Golang随记（map）

- map核心：
  - 基于key-value（key数据类型必须为可比较的类型，slice，map，func不可比较，chan是可比较的本质是指针）
  - 基于key维度实现存储数据的去重
  - 读，写，删操作时间复杂度为O（1）
- map初始化（写操作，map必须初始化，否则会panic）：
  - make(map，大小)
  - make（map），大小默认为0
  - ：=map直接初始化赋值
- 读取map中的val，推荐使用，ok去通过bool值去判断是否真的存在，否则如果不存在贼会返回对应数据类型的零值
- 删除map中的数据通过delete（map，key）
- 遍历map可以通过for k，v：= range map，注意遍历不是幂等操作，多次遍历结果可能顺序不一致
- map本身是不支持并发安全的，存在并发读写会fatal error
  - 并发读没问题
  - 读的时候其它goroutine在并发写（写入，删除，更新）不行
  - 写的时候其它goroutine在并发写不行
- map又称为hash map，基于hash（将任意长度的输入压缩到某一固定长度的输出摘要的过程，hash可能会产生哈希冲突（输入域大于输出域），此外哈希不可逆，但哈希具有幂等性）实现key的映射和寻址，数据结构上基于桶数组实现key- value对的存储
- key- value的存储和写入基于桶数组
  - 一个map有多个桶，key进行hash计算并离散化后被分配到这些桶中
  - 由于桶的数量是很小的，此外还可能产生哈希冲突，因此每个桶都可能保存不止一个数据，也就是通过链地址法去解决，再对这个链表进行一个遍历
  - 每个桶固定只能存8对数据（这八对是连续的），超过的数据通过创建新的桶，并通过拉链法去连接（另外一种方法是开放寻址法，这个可以利用局部性原理，缓存）
- 如果桶太少了，而数据太多了，那每个桶承载的数据就很多，O(1)的时间复杂度可能会退化为O(n)，因此需要对map进行扩容：
  - 扩容分为增量扩容
  - 等量扩容，overflow的bucket，很多bucket数据较少，扩容后总的bucket不变
  - 扩容一定是在写的时候触发的
- 增量扩容
  - key-value总数/桶数组长度>6.5，发生增量扩容，新的桶是老的桶的大小两倍（保持2的整数次幂）
  - 将老的桶的数据迁移到新的桶（要不存在老桶对应的索引，要不存在老索引加老桶长度的索引位置）
  - 集中迁移会引起性能抖动，因此采用渐进迁移，新写入的数都由新的桶去承载，查数据时如果新的桶查不到，查不到再去查老的桶
  - 每次触发写，删除操作都会完成两组桶的数据迁移
    - 一组是当前操作的桶
    - 未迁移索引最小的桶
- 实现并发安全的map
  - 使用读写锁，我们可以将map的读写操作进行封装，不用手动的去加锁和释放锁
  - sync.map是go标准库提供的并发安全的map实现，主要包括read和dirty两个map，

通过读写分离，使用只读数据结构和锁保证并发安全

- read是只读的，包含当前所有可见的键值对，读操作大多只在这个结构上进行（找不到才去dirty中找），无需加锁
- dirty包含可能被修改的键值对，写操作时如果read中没有那个key（有的话直接更新）先在这个上面进行，并在合适的时候将数据进行同步到read中，

- sync.map适合读多写少，对于写多读少的场景，可以用分片读写加锁，对map进行分片，降低map写操作加锁阻塞的概率