## 一、Redis 分布式锁

应用系统在业务量增长到一定时候肯定会遇到并发问题，为了规避并发安全带来问题，这个时候必须使用到分布式锁来限制程序的并发执行，确保某一个时刻只处理一个请求，Redis提供了分布式锁命令支持。

1. 命令

set key value [ex seconds] [px milliseconds] [nx]

1. java 客户端支持
2. jedis原生支持

jedis.set(key, value,"nx", "ex", time);

1. SpringDataRedis 支持

redisTemplate.opsForValue().setIfAbsent("key", "value ", time, unit);

## 二、数据预热

当应用上平台业务越来越大的时候，系统并发越来越高时候，平台上一个新功能或搞个活动，在可预见的情况下，并发量高的时候，有些基础数据（如：配置信息，商品信息等）访问很频繁。

这时数据库压力比较大，可以通过数据预热方式，后台定时任务批量（Redis 提供管道批处理）查数据库数据写入缓存。基础数据提前写入缓存，带来应用更快处理速度和更大吞吐量。

## 三、缓存雪崩

高并发场景下，说商品类的缓存数据过期时间设置过于集中，大量缓存集中失效，这样场景称为缓存雪崩。在缓存大量失效且高并发场景下，大量请求击穿缓存，直击DB。

解决方案：将过期时间散列表开。设置一个过期时间基数+rand.nextInt(n) .这样就避免缓存过期时间集中大量的失效。避免出现缓存雪崩问题。

## 四、缓存穿透

某个key缓存失效，在高并发场景下，大量请求还是直击DB。

解决方案：分布式锁。此方案只允许在某个时间只有一个线程重建缓存，其他线程等待重建缓存的线程执行完，重新从缓存获取数据即可。

## 五、数据过热

如果某个key数据实在太热，查询请求海量，就比如微博热搜。单个redis节点都可能撑不住。

解决方案：数据散列开。将一份热点数据的key,做成N份，就弄成3份，key1,key2,key3,分别写入Redis-1,Redis-2,Redis-3，这样数据均匀打散到这三个redis节点上，分担了读压力，缓存就不至于压力太大而导致宕机。

## 六、缓存穿透

业务应用系统受到黑客大量攻击，查询那些既不在缓存也不在数据库里数据，缓存失去保护DB作用，这种场景业界称之为缓存穿透。发生缓存穿透，高并发场景下，如果是黑客恶意攻击也是致命的。那么在缓存这层怎么作保护方案？

1. **缓存空数据**

将数据库查询结果为空的key也存储在缓存中。当后续又出现该key的查询请求时，缓存直接返回null，而无需查询数据库。

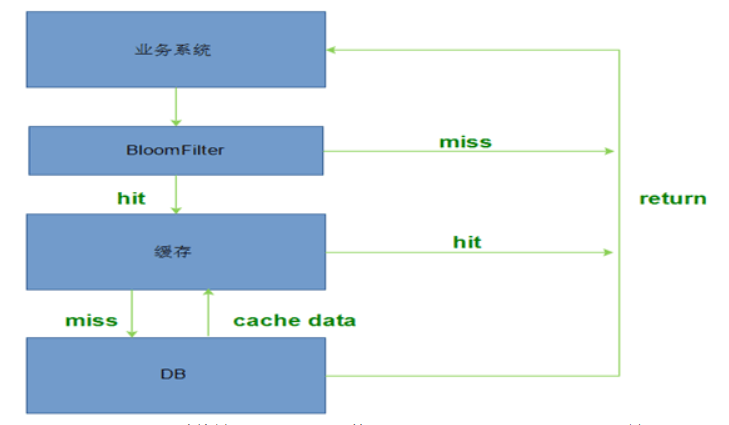
缓存空对象会有两个问题：

第一，空值做了缓存，意味着缓存层中存了更多的键，需要更多的内存空间 ( 如果是攻击，问题更严重 )，比较有效的方法是针对这类数据设置一个较短的过期时间，让其自动剔除。

第二，缓存层和存储层的数据会有一段时间窗口的不一致，可能会对业务有一定影响。例如过期时间设置为 5 分钟，如果此时存储层添加了这个数据，那此段时间就会出现缓存层和存储层数据的不一致，此时可以利用消息系统或者其他方式清除掉缓存层中的空对象。

1. **BloomFilter**

BloomFilter推荐使用Redis 中的布隆过滤器，不过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。



查询请求进入应用系统，首先去BloomFilter中查询该key是否存在。若不存在，则说明数据库中也不存在该数据，因此缓存都不要查了，直接返回null。若存在，则继续执行后续的流程，先前往缓存中查询，缓存中没有的话再前往数据库中的查询。

**缓存空数据 VS BloomFilter**

对于一些恶意攻击，查询的key往往各不相同，而且数据贼多。此时，第一种方案就显得提襟见肘了。因为它需要存储所有空数据的key，而这些恶意攻击的key往往各不相同，而且同一个key往往只请求一次。因此即使缓存了这些空数据的key，由于不再使用第二次，因此也起不了保护数据库的作用。

因此，对于空数据的key各不相同、key重复请求概率低的场景而言，应该选择第二种方案。而对于空数据的key数量有限、key重复请求概率较高的场景而言，应该选择第一种方案。