着你的系
事中：本地ehcache缓存
+限流组件
jiang.liu



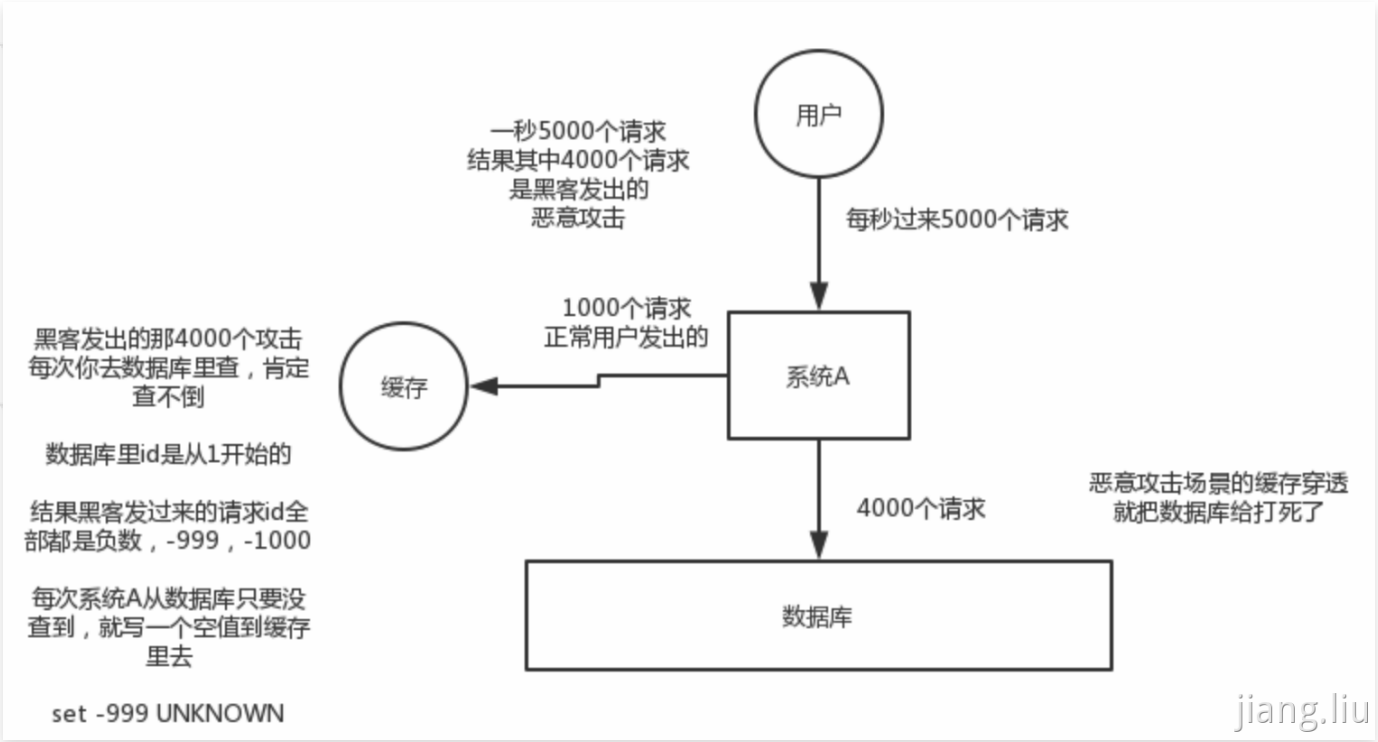
缓存穿透:

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。每次请求都去查询数据库，导致缓存形同虚设。

一般来说，缓存穿透的场景发生在故意攻击的场景下；比如说，本来查询一件商品的序号是正数，但是请求方总是请求大量的负数过来，导致缓存无效，全部请求都打在了数据库中，如果某一时刻流量过大，可能会导致数据库崩溃。

解决方案：

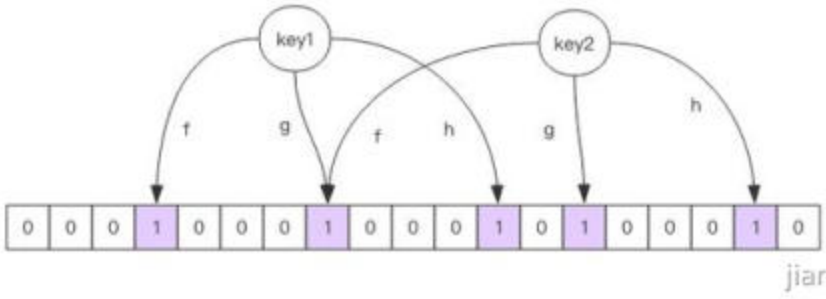
- 接口层增加校验，如用户鉴权校验，id做基础校验，id<=0的直接拦截；
- 从缓存取不到的数据，在数据库中也没有取到，这时也可以将key-value对写为key-null，缓存有效时间可以设置短点，如30秒（设置太长会导致正常情况也没法使用）。这样可以防止攻击用户反复用同一个id暴力攻击.



