## **Go 中切片扩容的策略是这样的：**

* 首先判断，如果新申请容量（cap）大于2倍的旧容量（old.cap），最终容量（newcap）就是新申请的容量（cap）
* 否则判断，如果旧切片的长度小于1024，则最终容量(newcap)就是旧容量(old.cap)的两倍，即（newcap=doublecap）
* 否则判断，如果旧切片长度大于等于1024，则最终容量（newcap）从旧容量（old.cap）开始循环增加原来的 1/4，即（newcap=old.cap,for {newcap += newcap/4}）直到最终容量（newcap）大于等于新申请的容量(cap)，即（newcap >= cap）

## **Map**

Go 语言使用拉链法来解决哈希碰撞的问题实现了哈希表。哈希在每一个桶都只能存储 8 个键值对，一旦当前哈希的某个桶超出 8 个，新的键值对就会被存储到哈希的溢出桶中。

在以下两种情况发生时触发哈希的扩容：

1. 装载因子已经超过 6.5；
2. 哈希使用了太多溢出桶；
3. 当Map超过了负载因子大小会进行双倍重建,溢出桶太大会进行等量重建。数据的转移采取了写时复制的策略，即在用到时才会将旧桶的数据打散放入到新桶中。
4. Map有多种初始化的方式,如果指定了长度N,在初始化时会生成桶。桶的数量为log2(N).如果map的长度大于了2^4，则会在初始化的时候生成溢出桶。溢出桶的大小为2^(b-4),b为桶的大小
5. map禁止并发读写的设计原因,即为了大部分程序的效率而牺牲了小部分程序的安全。

## **为什么遍历 Go map 是无序的**

在本文开始，咱们先提出核心讨论点：“为什么 Go map 遍历输出是不固定顺序？”。而通过这一番分析，原因也很简单明了。就是 for range map 在开始处理循环逻辑的时候，就做了随机播种…

你想问为什么要这么做？当然是官方有意为之，因为 Go 在早期（1.0）的时候，虽是稳定迭代的，但从结果来讲，其实是无法保证每个 Go 版本迭代遍历规则都是一样的。而这将会导致可移植性问题。因此，改之。也请不要依赖…