怎么实现一个并发的词拼统计应用?

- 需求整理：
  1. 统计多个文件中英文单词出现的次数
  2. 按照词频从多到少排序输出
  3. 支持并发
- 怎么设计算法
  - 自定义一个 map[string]int 类型作为词频统计，然后实现相应的接收器方法
  - 首先需要一个从文件中读取数据的方法，参数是 string 类型的文件名，因为字典类型是传引用，所以不需要返回值，直接在方法内部更新词频统计
  - 上面方法只是统计一个文件的，需要一个并发的统计所有文件的方法，调用上面的方法并进行协程间消息传递
  - 消息传递的位置是在启动了并发操作的协程里面，需要传递两类值，一类是词频；一类是一个文件结束的标志，用于控制主协程的循环
- 局部数据结构：
  - 存储单词和次数的字典结构：map[string]int
  - 存储文件的 byte 切片：[]byte
  - 传递词频信息的通道 result：chan string
  - 传递一个文件统计完成信息的通道 done：chan bool

- 非并发算法流程：

  1. 首先调用 os.Open(文件名)函数打开一个文件
  2. 然后调用文件的 Read(存储文件的 byte 切片)函数将文件数据读取到 byte 切片中
  3. 遍历 byte 切片统计词频
  4. 对字典结构进行排序并输出

```go
package main

import (
  "fmt"
  "os"
)

var ans = make(map[string]int)


func account(article *[]byte)  {

  str := string(*article)
  //fmt.Println(str)
  now := ""
  for _,val := range str {
    if val >= 'a' && val <= 'z' || val >= 'A' && val <= 'Z' {
      now += string(val)
    } else {
      if now != "" {
        ans[now] += 1
        fmt.Printf("%s : %d\n", now, ans[now])
      }
      now = ""
    }
  }
}

func main() {
  //1.打开文件
  f, err := os.Open("a.txt")
  if err != nil {
    fmt.Println(err.Error())
  }
  defer f.Close()

  //2.读取文件内容
  var b []byte = make([]byte, 2*1024)
  _, err = f.Read(b)
  if err !=nil {
    fmt.Println(err.Error())
  }
  //fmt.Println(n, string(b))

  //3.统计词频
  account(&b)
  fmt.Println(ans)
}
```

Go

- 并发算法改造：

  1. 自定义 Pair 结构和 PairList 结构，并实现 PariList 的排序接口
  2. 在匿名协程方法中更新词频统计，并将字典结构转换成 Pair 类型发送到通道
  3. 在主协程中监听通道并汇总词频统计