还有第三种解决方案：**布隆过滤器**

将数据库中所有的查询条件，放入布隆过滤器中，当一个查询请求过来时，先经过布隆过滤器进行查，如果判断请求查询值存在，则继续查；如果判断请求查询不存在，直接丢弃。

> ps：布隆过滤器原理
>
> 布隆过滤器是由一个长度为m比特的位数组（bit array）与k个哈希函数（hash function）组成的数据结构。原理就是对一个key进行k个hash算法获取k个值，在比特数组中将这k个值散列后设定为1，然后查的时候如果特定的这几个位置都为1，那么布隆过滤器判断该key存在。
>
> 布隆过滤器可能会误判，如果它说不存在那肯定不存在，如果它说存在，那数据有可能实际不存在；
>
> Redis的bitmap只支持2^32大小，对应到内存也就是512MB，误判率万分之一，可以放下2亿左右的数据，性能高，空间占用率及小，省去了大量无效的数据库连接。
>
> 因此我们可以通过布隆过滤器，将Redis缓存穿透控制在一个可容范围内

> Redis 官方提供的布隆过滤器到了 Redis 4.0 提供了插件功能之后才正式登场。布隆过滤器作为一个插件加载到 Redis Server 中，给 Redis 提供了强大的布隆去重功能。
>
> 
>
>
> 每个布隆过滤器对应到 Redis 的数据结构里面就是一个大型的位数组和几个不一样的无偏 hash 函数。所谓无偏就是能够把元素的 hash 值算得比较均匀。
> 向布隆过滤器中添加 key 时，会使用多个 hash 函数对 key 进行 hash 算得一个整数索引值然后对位数组长度进行取模运算得到一个位置，每个 hash 函数都会算得一个不同的位置。再把位数组的这几个位置都置为 1 就完成了 add 操作。
> 向布隆过滤器询问 key 是否存在时，跟 add 一样，也会把 hash 的几个位置都算出来，看看位数组中这几个位置是否都为 1，只要有一个位为 0，那么说明布隆过滤器中这个 key 不存在。如果都是 1，这并不能说明这个 key 就一定存在，只是极有可能存在，因为这些位被置为 1 可能是因为其它的 key

> 存在所致。如果这个位数组比较稀疏，判断正确的概率就会很大，如果这个位数组比较拥挤，判断正确的概率就会降低。

布隆过滤器的其他应用场景：

> 1. 爬虫爬取到的海量url去重。
> 2. Hbase中也有布隆过滤器结构。
> 3. 邮箱系统的垃圾邮件过滤功能也普遍用到了布隆过滤器。

缓存击穿

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，又同时去数据库去取数据，引起数据库压力瞬间增大，造成过大压力。

解决方案：

a. 设置热点数据永远不过期。

b. 加互斥锁

> 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。

2. 如何保证缓存和数据库双写一致性?

> 最经典的缓存+数据库读写的模式，cache aside pattern

Cache Aside Pattern

1）读的时候，先读缓存，缓存没有的话，那么就读数据库，然后取出数据后放入缓存，同时返回响应。

2）更新的时候，先删除缓存，然后再更新数据库。

为什么是删除缓存，而不是更新缓存呢?

原因很简单，很多时候，复杂点的缓存的场景，因为缓存有的时候，不简单是数据库中直接取出来的值。

例如有一个商品详情页的系统，修改库存，只是修改了某个表的某些字段，但是要真正把这个影响的最终的库存计算出来，可能还需要从其他表查询一些数据，然后进行一些复杂的运算，才能最终计算出现在最新的库存是多少，然后才能将库存更新到缓存中去。比如可能更新了某个表的一个字段，然后其对应的缓存，是需要查询另外两个表的数据，并进行运算，才能计算出缓存最新的值的。

