# **Go 语言和词频统计**

## **实验简介**

本节实验是 Go 语言编程实验的最后一节，我们将编写一个词频统计程序，我们希望该词频统计应用有以下功能：

* 统计多个文件中英文单词出现的次数
* 按照词频从多到少排序输出
* 支持并发

同时我们会将该词频统计程序打成包，以便在其他程序中使用。

## **一、实验说明**

本课程所有源代码，可以在XfceTerminal中通过以下方式克隆到实验环境:

$ git clone http://git.shiyanlou.com/shiyanlou/Golang\_Programming

## **二. 包**

这一课内容作为本课程(Go语言基础课程)最后的课程，我们将学习“包”，同时我们将写一个英文词频统计程序作为本课程的结课练习。

前面我们了解过Go语言组织代码的方式是包，包是各种类型和函数的集合。在包中，如果标示符(类型名称，函数名称，方法名称)的首字母是大写，那这些标示符是可以被导出的，也就是说可以在包以外直接使用。前面我们也提到了$GOPATH环境变量（指向一个或多个目录），以及其子目录src目录的，当我们使用import关键字导入包的时候，Go语言会在$GOPATH，GOROOT目录中搜索包。

我们创建的自定义的包最好放在$GOPATH的src目录下，如果这个包只属于某个应用程序，可以直接放在应用程序源代码的子目录下，但如果我们希望这个包可以被其他的应用程序共享，那就应该放在$GOPATH的src目录下，每个包单独放在一个目录里，如果两个不同的包放在同一目录下，会出现名字冲突的编译错误。作为惯例，包的源代码应该放在一个同名的文件夹下面。同一个包可以有任意多的源文件，文件名的名字也没有任何规定。

## **三. 词频统计程序**

### **1. 准备工作**

作为最后本课程最后一门课，我们将编写一个词频统计程序，我们希望该词频统计应用有以下功能：

* 统计多个文件中英文单词出现的次数
* 按照词频从多到少排序输出
* 支持并发

同时我们会将该词频统计程序打成包，以便在其他程序中使用。在使用该包前，我们需要设置$GOPATH环境变量。在console中(lxterminal)中按照以下步骤操作。

$ cd $HOME

$ mkdir -p golang/src/wordcount

$ export GOPATH=$HOME/golang

$ cd $GOPATH/src/wordcount

以上linux命令中，我们创建了目录golang/src/wordcount，并把$GOPATH设置成$HOME/golang。如果linux当前用户为shiyanlou，那么$HOME和$GOPATH的值应该如下所示.

$ echo $HOME

/home/shiyanlou

$ echo $GOPATH

/home/shiyanlou/golang

### **2. 实现**

词频统计的程序逻辑很简单。我们首先会创建一个映射，然后读取文件的每一行，提取单词，然后更新映射中单词所对应的数量即可。

为了演示面向对象和goroutine的使用，我们将基础映射类型封装成了一个统计单词频率的包。我们在基础映射类型上创建额类型WordCound，然后为该类型了实现了关键方法UpdateFreq()和WordFreqCounter()，其中前者会读取一个文件并统计该文件中的所有单词的词频，后者通过goroutine实现了并发统计。其并发逻辑是：对于每一个文件，创建一个goroutine，在这个goroutine内部调用UpdateFreq()方法统计对应文件的词频，当统计完成以后会将映射中每一对键值转化为Pair结构发送到results通道，并在发送完成时候发送一个空结构体的值到done通道以表示自己的任务已经完成。由于map映射结构不支持并发写操作，所以我们通过result通道来保证每次只有一个goroutine能更新映射。又因为当所有的goroutine结束以后，有可能results通道中还有没来得及处理的数据，所以在WordFreqCounter()的结尾我们又开启了一个for循环处理results通道中的剩余数据。说了这么多，我们直接写代码吧。

在$GOPATH/src/wordcount目录中创建文件wordcount.go，输入以下源码

package wordcount

import (

"bufio"

"fmt"

"io"

"log"

"os"

"sort"

"strings"

"unicode"

"unicode/utf8")

type Pair struct {

Key string

Value int

}

// PariList实现了sort接口，可以使用sort.Sort对其排序

type PairList []Pair

func (p PairList) Swap(i, j int) { p[i], p[j] = p[j], p[i] }

func (p PairList) Len() int { return len(p) }

func (p PairList) Less(i, j int) bool { return p[j].Value < p[i].Value } // 逆序

// 提取单词func SplitOnNonLetters(s string) []string {

notALetter := func(char rune) bool { return !unicode.IsLetter(char) }

return strings.FieldsFunc(s, notALetter)

}

/\*

基于map实现了类型WordCount, 并对期实现了Merge(), Report(), SortReport(), UpdateFreq(),

WordFreqCounter() 方法

\*/

type WordCount map[string]int

// 用于合并两个WordCount

func (source WordCount) Merge(wordcount WordCount) WordCount {

for k, v := range wordcount {

source[k] += v

}

return source

}

// 打印词频统计情况

func (workdcount WordCount) Report() {

words := make([]string, 0, len(workdcount))

wordWidth, frequencyWidth := 0, 0

for word, frequency := range workdcount {

words = append(words, word)

if width := utf8.RuneCountInString(word); width > wordWidth {

wordWidth = width

}

if width := len(fmt.Sprint(frequency)); width > frequencyWidth {

frequencyWidth = width

}

}

sort.Strings(words)

gap := wordWidth + frequencyWidth - len("Word") - len("Frequency")

fmt.Printf("Word %\*s%s\n", gap, " ", "Frequency")

for \_, word := range words {

fmt.Printf("%-\*s %\*d\n", wordWidth, word, frequencyWidth,

workdcount[word])

