参考文档

https://blog.csdn.net/weixin_39685836/article/details/110397523

总结

Go语言的map，**底层是哈希表实现的，通过链地址法解决哈希冲突**，它依赖的核心数据结构是数组加链表。

map中定义了2的B次方个桶，每个桶中能够容纳8个key。根据key的不同哈希值，将其散落到不同的桶中。哈希值的低位（哈希值的后B个bit位）决定桶序号，高位（哈希值的前8个bit位）标识同一个桶中的不同 key。

当向桶中添加了很多 key，造成元素过多，超过了装载因子所设定的程度，或者多次增删操作，造成溢出桶过多，均会触发扩容。

扩容分为增量扩容和等量扩容。增量扩容，会增加桶的个数（增加一倍），把原来一个桶中的 keys 被重新分配到两个桶中。等量扩容，不会更改桶的个数，只是会将桶中的数据变得紧凑。不管是增量扩容还是等量扩容，都需要创建新的桶数组，并不是原地操作的。

扩容过程是渐进性的，主要是防止一次扩容需要搬迁的 key 数量过多，引发性能问题。触发扩容的时机是增加了新元素， 桶搬迁的时机则发生在赋值、删除期间，每次最多搬迁两个 桶。查找、赋值、删除的一个很核心的内容是如何定位到 key 所在的位置，需要重点理解。一旦理解，关于 map 的源码就可以看懂了。