是不是说，每次修改数据库的时候，都一定要将其对应的缓存去更新一份？也许有的场景是这样的，但是对于比较复杂的缓存数据计算的场景，就不是这样了。举个例子，一个缓存涉及的表的字段，在1分钟内就修改了20次，或者是100次，那么缓存跟新20次，100次; 但是这个缓存在1分钟内就被读取了1次，有大量的冷数据。实际上，如果你只是删除缓存的话，那么1分钟内，这个缓存不过就重新计算一次而已，开销大幅度降低。

> 28法则，黄金法则，20%的数据，占用了80%的访问量

其实删除缓存，而不是更新缓存，就是一个lazy计算的思想，不要每次都重新做复杂的计算，不管它会不会用到，而是让它到需要被使用的时候再重新计算。

> mybatis，hibernate，懒加载，思想。

查询一个部门，部门带了一个员工的list，没有必要说每次查询部门，都里面的1000个员工的数据也同时查出来。80%的情况，查这个部门，就只是要访问这个部门的信息就可以了，先查部门，同时要访问里面的员工，那么这个时候只有在你要访问里面的员工的时候，才会去数据库里面查询1000个员工。

缓存不一致问题以及解决方案

问题1：先修改数据库，再删除缓存，如果删除缓存失败了，那么会导致数据库中是新数据，缓存中是旧数据，数据出现不一致。

解决思路：先删除缓存，再修改数据库，如果删除缓存成功了，如果修改数据库失败了，那么数据库中是旧数据，缓存中是空的，那么数据不会不一致，因为读的时候缓存没有，则读数据库中旧数据，然后更新到缓存中。

还有另一种思路，先修改数据库，再删除缓存，如果删除缓存失败，将删除的key写入mq中，自己再消费，然后重复删，知道删除成功。

问题2：数据发生变更，先删除缓存然后准备修改数据库，此时还没修改，一个请求过来，去读缓存，发现缓存是空的，然后去数据库查到了修改前的旧数据，又放到了缓存中，这时候更新程序才完成了数据库的修改，完了，数据库和缓存中的数据不一样了！

解决思路：只有在对一个数据在并发的进行读写的时候，才可能会出现这种问题，可以采用**采用延时双删策略** 伪代码如下：

Java 复制代码

```java
public void write(String key,Object data){
        redis.delKey(key);
        db.updateData(data);
        Thread.sleep(1000);
        redis.delKey(key);
    }
```

转化为中文描述就是：

（1）先删除缓存

（2）再更新数据库（这两步和原来一样）

（3）休眠1秒，再删除缓存，这样可以将1秒内所造成的缓存脏数据再次删除。

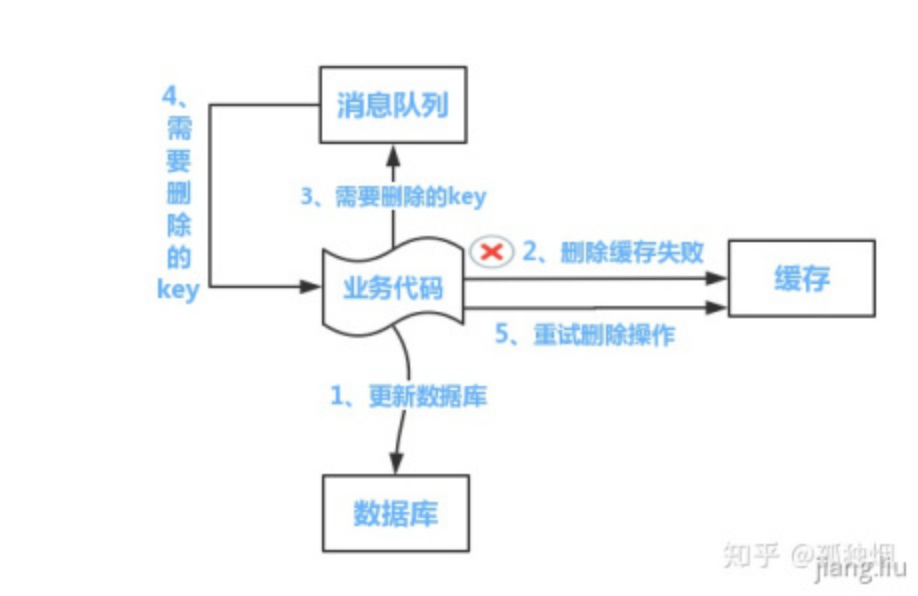
那么，这个1秒怎么确定的，具体该休眠多久呢？应该自行评估自己的项目的读数据业务逻辑的耗时。然后写数据的休眠时间则在读数据业务逻辑的耗时基础上，加几百ms即可。

