Go hash

一、Go 语言里的 hash 结构：

1. 哈希函数：Go 语言的标准库提供了多种哈希函数，如 crypto/md5、crypto/sha1、crypto/sha256 等，这些函数可以将任意长度的数据转换为固定长度的哈希值。

2. 哈希表（map）：Go 语言中的 map 类型实现了一个哈希表，它使用哈希函数来映射键到值的存储位置。map 的底层实现不是直接暴露给用户。map 使用键（key）来存储和检索值（value），并且内部使用哈希算法来实现高效的键值对存储和检索。

二、出现 hash 键冲突如何处理：

1.Hash 键冲突是因为由于两个不同的键 Key 经过 hash 函数计算产生了两个相同的哈希值被映射到哈希表的同一位置。哈希表是通过哈希值来快速定位 Value 的。

2.处理策略：

（1）分离链接法：

使用链表或其他数据结构来存储冲突的键值对。当发生哈希冲突时，将冲突的键值对添加到对应位置的链表中。这种方法可以有效地处理冲突，但会增加额外的内存开销。

（2）开发寻址法：

当键冲突发生时，通过一定的探测方式（如线性探测、平方探测等）来寻找下一个可用的槽位。可以减少额外的内存开销，但可能导致聚集现象和性能损失。

1>线性探测：

插入[64，63，54，32，31]

	空表	插入 64	插入 63	插入 54	插入 32	插入 31
0				54	54	54
1						31
2					32	32
3			63	63	63	63
4		64	64	64	64	64

插入 54 的时候在 64 后按顺序插入。

2>平方探测：

3>[64，63，54，34，32，31]

	空表	插入 64	插入 54	插入 34	插入 32	插入 31	
0				54	54	54	
1					31	31	
2					32	32	
3				34	34	34	
4		64	64	64	64	64	

插入 34 的时候 64 后的位置有了 54，所以进行 2^2=4，向后移动 4 格存储。

三、hash 的扩容：

1.go 语言中 map 会在以下情况下扩容：

（1）当装载因子超过某个阈值（通常是 6.5，即平均每个槽有 6.5 个 key）时。

（2）当使用太多的溢出桶（即超过一定数量的普通桶后，新加入的键值对会被放到溢出链中）。

2. hash 的扩容方式：

（1）等量扩容：当数据不多但溢出桶太多时，会进行等量扩容，即重新组织数据，使数据更紧凑。

（2）翻倍扩容：当数据量很大时，会增加普通桶的数量来进行扩容。

3. hash 等量扩容和翻倍扩容的好处：

等量扩容：指的是在扩容过程中保持哈希表的大小不变，而是通过其他方式优化其性能。这种策略的好处在于它避免了因扩容而导致的内存重新分配和数据迁移的开销，从而维持了哈希表的稳定性。等量扩容可能涉及到对哈希函数或内部数据结构的调整，以更好地分布数据并减少冲突，从而提高查找和插入的效率。这种策略在数据量相对稳定，但性能需要进一步优化的情况下可能更为适用。

翻倍扩容：即每次扩容时将哈希表的大小增加一倍，是一种更为常见的扩容策略。它的好处在于能够更均匀地重新分布已有的元素，从而减少哈希冲突的概率。通过翻倍扩容，可以增加桶的数量，使得每个桶中的元素数量减少，从而提高查找效率。此外，翻倍扩容通常能够在性能和内存占用之间找到一个平衡点。在数据量不断增长的情况下，翻倍扩容可以有效地维持哈希表的性能，避免频繁的性能下降。

四、hash 的查找流程

1. 计算哈希值：首先，根据要查询的元素的键（Key），通过哈希函数计算出哈希值（Hash Value）。哈希函数是一种将任意长度的输入（键）映射为固定长度的输出（哈希值）的算法。这个哈希值用于确定元素在哈希表中的位置。

2. 确定索引位置：使用计算出的哈希值与哈希表的容量进行某种运算（如取模运算），得到元素在哈希表中的索引位置。这个索引位置就是元素在哈希表中可能的存储位置。

3. 检查索引位置：在得到的索引位置上，检查是否存在元素。如果没有任何元素，说明哈

希表中不存在该键对应的元素，查找失败。

4. 处理冲突：如果在索引位置上存在元素（可能是一个元素，也可能是一个链表或红黑树等数据结构），则需要处理哈希冲突。对于单个元素，直接比较其键与要查询的键是否相等。如果存在链表或红黑树等数据结构，则需要遍历这些数据结构，逐个比较键的值，直到找到与要查询的键相等的元素。

5. 返回结果：如果找到了与要查询的键相等的元素，则返回该元素对应的值。如果遍历完所有可能的元素后仍未找到匹配的键，则查找失败，返回相应的空值或错误提示。

五、hash 的插入流程

1. 计算哈希值：首先，根据要插入的元素的键（Key），通过哈希函数计算出对应的哈希值。这个哈希值用于确定元素在哈希表中的存储位置。

2. 确定索引位置：使用计算出的哈希值与哈希表的容量进行某种运算（如取模运算），得到元素在哈希表中的索引位置。这个索引位置就是元素应该插入的位置。

3. 处理冲突：如果在确定的索引位置上已经存在元素，就需要处理哈希冲突。处理冲突的方法有多种，例如链地址法（开放寻址法的一种）和红黑树法等。在链地址法中，每个桶（即索引位置）可以是一个链表，当冲突发生时，新元素会被添加到该桶对应的链表的末尾。而在红黑树法中，当链表过长时，会将其转化为红黑树以提高查询效率。

4. 插入元素：如果没有冲突，或者冲突已经得到妥善处理，就可以将元素插入到哈希表的指定位置。这可能涉及到在链表或红黑树中插入新节点。

5.调整容量：如果插入元素后，哈希表的容量达到了某个阈值（例如，当哈希表中的键值对数量超过了容量的某个百分比），就需要进行扩容操作。扩容操作会创建一个新的哈希表，并将原哈希表中的所有元素重新计算哈希值，并分布到新的哈希表中。这样可以确保哈希表保持良好的性能，避免过多的冲突。