缓存雪崩、缓存击穿、缓存穿透的区别和解决方案

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，如发起为id为“-1”的数据或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，也可以使用会话重放，防止不断的攻击。

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：

设置热点数据永远不过期。 加互斥锁

```
public static String getData(String key) throws InterruptedException
{
    //从缓存读取数据
    String result = getDataFromRedis(key);
    //缓存中不存在数据
    if (result == null)
    {
        //去获取锁，获取成功，去数据库取数据
        if (reenLock.tryLock())
        {
            //从数据获取数据
            result = getDataFromMysql(key);
            //更新缓存数据
            if (result != null)
            {
                setDataToCache(key,result);
            }
            //释放锁
            reenLock.unlock();
        }
        //获取锁失败
        else
        {
            //暂停100ms再重新去获取数据
            Thread.sleep(100);
            result =  getData(key);
        }
    }
    return result;
}
```


缓存雪崩

缓存雪崩是指缓存中数据大批量到过期时间，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

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发生。 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。也可以是二级缓存，本地和redis共同使用。 设置热点数据永远不过期。

redis的常用数据类型

string类型，key-value

hash类型，键值对组合

list类型，简单的字符串列表。按照插入顺序排序。lpush，lpop

set类型，string的无序不重复集合。比如在微博应用中，可以将一个用户所有的关注人存在一个集合中，将其所有粉丝存在一个集合。Redis还为集合提供了求交集、并集、差集等操作，可以非常方便的实现如共同关注、共同喜好、二度好友等功能

zset类型，Redis sorted set的内部使用HashMap和跳跃表(SkipList)来保证数据的存储和有序，HashMap里放的是成员到score的映射，而跳跃表里存放的是所有的成员，排序依据是HashMap里存的score,使用跳跃表的结构可以获得比较高的查找效率，并且在实现上比较简单。

redis的高级数据类型

bitmap

【存储需求】

党员	党员	党员
是	Y	1

- **获取指定key对应偏移量上的bit值**

```
getbit key offset
```

- **设置指定key对应偏移量上的bit值，value只能是1或0**

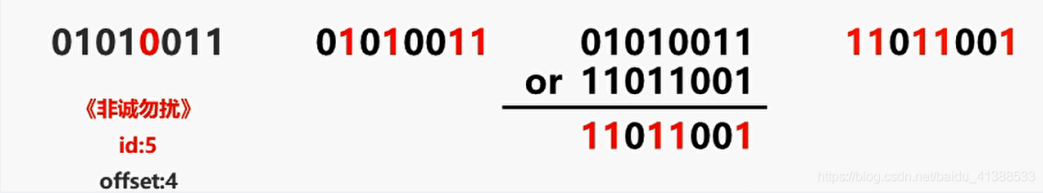
```
setbit key offset value
```

【示例】

```
127.0.0.1:6379> setbit bits 0 1
(integer) 0
127.0.0.1:6379> getbit bits 0
(integer) 1
127.0.0.1:6379> getbit bits 10
(integer) 0
127.0.0.1:6379> setbit bits 100000000 1
(integer) 0
(1.21s)
127.0.0.1:6379>
```

业务场景【redis 应用于信息状态统计】

- 电影网站
  - 统计每天某一部电影是否被点播
  - 统计每天有多少部电影被点播
  - 统计每周/月/年有多少部电影被点播
  - 统计年度哪部电影没有被点播
- 分析：



hyperloglog

- **基数是数据集去重后元素个数**
- HyperLogLog 是**用来做基数统计**的，运用了LogLog的算法

{1, 3, 5, 7, 5, 7, 8}	基数集：	{1, 3, 5 ,7, 8}	基数：5
{1, 1, 1, 1, 1, 7, 1}	基数集：	{1,7}	基数：2

- 用于进行基数统计，不是集合，不保存数据，只记录数量而不是具体数据
- 核心是基数估算算法，最终数值存在一定误差
- 误差范围：基数估计的结果是一个带有 0.81% 标准错误的近似值
- 耗空间极小，每个hyperloglog key占用了12K的内存用于标记基数
- pfadd命令不是一次性分配12K内存使用，会随着基数的增加内存逐渐增大
- Pfmerge命令合并后占用的存储空间为12K，无论合并之前数据量多少

geo：应用于地理位置计算

- **添加坐标点**

```
geoadd key longitude latitude member [longitude latitude member ...]
```

- **获取坐标点**

```
geopos key member [member ...]
```

- **计算坐标点距离**

```
geodist key member1 member2 [unit]
```

redis存储bean如何优化

在Memcached中，我们经常将一些结构化的信息打包成HashMap，在客户端序列化后存储为一个字符串的值，比如用户的昵称、年龄、性别、积分等，这时候在需要修改其中某一项时，通常需要将所有值取出反序列化后，修改某一项的值，再序列化存储回去。这样不仅增大了开销，也不适用于一些可能并发操作的场合（比如两个并发的操作都需要修改积分）。而Redis的Hash结构可以使你像在数据库中Update一个属性一样只修改某一项属性值。

redisTemplate.opsForHash().put(redisKey, path, String.valueOf(num))

Map<Object,Object> objectMap = redisTemplate.opsForHash().entries(redisKey);

redis的setnx和setex区别

setex： setex key seconds value：将key值设置为value，并将设置key的生存周期 1，属于原子操作，作用和set key value、expire key seconds作用一致。 2，如果key值存在，使用setex将覆盖原有值 setnx: setnx key value:只有当key不存在的情况下，将key设置为value；若key存在，不做任何操作，结果成功返回1，失败返回0