原理

golang中的map数据结构包括 **数组+链表**

数组是用于存储多个hash后的key, 链表是用于解决哈希冲突

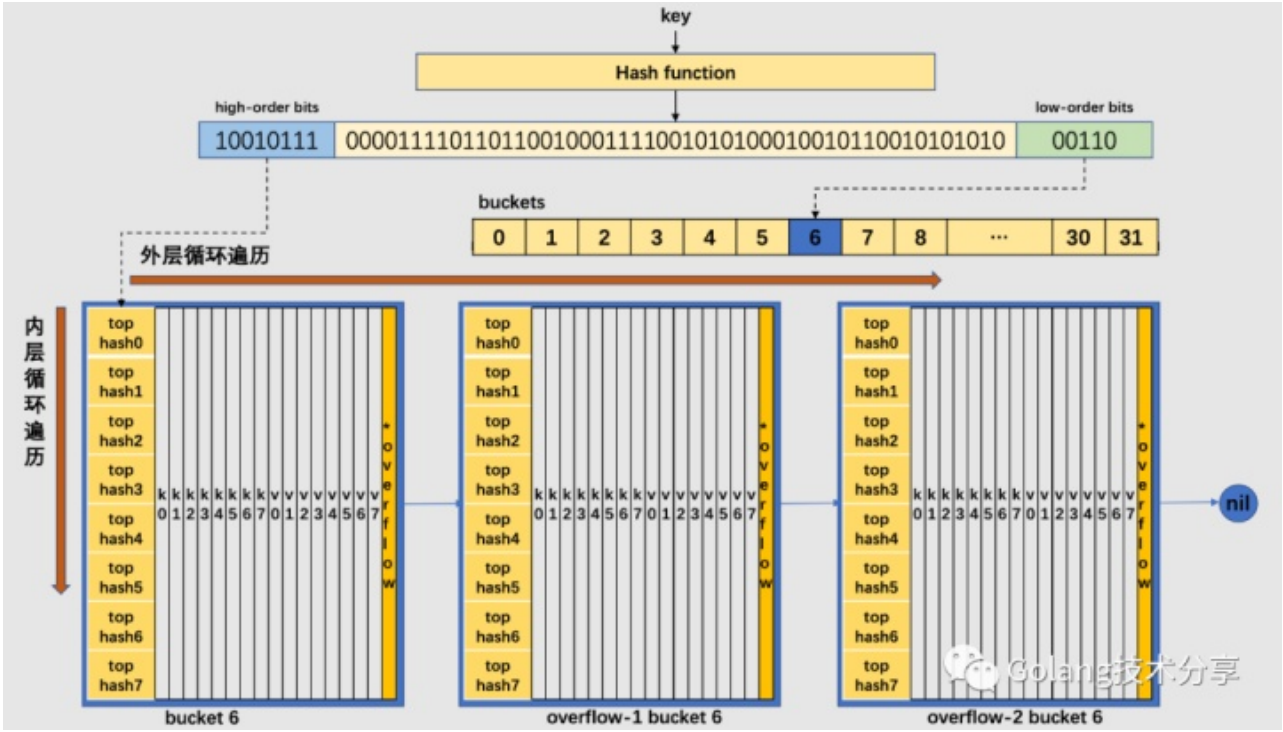
寻找key的过程

其中key 先经过hash函数 ->

使用哈希低位 用于寻找自己的Bucket ->

使用哈希高位用于内层遍历寻找自己在Bucket中的位置

最后比对哈希值是否相等



map扩容

装载因子 = 存放的key数量/全部能存放的数量

在go的map扩容中，除了装载因子会决定是否需要扩容，溢出桶的数量也是扩容的另一关键指标。

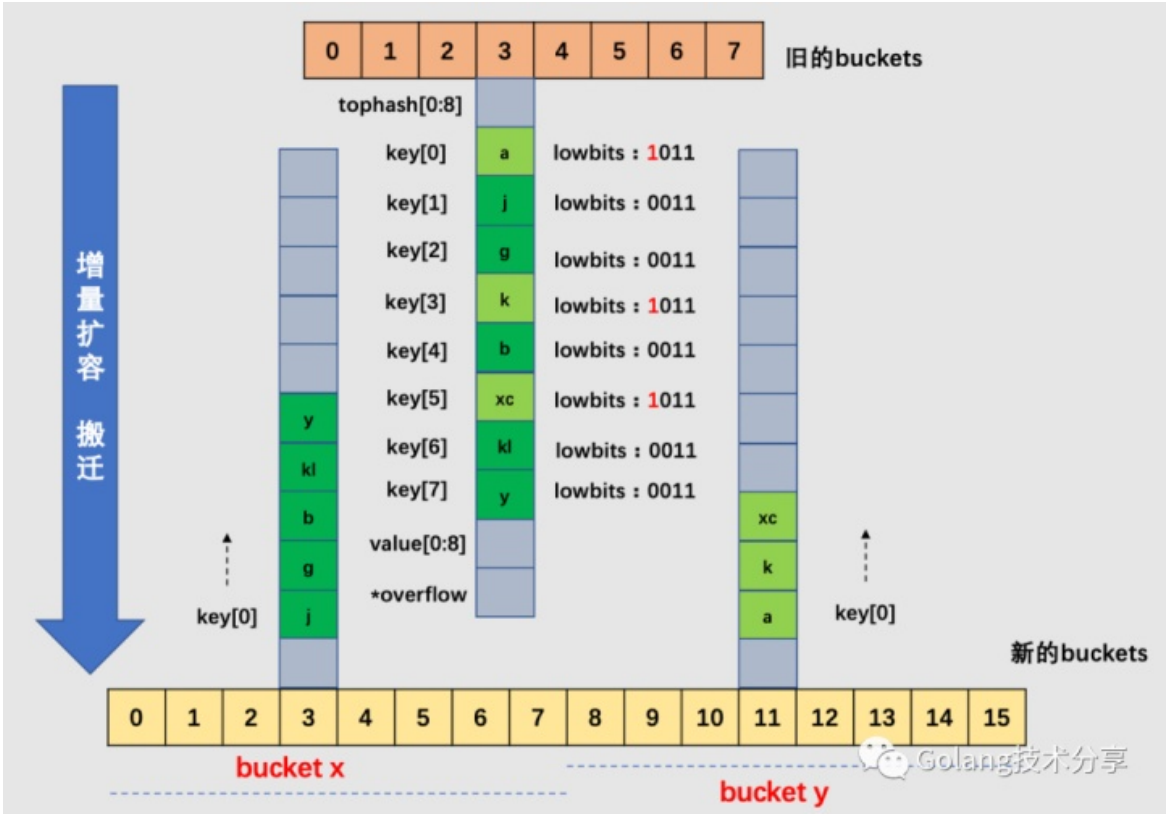
当map添加修改操作时，都检查是否需要扩容：

条件如下：

1- 已经达到装载因子的临界点，即元素个数 >= 桶（bucket）总数 * 6.5，这时候说明大部分的桶可能都快满了（即平均每个桶存储的键值对达到6.5个）

针对1 使用的是增量扩容：

新建一个buckets数组，新的buckets大小是原来的2倍，然后旧buckets数据搬迁到新的buckets。



2- 判断溢出桶数量：如果溢出桶总数 >= 桶总数，则认为溢出桶过多

针对2 使用的是等量扩容

并不扩大容量，buckets数量维持不变，重新做一遍类似增量扩容的搬迁动作，把松散的键值对重新排列一次，以使bucket的使用率更高，进而保证更快的存取