Golang随记（context）

- context在使用中经常被放到入参的第一个。是我们调用链路的上下文，有生命周期，是和彼此之间的调用链强相关。主要在异步场景中用于实现并发协调以及对goroutine的生命周期控制。此外，context还有一定的数据存储能力。
- context本身是个interface，定义了一系列的方法
  - deadline，返回context的过期时间，不是所有的context都有过期时间
  - done，返回标识context是否关闭的channel（只读的struct {}）
  - err，返回错误，比如过期和被手动取消
  - value，返回存储的key- value
- 既然context是个interface，那就有多种实现
  - emptycontext，本质上是一个整形，context.background()和todo()也都是这个
    - 永不过期（返回公元元年
    - 返回的channel也是nil，任何goroutine尝试去读写都会阻塞，陷入一个死锁
    - 返回错误也为nil
    - value返回也为nil
  - cancelcontext
    - 父context
    - lock
    - 标识context是否结束的channel
    - children，用map[]struct{}实现的set，包含子context
    - err
    - 没有实现deadline方法，而是复用父context的deadline方法
    - 通过withCancel传入一个父context，返回一个cancelcontext和其cancel方法（func () {cancel}）的闭包
    - cancelcontext取消时会遍历其children去取消
  - timercontext
    - cancelcontext，可以复用其cancel
    - timer，用于在过期时间中止context，是持续时间
    - deadline，ctx的过期时间，是截止时间
    - 通过withtimeout或者withdeadline去创建
  - valuecontext，数据存储
    - context，父
    - key，value any，每个value context只会有一对key- value，如果有多对key-value，则会通过value依次向上去从父节点寻找，是个遍历过程，查找是O(n)。因为是查找过程是链表，而不是使用map，因此查找具有不确定性，取决于起点。
    - 通过context.withvalue创建
    - 不适合作为存储介质，存放大量的kv数据，只适合存储少量作用域较大的全局meta数据
      - n个kv对需要嵌套n个valuectx，空间浪费
      - 查找需要O(n)
      - 不去重
- 控制并发常见的两种方式
  - waitgroup：多个goroutine执行同一个事情，我们去等待goroutine执行完毕。
  - context：适合主动去停止goroutine的场景（当然我们也可以通过select+channel），但是场景是多个goroutine呢，或者goroutine内又有goroutine呢，就需要context了。