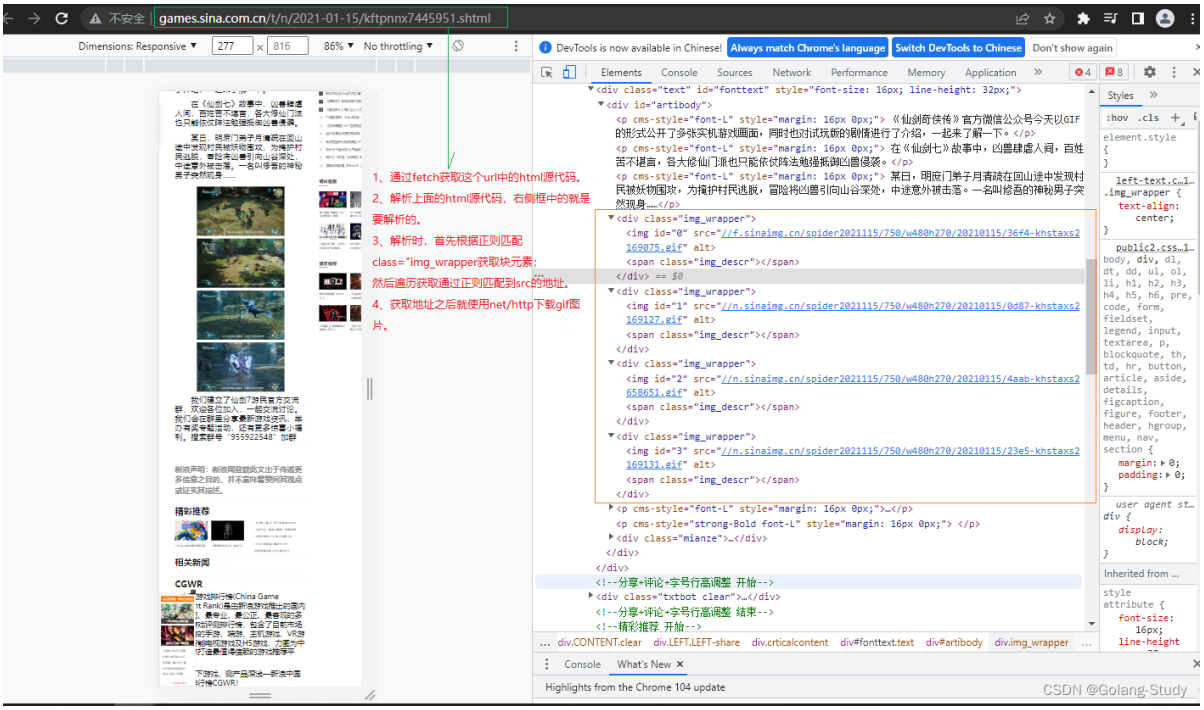
1、net/http爬虫

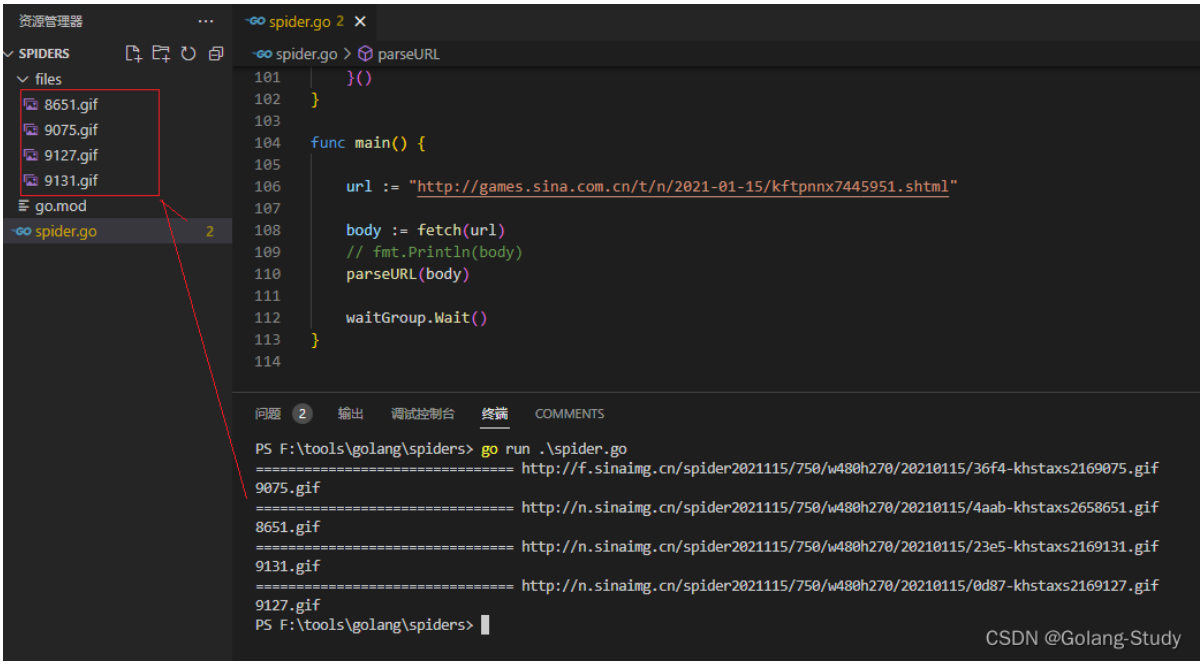
net/http配合正则表达式 爬虫。




```
package main

import (
    "fmt"
    "io/ioutil"
    "log"
    "net/http"
    "os"
    "regexp"
    "strings"
    "sync"
)

// 负责抓取页面的源代码(html)
// 通过http包实现
func fetch(url string) string {

    // 得到一个客户端
    client := &http.Client{}
    request, _ := http.NewRequest("GET", url, nil)

    request.Header.Set("User-Agent", "Mozilla/5.0 (Linux; Android 6.0; Nexus 5 Build/MRA58N) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/104.0.0.0 Mobile Safari/537.36")
    request.Header.Add("Cookie", "test_cookie=CheckForPermission; expires=Tue, 30-Aug-2022 01:04:32 GMT; path=/; domain=.doubleclick.net; Secure; HttpOnly; SameSite=none")

    // 客户端发送请求，并且获取一个响应
    response, err := client.Do(request)
    if err != nil {
```

```
        log.Println("Error: ", err)
        return ""
    }

    // 如果状态码不是200，就是响应错误
    if response.StatusCode != 200 {
        log.Println("Error: ", response.StatusCode)
        return ""
    }

    defer response.Body.Close() // 关闭

    // 读取响应体中的所有数据到body中，这就是需要的部分
    body, err := ioutil.ReadAll(response.Body)
    if err != nil {
        log.Println("Error: ", err)
        return ""
    }

    // 转换为字符串(字节切片 --> 字符串)
    return string(body)
}