有人会问，采用这种同步淘汰策略，吞吐量降低怎么办？

OK，那就将第二次删除异步执行。新起一个线程异步删除。这样写的请求就不用沉睡一段时间后再返回了。加大吞吐量。

还有人会问，第二次删除,如果删除失败怎么办？

重试机制，如下图：



（1）更新数据库数据；

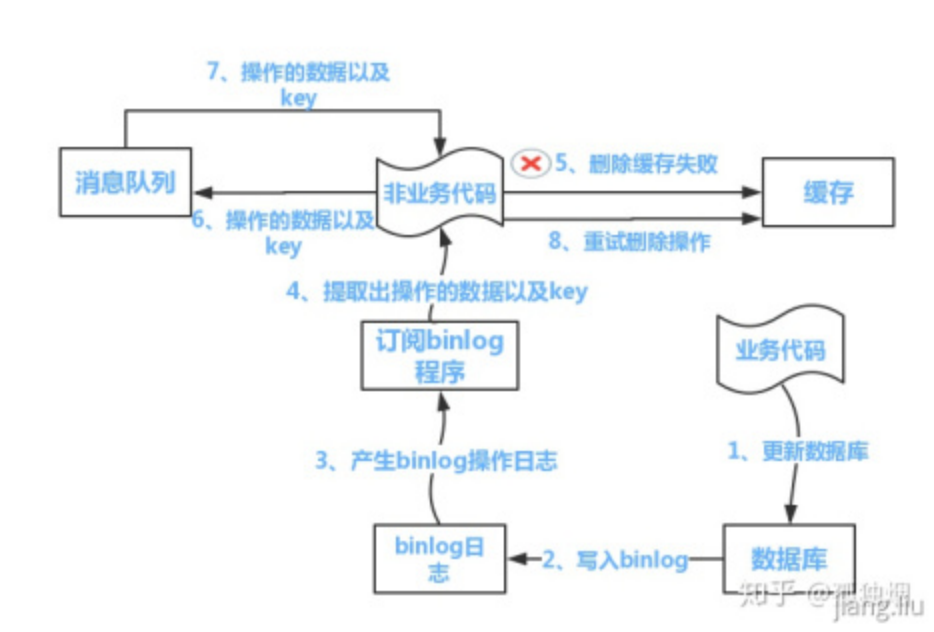
（2）缓存因为种种问题删除失败

（3）将需要删除的key发送至消息队列

（4）自己消费消息，获得需要删除的key

（5）继续重试删除操作，直到成功 然而，该方案有一个缺点，对业务线代码造成大量的侵入。

于是有了方案二：



（1）更新数据库数据

（2）数据库会将操作信息写入binlog日志当中

（3）订阅程序提取出所需要的数据以及key

（4）另起一段非业务代码，获得该信息

（5）尝试删除缓存操作，发现删除失败

（6）将这些信息发送至消息队列

（7）重新从消息队列中获得该数据，重试操作。

> 备注说明：上述的订阅binlog程序在mysql中有现成的中间件叫canal，可以完成订阅binlog日志的功能。另外，重试机制，我采用的消息队列的方式。如果对一致性要求不是很高，直接在程序中另起一个线程，每隔一段时间去重试即可，这些大家可以灵活自由发挥，只是提供一个思路。

