**Golang分片读取http超大文件流和并发控制**

**分片读取http超大文件流**

Golang中的HTTP发送get请求，在获取内容有两种情况。  
Golang发送http get请求方式

resp, err := http.Get(sendUrl)

if err != nil {

fmt.Println("出错", err)

return

}

* 1
* 2
* 3
* 4
* 5

**第一种方式**是直接全部读取出来，这种方式在小数据量的时候很方便。  
body变量直接全部接收resp响应内容

body, err2 := ioutil.ReadAll(resp.Body)

* 1

**第二种方式**，分片获取。  
首先顶一个切片buf := make([]byte, 4096) 容量和大小都是4096个char字符（切片就是Golang自己的动态变长数组）

for无限循环读取，读取一块内容放入buf中。**result** 是一个string字符串，然后用 **result** 自动拼接。

for {

n, err:= resp.Body.Read(buf)

if err != nil || n == 0{

fmt.Println("出现错误")

break

}

result += string(buf[:n])

}

* 1
* 2
* 3
* 4
* 5
* 6
* 7
* 8

**Golang中匹配正则**  
MustCompile内容不是单引号，而是 **`**

// 解析，编译正则表达式

ret := regexp.MustCompile(`<img width="100" alt="(?:(.\*?))"`)

//提取需要信息

var fileName [][] string = ret.FindAllStringSubmatch(result, -1)

* 1
* 2
* 3
* 4

该方法可以进一步推广，可以推广到下载超大文件，然后分片读取和写入硬盘。  
然后再开启另一个线程去将这些已经写入硬盘的分片合称在一起。

**Golang 并发控制**

Java中的多线程，如果主线程开启多个子线程后，在多个子线程全部完成后，主线程才会退出，程序终止。  
而在Golang中不会出现此类情况，**主Go程**和**子Go程**需要人为控制**主GO程**和**子GO程**同步，否则**主GO程**瞬间结束。  
在**主GO程**开始执行**子GO程**地方创建一个 channel。**子GO程**开始往 **channel** 中添加数据，**主GO程**再消费数据。

func toWork(start, end int) {

//子Go程与主Go程完成同步，意思是子Go程没有全部执行完毕，主Go程不许退出。

page := make(chan int)

for i:=start; i<=end; i++ {

//开启子GO程

go SpiderPage(i, page)

}

//主GO程开始消费管道中的数据

for i:=start; i<=end; i++ {

fmt.Println("爬取完成", <- page)

}

}

* 1
* 2
* 3
* 4
* 5
* 6
* 7
* 8
* 9
* 10
* 11
* 12
* 13

**往主GO程的channel中添加数据，提供给主GO程消费从而达到同步**

func SpiderPage(index int, page chan int) {

page <- index

}

* 1
* 2
* 3

**读取http二进制码写入图片**

创建文件  
获取http流  
分片逐步写入

func saveImag(sendUrl string, index int, imageChannel chan int) {

//创建文件

path := "C:/img/" + strconv.Itoa(index) + ".jpg"

f, err := os.Create(path)

if err != nil {

return

}

defer f.Close()

//获取http流

resp, err :=http.Get(sendUrl)

if err != nil {

return

}

defer resp.Body.Close()

//分片逐步写入

buf := make([]byte, 4096)

for {

n, err := resp.Body.Read(buf)

if err != nil {

break

}

f.Write(buf[:n])

}

imageChannel <- index

}