}

}

// 从多到少打印词频

func (wordcount WordCount) SortReport() {

p := make(PairList, len(wordcount))

i := 0

for k, v := range wordcount { // 将wordcount map转换成PairList

p[i] = Pair{k, v}

i++

}

sort.Sort(p) // 因为PairList实现了排序接口，所以可以使用sort.Sort()对其排序

wordWidth, frequencyWidth := 0, 0

for \_, pair := range p {

word, frequency := pair.Key, pair.Value

if width := utf8.RuneCountInString(word); width > wordWidth {

wordWidth = width

}

if width := len(fmt.Sprint(frequency)); width > frequencyWidth {

frequencyWidth = width

}

}

gap := wordWidth + frequencyWidth - len("Word") - len("Frequency")

fmt.Printf("Word %\*s%s\n", gap, " ", "Frequency")

for \_, pair := range p {

fmt.Printf("%-\*s %\*d\n", wordWidth, pair.Key, frequencyWidth,

pair.Value)

}

}

// 从文件中读取单词，并更新其出现的次数

func (wordcount WordCount) UpdateFreq(filename string) {

var file \*os.File

var err error

if file, err = os.Open(filename); err != nil {

log.Println("failed to open the file: ", err)

return

}

defer file.Close() // 本函数退出之前时，关闭文件

reader := bufio.NewReader(file)

for {

line, err := reader.ReadString('\n')

for \_, word := range SplitOnNonLetters(strings.TrimSpace(line)) {

if len(word) > utf8.UTFMax ||

utf8.RuneCountInString(word) > 1 {

wordcount[strings.ToLower(word)] += 1

}

}

if err != nil {

if err != io.EOF {

log.Println("failed to finish reading the file: ", err)

}

break

}

}

}

// 并发统计单词频次

func (wordcount WordCount) WordFreqCounter(files []string) {

results := make(chan Pair, len(files)) // goroutine 将结果发送到该channel

done := make(chan struct{}, len(files)) // 每个goroutine工作完成后，发送一个空结构体到该channel，表示工作完成

for i := 0; i < len(files); { // 有多少个文件就开启多少个goroutine, 使用匿名函数的方式

go func(done chan<- struct{}, results chan<- Pair, filename string) {

wordcount := make(WordCount)

wordcount.UpdateFreq(filename)

for k, v := range wordcount {

pair := Pair{k, v}

results <- pair

}

done <- struct{}{}

}(done, results, files[i])

i++

}

for working := len(files); working > 0; { // 监听通道，直到所有的工作goroutine完成任务时才退出

select {

case pair := <-results: // 接收发送到通道中的统计结果

wordcount[pair.Key] += pair.Value

case <-done: // 判断工作goroutine是否全部完成

working--

}

}

DONE: // 再次启动for循环处理通道中还未处理完的值

for {

select {

case pair := <-results:

wordcount[pair.Key] += pair.Value

default:

break DONE

}

}

close(results)

close(done)

}

然后在$GOPATH目录中创建文件wordfreq.go，输入以下源码:

package main

import (

"fmt"

"os"

"path/filepath"

"wordcount"

)

func main() {

if len(os.Args) == 1 || os.Args[1] == "-h" || os.Args[1] == "--help" {

fmt.Printf("usage: %s <file1> [<file2> [... <fileN>]]\n",

filepath.Base(os.Args[0]))

os.Exit(1)

}

wordcounter := make(wordcount.WordCount)

// for \_, filename := range os.Args[1:] {

// wordcount.UpdateFreq(filename)

// }

wordcounter.WordFreqCounter(os.Args[1:])

wordcounter.SortReport()

}

### **3. 编译执行**

最后我们编译该程序，输入以下命令:

$ go build wordfreq.go

当运行以上命令后，当前目录已经有了一个可执行文件wordfreq。为了验证该程序，我们使用程序统计官方包os中的英文单词的词频。Go语言一门开源的语言，所有的官方包都可以在Go语言的安装目录下看到。首先输入命令:

$ go env

GOARCH="amd64"

GOBIN=""

GOCHAR="6"

GOEXE=""

GOHOSTARCH="amd64"

GOHOSTOS="linux"

GOOS="linux"

GOPATH=""

GORACE=""

GOROOT="/usr/lib/go"

GOTOOLDIR="/usr/lib/go/pkg/tool/linux\_amd64"

TERM="dumb"

CC="gcc"

GOGCCFLAGS="-g -O2 -fPIC -m64 -pthread"

CXX="g++"CGO\_ENABLED="1"

可以看到GOROOT的指向的目录为/usr/lib/go，则os包的源码路径为/usr/lib/go/src/pkg/os，下面让我们统计下该目录下所有源文件的词频率，为了方便输出我们只打印了排名前5的单词：

$ ./wordfreq /usr/lib/go/src/pkg/os/\*.go |head -n 6

Word Frequency

err 811if 722

the 576nil 545return 539

## **作业**

请使用Go语言实现一个小型计算器程序。

到目前为止，这门课程就结束了。在这门课程中我们学习了Go语言的基础知识，如数据类型、过程面层、面向对象编程等。可以看到Go语言的关键词非常少，使用起来相当灵活，相对于C、C++等语言很大程度上提高了开发效率。想要进一步熟练Go语言，多看别人的源码，多加练习，多做项目即可，贵在坚持。