var waitGroup sync.WaitGroup

// 解析页面源代码
func parseURL(body string) {

    // 将body(响应结果)中的换行替换掉，防止正则匹配出错
    html := strings.Replace(body, "\n", "", -1)
    // 正则匹配
    re_Img_div := regexp.MustCompile(`<div class="img_wrapper">(.*?)</div>`)

    img_div := re_Img_div.FindAllString(html, -1) // 得到<div><img/></div>

    for _, v := range img_div {

        // img正则
        re_link := regexp.MustCompile(`src="(.*?)"`)
        // 找到所有的图片链接
        links := re_link.FindAllString(v, -1) // 得到所有图片链接

        // 遍历links，切掉不必要的部分src="和最后的"
        for _, v := range links {

            src := v[5 : len(v)-1]
            src = "http:" + src

            waitGroup.Add(1)
            go downLoad(src)
        }
    }

}

// 下载
```

```go
func downLoad(src string) {

    fmt.Println("================================", src)

    // 取一个文件名
    filename := string(src[len(src)-8 : len(src)])
    fmt.Println(filename)

    response, _ := http.Get(src)
    picdata, _ := ioutil.ReadAll(response.Body)

    image, _ := os.Create("./files/" + filename)
    image.Write(picdata)

    defer func() {
        image.Close()
        waitGroup.Done()
    }()
}

func main() {

    url := "http://games.sina.com.cn/t/n/2021-01-15/kftpnnx7445951.shtml"

    body := fetch(url)
    // fmt.Println(body)
    parseURL(body)

    waitGroup.Wait()
}
```