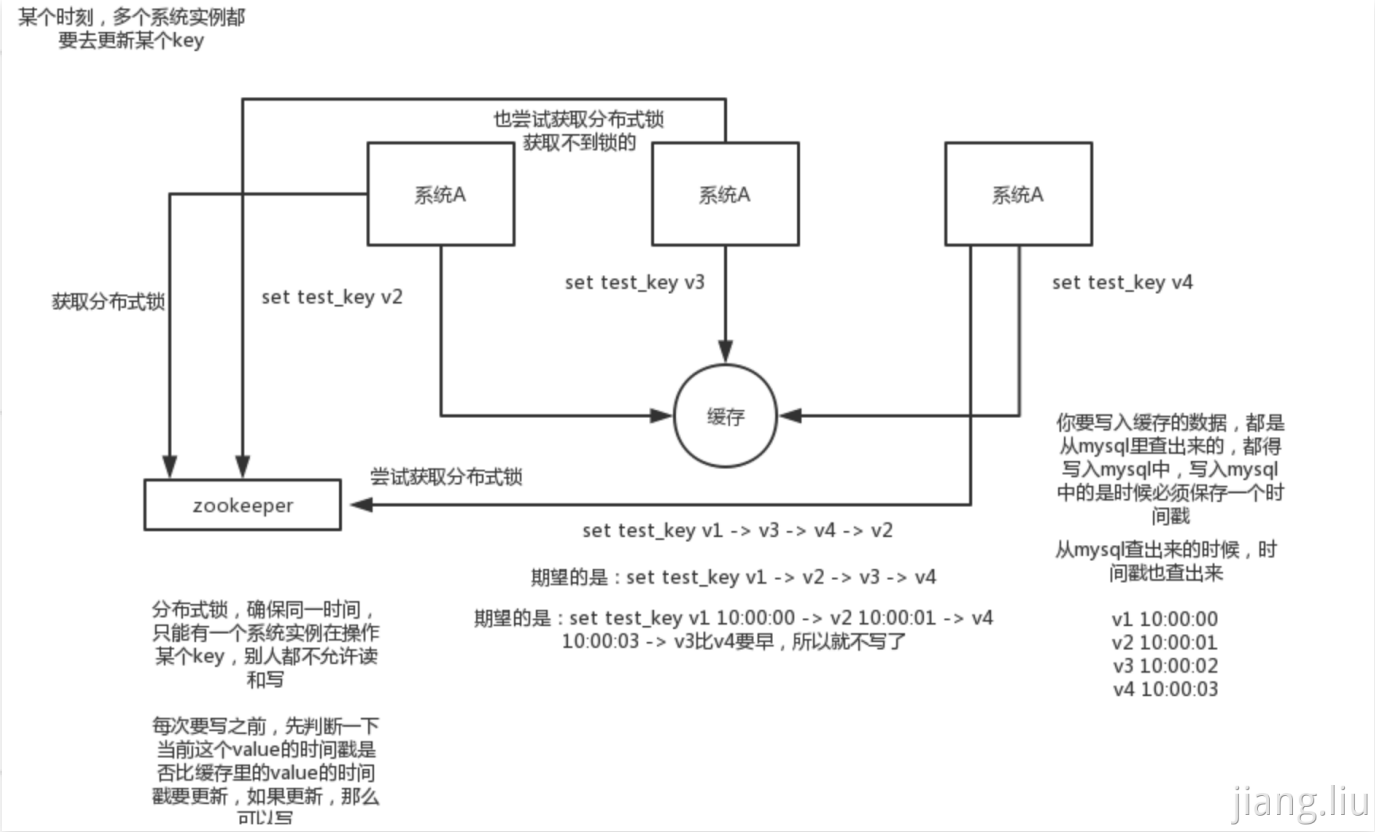
问题3：使用Mysql主从架构读写分离，主库和从库造成数据不一致。

解决思路：延时双删+重试。

3. Redis的并发竞争问题？如何解决？

这个也是非常常见的一个问题，就是多客户端同时并发写一个key，可能本来应该先到的数据后到了，导致数据版本错了。或者是多客户端同时获取一个key，修改值之后再写回去，只要顺序错了，数据就错了。

> redis自己有天然解决这个问题的CAS类的乐观锁方案



使用分布式锁，确保同一时间，只能有一个线程在操作某个key，别人都不允许读和写，每次要写之前，先判断一下当前这个value的时间戳是否比缓存里的value的时间戳要更新一些，如果更新，那么可以写，如果更旧，就不能用旧的数据覆盖新的数据。

4. 生产环境中的redis是怎么部署的?

redis cluster，10台机器，5台机器部署了redis主实例，另外5台机器部署了redis的从实例，每个主实例挂了一个从实例，5个节点对外提供读写服务，每个节点的读写高峰qps可能可以达到每秒5万，5台机器最多是25万读写请求/s。

机器是什么配置？32G内存+8核CPU+1T磁盘，但是分配给redis进程的是10g内存，一般线上生产环境，redis的内存尽量不要超过10g，超过10g可能会有问题。

5台机器对外提供读写，一共有50g内存。

因为每个主实例都挂了一个从实例，所以是高可用的，任何一个主实例宕机，都会自动故障迁移，redis从实例会自动变成主实例继续提供读写服务。

你往内存里写的是什么数据？每条数据的大小是多少？商品数据，每条数据是10kb。100条数据是1mb，10万条数据是1g。常驻内存的是200万条商品数据，占用内存是20g，仅仅不到总内存的50%。

目前高峰期每秒就是3500左右的请求量。

比如我们吧，大型的公司，其实基础架构的team，会负责缓存集群的运维。