**Redis槽道原理**

1. **Redis集群计算Key的算法**

* Redis集群计算Key的算法

<https://segmentfault.com/a/1190000018094567?utm_source=tag-newest>

**Key=CRC16(name)%16384=5798 所以key的范围0-16383**

**Key= CRC16(location)%16384=7307**

* Redis之间进行数据交换的时候消息头所占空间为CLUSTER\_SLOTS/8，

也就是16384÷8÷1024=2kb

**1.槽道原理解决的2个问题**

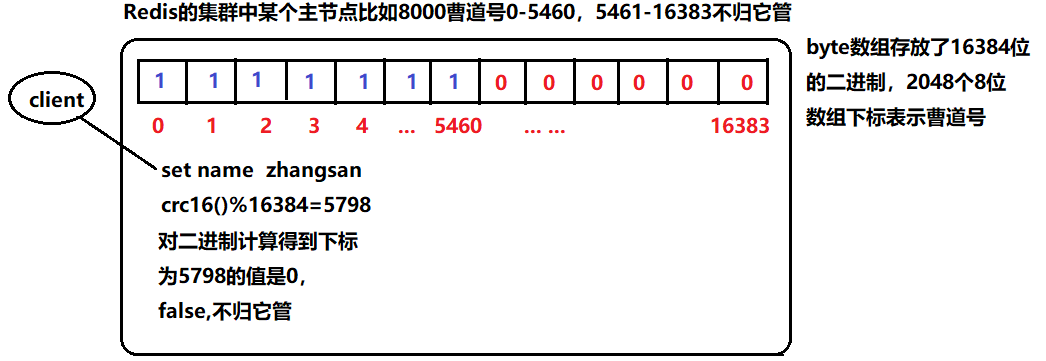
* 节点redis-server接收写命令之后,根据key的计算槽道号,判断归谁管?
* 如果不属自己所管,归谁管，从哪获取负责管理得节点信息

**2.槽道原理**

**2.1槽道的组成部分**

* 16384位的二进制,本质上一个2048个元素的byte数组（1个8位，8\*2048=16384）;通过对二进制的计算,获取(自定义的)下标,下标正好对应的槽道号,每个维护槽道的节点都会存储这样一个二进制的值,管理的和不管理的槽道号都会在二进制中体现,管理的槽道号对应的二进制值是 "1"/true,不管理的就是"0"/false,当一个key值传递到节点上做hash取模计算之后得到的0-16383的整数,到二进制判断结果1/0
* 16384个元素的数组,数组的元素值是一个引用的变量,该变量指向内存中保存的集群所有节点的信息的对象数据,每个节点都维护了一个完全一样的数组对象,只要判断槽道所属权是false将会调用这个数组的元素值(利用数组下标 0-16383),元素值的意义就是下标对应槽道号的管理节点信息的引用对象

**2.2二进制(位序列)**



* **二进制的值的特点:**

**主节点:** 有槽道管理权的主节点,各自二进制不相同,自己管理的槽道的二进制是1,不是自己管理的槽道的二进制全是0;没有槽道管理权的节点.二进制值是16384位都是0

**从节点:**从节点没有槽道管理权，由于它备份了主节点的位序列，在key的槽道号计算完毕后,直接到索引数组中获取节点信息

* **二进制的值赋值**

创建集群时将该主节点管理的槽道号对应的二进制值从0变成1

* **二进制怎么算的下标**

对二进制的位移进行计算;

public void test(){

byte binary=126;**//全是32位。1111110，前面都是0**

//槽道号0 管理权fasle

//槽道号1 管理权true

//槽道号2,管理权true

//槽道号7,管理权false

System.out.println(Integer.toBinaryString(binary));

for(int i=0;i<8;i++){

//计算126对应的槽道号的二进制

// 01111110

int result=binary>>(7-i)&1;

//System.out.println(Integer.toBinaryString(result));

boolean belong= result==1? true:false;

System.out.println("槽道号为:"+i+";所属权为:"+belong);}}

**2.3索引数组/共享数组**

* 节点相互握手的过程中,每个节点都会记录集群的所有节点的信息对象

文本

低可信度描述已自动生成

创建集群时,需要节点与节点进行握手meet,一旦meet成功相当于把对方的信息对象(ip,id,port,ms关系,time)抓取到了自己的内存管理保存

* 对槽道的相关操作,会在每个节点中的数组对象里保存一个引用值

图示

描述已自动生成

集群最终创建完毕之后,每个节点内的数组结构

图示, 示意图

描述已自动生成

有了数组结构,对于第二个问题就可以解决了

图示

描述已自动生成

* 数组中每一个下标对应的都是槽道号,对应元素内容,除了有管理槽道的节点的对象引用意外,还有其他数据(key值的对应关系)