2、goquery库爬虫

goquery可以避免操作复杂的正则表达式，它可以直接根据url获取一个Document对象，然后根据标签选择器、类选择器和id选择器获取相应的选择对象，进行自定义的操作。

goquery可以灵活的获取页面中的元素。

*** 一个简单的例子，引出goquery中的重要API

```
package main

import (
    "fmt"
    "strings"

    "github.com/PuerkitoBio/goquery"
)

func main() {

    url := "http://games.sina.com.cn/t/n/2021-01-15/kftpnnx7445951.shtml"

    // 得到页面原文档对象
    d, _ := goquery.NewDocument(url)

    // 根据文档对象借助类选择器获取Selection对象，通过Each遍历所有的适配类选择器的对象
    // Each的参数是一个函数，里面是处理逻辑
    d.Find("img").Each(func(index int, s *goquery.Selection) {

        // 根据属性名获取属性值    一个Selection对象 --> <img
src="http://localhost:8080/images" > text </img>
        text, _ := s.Attr("src")

        // 只处理gif动态图片
        if strings.HasSuffix(text, ".gif") {
            text = "http:" + text
            fmt.Println(text)
        }

    })
}
```

```
PS F:\tools\golang\spiders> go run .\goquery.go
http://f.sinaimg.cn/spider2021115/750/w480h270/20210115/36f4-khstaxs2169075.gif
http://n.sinaimg.cn/spider2021115/750/w480h270/20210115/0d87-khstaxs2169127.gif
http://n.sinaimg.cn/spider2021115/750/w480h270/20210115/4aab-khstaxs2658651.gif
http://n.sinaimg.cn/spider2021115/750/w480h270/20210115/23e5-khstaxs2169131.gif
PS F:\tools\golang\spiders>
```


*** 操作一、获取html整个原文档

分别是`goquery.NewDocument(url string)`、`goquery.NewDocumentFromResponse(*http.Response)`、`goquery.NewDocumentFromReader(*io.Reader)`。三种方式的第一种比较最为方便使用。

```
package main

import (
```

```
    "log"
    "net/http"
    "strings"

    "github.com/PuerkitoBio/goquery"
)

/*
    goquery得到Document对象的3种方式
*/

// 1、通过NewDocument传入一个URL地址
func GetDocument_1(url string) string {

    document, _ := goquery.NewDocument(url)

    document.Find("href")

    return "document.Text()"
}

// 2、通过响应获取。第一种方式是第二种方式的封装
func GetDocument_2(url string) string {

    client := &http.Client{}
    request, _ := http.NewRequest("GET", url, nil)

    response, _ := client.Do(request)
    document, err := goquery.NewDocumentFromResponse(response)

    if err != nil {
        log.Fatalln(err)
    }
    document.Find("")

    return ""
}

// 3、有一个html文本的情况下，读取转换为Document对象
func GetDocument_3(html string) string {

    document, _ := goquery.NewDocumentFromReader(strings.NewReader(html))
    document.Find("")

    return ""
}
```

*** 操作二、选择器
同html的标识方式，在Find函数中。

*** 操作三、Selection相关方法

> Selection可以提取、删除、添加元素、属性内容

内置函数

1）类似函数的位置操作

```
Eq(index int) *Selection //根据索引获取某个节点集
First() *Selection //获取第一个子节点集
Last() *Selection //获取最后一个子节点集
Next() *Selection //获取下一个兄弟节点集
NextAll() *Selection //获取后面所有兄弟节点集
Prev() *Selection //前一个兄弟节点集
Get(index int) *html.Node //根据索引获取一个节点
Index() int //返回选择对象中第一个元素的位置
Slice(start, end int) *Selection //根据起始位置获取子节点集
```

