## 千万数据的快速定位

1. 海量的数据应该如何查询?
   * 1. 使用应用缓存组件redis

## 缓存穿透

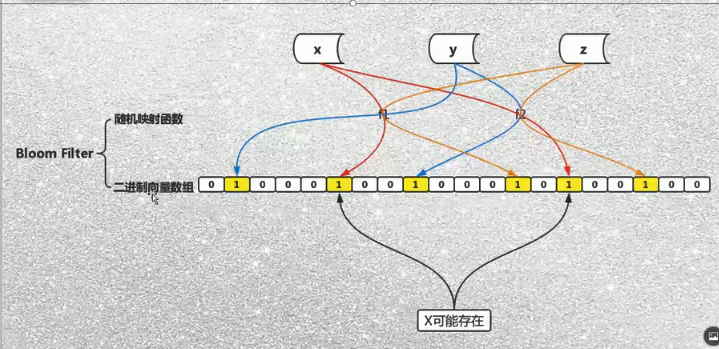
1. **redis的应用**

在查询的操作中,redis中主要存放热点的数据,不适合全部存放数据库中的数据,因为内存太昂贵了,没有必要.

1. **什么叫做缓存穿透?**

缓存穿透是查询的数据在redis中是不存在的,请求直接访问到了数据库,如果数据库中存在大量的数据的情况下,进行一个查询操作,查询耗时是比较长的,非常的影响用户体验.

1. **解决方案**
   1. 使用布隆过滤器
      1. 什么是布隆过滤器

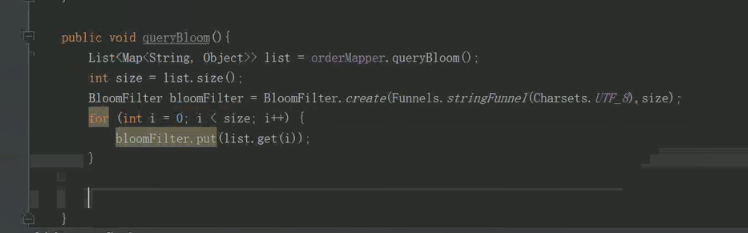


1. 刚开始对数据初始化之前,二进制数组中放的都是0
2. 初始化的时候,比如初始化key为x,y,z的数据,会分别根据hash函数进行运算二进制向量数组的位置(一个key会有多个位置),并把0改为1
3. 查询的时候,比如查询key为q的值,如果多个hash运算的结果位置都为1,说明可能存在,如果有一个为0,说明一定不存在这个key.由于hash算法的hash碰撞问题会导致有一定不存在的值,会出现运算结果都是1的情况.
4. 如何减少布隆过滤器的误判率,可以添加二进制向量数组的长度和hash函数的数量.那么问题来了,如何合理的设置,多了浪费内存,少了,布隆过滤器的应用没有效果,还是会出现穿透的问题.

布隆过滤器可以用于检索一个元素是否存在一个集合中.

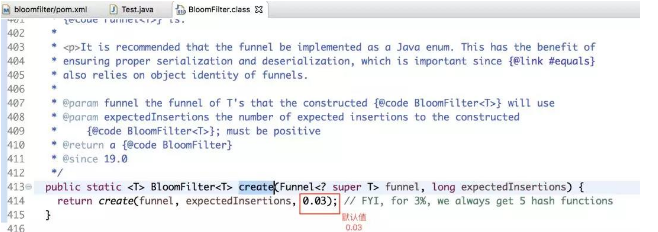
布隆过滤的使用

1. 首先布隆过滤器存在于google发行的guava的包里面
2. 布隆过滤器的初始化代码如下所示:



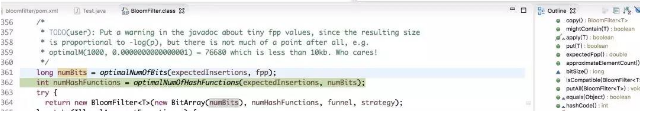
1. 布隆过滤器误判率处理

如果上述代码所示，我们故意取10000个不在过滤器里的值，却还有330个被认为在过滤器里，这说明了误判率为0.03.即，在不做任何设置的情况下，默认的误判率为0.03。  
下面上源码来证明：



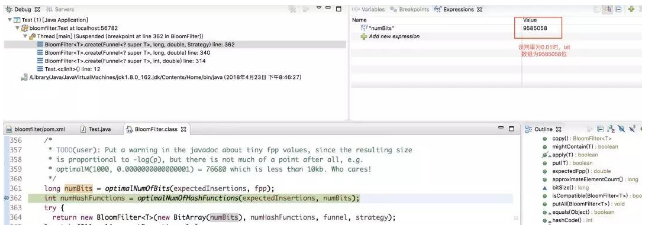
接下来我们来看一下，误判率为0.03时，底层维护的bit数组的长度如下图所示





将bloomfilter的构造方法改为  
private static BloomFilter<Integer> bloomFilter = BloomFilter.create(Funnels.integerFunnel(), size,0.01);

即，此时误判率为0.01。在这种情况下，底层维护的bit数组的长度如下图所示



由此可见，误判率越低，则底层维护的数组越长，占用空间越大。因此，误判率实际取值，根据服务器所能够承受的负载来决定，不是拍脑袋瞎想的。

## 处理redis缓存穿透的方案总结:

