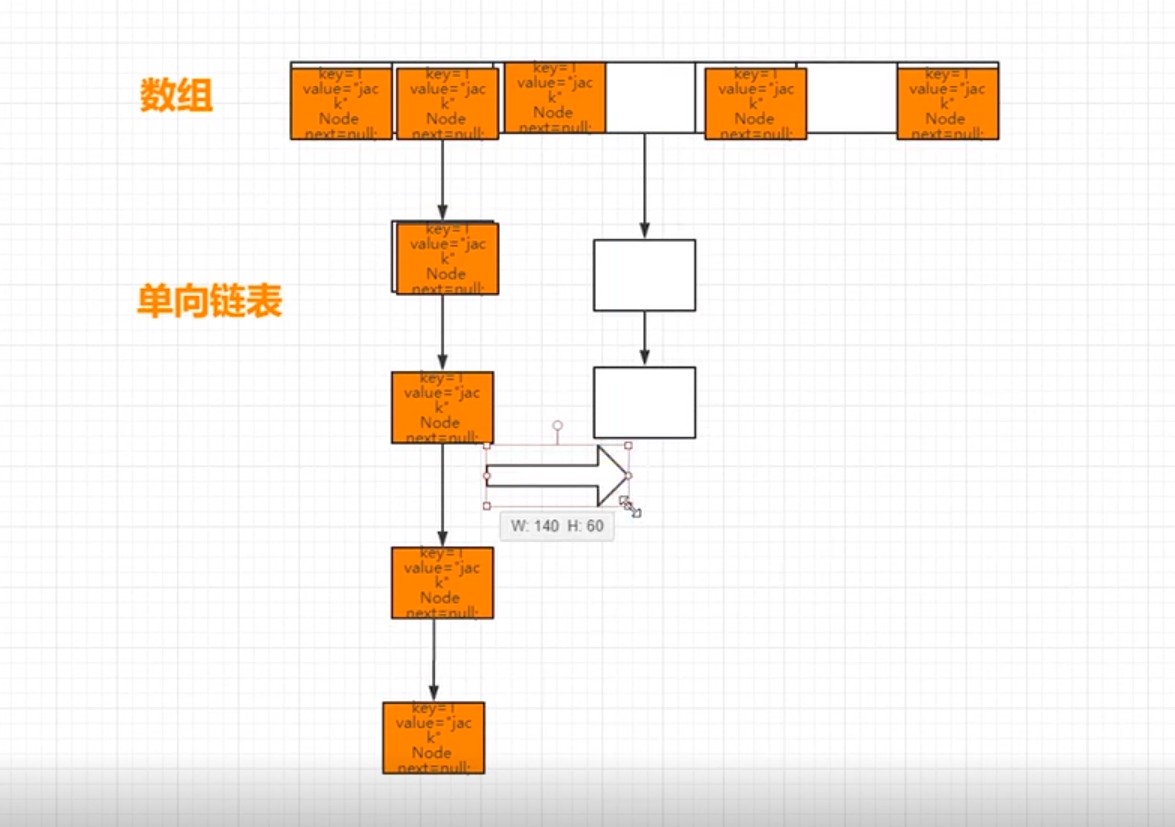
1. 自我介绍
2. 介绍一下项目
3. 你的并发项目有做过压测吗
4. HashMap的底层实现原理，

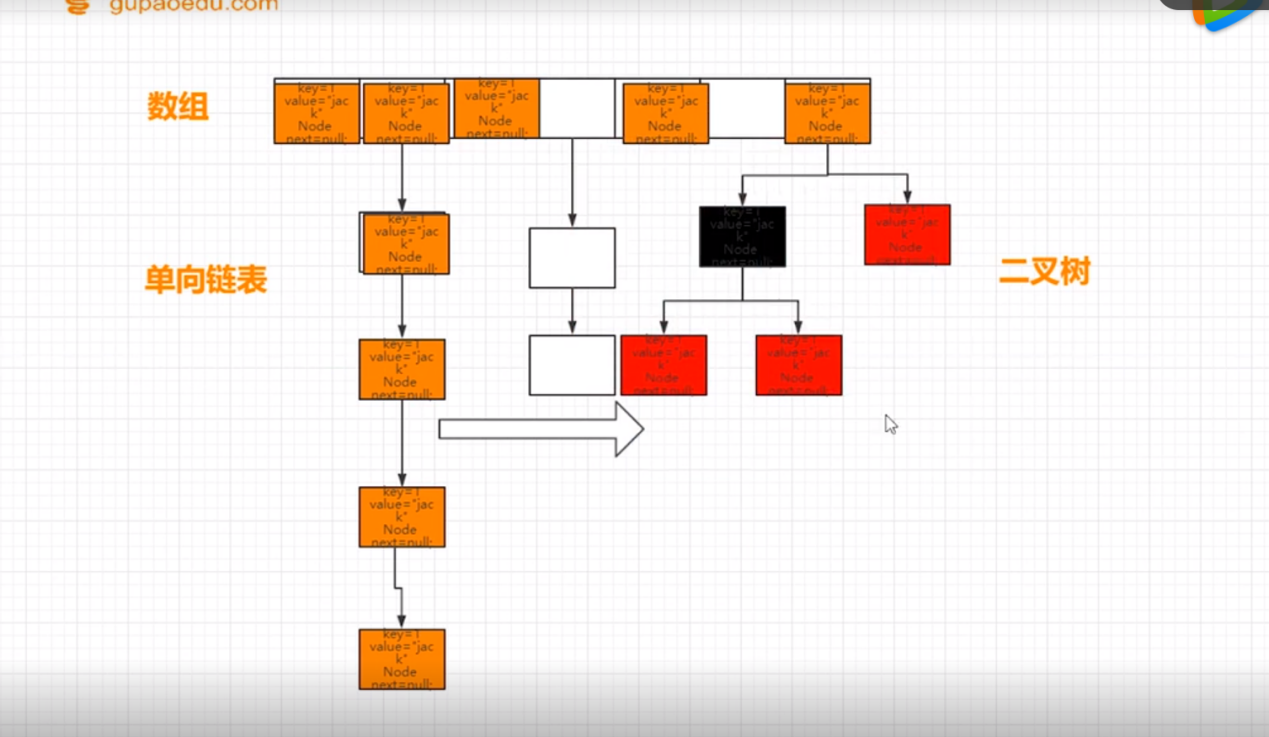


1. HashMap的数据结构是一个数组+单向链表
2. 如何计算Node在数组中的位置

* index(即为0-15的值)，这样的算法就成为Hash算法,
* 取模法 key的hashCode %数组的长度

1. 形成单向链表是因为哈希冲突， 比如 key值为13 % 数组的长度为16 =3, key为3. 3%数组的长度16=3， 那么， key为3的Node{key, value, Node next}就会代替原先key为16的Node 的桶位， 返回旧的key为16的value值，并且key为3的Node 的next 指向key为16的Node，
2. 当单项链表的长度>=8 ,就开始变性数据结构（变成一个有颜色二叉树即为红黑树，特点是左边的权值都小于右边的权值）
3. 当红黑树的节点小于6后又转为链表

如下图：



1. HashMap怎么解决冲突问题

解决Hash冲突的几种方法：

(1`).开放定址法

线性探查法：当发生哈希冲突时， 从冲入的位置开始，依次判断下一个单元是否为空，当达到最后一个单元时， 再从首表依次判断， 直到碰到空闲的单元

平方探查法：加上 1²、 2²等。即d[i] + 1²，d[i] + 2², d[i] + 3²…直到找到空闲单元。

双散列函数探查法

（2）链地址法

链接地址法的思路是将哈希值相同的元素构成一个同义词的单链表，并将单链表的头指针存放在哈希表的第i个单元中，查找、插入和删除主要在同义词链表中进行。链表法适用于经常进行插入和删除的情况。

（3） 再哈希法

就是同时构造多个不同的哈希函数：   
Hi = RHi(key) i= 1,2,3 … k;   
当H1 = RH1(key) 发生冲突时，再用H2 = RH2(key) 进行计算，直到冲突不再产生，这种方法不易产生聚集，但是增加了计算时间

（4）建立公共溢出区

将哈希表分为公共表和溢出表，当溢出发生时，将所有溢出数据统一放到溢出区。

## HashMap是否线程安全

<https://mp.weixin.qq.com/s/oA-uEbzNYA4KYwLtRWXRVw>

线程安全：在单线程还是多线程的执行下永远都能获得一样的结果， 那代码就是线程安全的

线程安全的级别：

1. 不可变：被final修饰的类，如String, Flaot, Double,等，默认final类里的方法都是final修饰，可以被继承，但是不能被覆盖，任何一个线程都不能改变他的值，要改变除非创建一个新的对象，这些对象不需要任何同步手段就能在多线程环境下使用
2. 绝对线程安全：不管运行环境如何，都不需要加额外的同步措施，有CopyOnWriteArrayList, CopyOnWriteArraySet
3. 相对线程安全:通常所说的线程安全，如Vector, HashTable, , add、remove、方法都是原子性操作，不会被打断，但如果有个线程在遍历某个vector, 有个线程同时add ，也会出现ConrrentModificationException, 就是fail-fast机制
4. 线程不安全： 如ArrayList, HashMap,LinkedList等

## Object对象的常用方法

getClass();

class java.lang.Object

hashCode();



equals(Object );



toString(); //类名加存储地址java.lang.Object@53bd815b



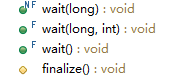
notify();



notifyAll();



wait();



1. 有什么想问的