2）循环遍历选择的节点

```
Each(f func(int, *Selection)) *Selection //遍历
EachWithBreak(f func(int, *Selection) bool) *Selection //可中断遍历
Map(f func(int, *Selection) string) (result []string) //返回字符串数组
```

3）检测或获取节点属性值

```
Attr(), RemoveAttr(), SetAttr() //获取，移除，设置属性的值可以获取Selection指向的标签的属性，以及对标签进行操作
AddClass(), HasClass(), RemoveClass(), ToggleClass()
Html() //获取该节点的html  获取Selection指向的html标签元素内部的标签元素，不包括自身
Length() //返回该Selection的元素个数
Text() //获取该节点的文本值  标签标识的文本
```

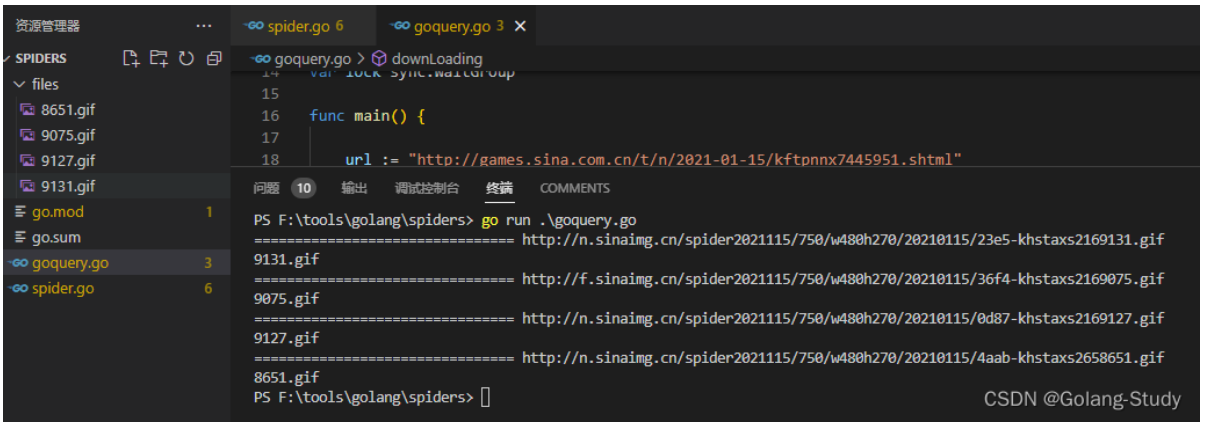
4）在文档树之间来回跳转（常用的查找节点方法

```
Children() //返回selection中各个节点下的孩子节点
Contents() //获取当前节点下的所有节点
Find() //查找获取当前匹配的元素
Next() //下一个元素
Prev() //上一个元素
```


*** 最后来完成net/http中的网页爬虫

```
package main

import (
    "fmt"
    "io/ioutil"
    "net/http"
    "os"
    "strings"
    "sync"

    "github.com/PuerkitoBio/goquery"
)
```

```go
var lock sync.WaitGroup

func main() {

    url := "http://games.sina.com.cn/t/n/2021-01-15/kftpnnx7445951.shtml"

    // 得到页面原文档对象
    d, _ := goquery.NewDocument(url)

    // 根据文档对象借助类选择器获取Selection对象，通过Each遍历所有的适配类选择器的对象
    // Each的参数是一个函数，里面是处理逻辑
    d.Find("img").Each(func(index int, s *goquery.Selection) {

        // 根据属性名获取属性值    一个Selection对象 --> <img
src="http://localhost:8080/images" > text </img>
        text, _ := s.Attr("src")

        // 只处理gif动态图片
        if strings.HasSuffix(text, ".gif") {
            lock.Add(1)

            http := "http:" + text

            // 得到图片地址，开启协程下载图片
            go downLoading(http)
        }

    })

    lock.Wait()
}

func downLoading(src string) {

    fmt.Println("==================================", src)

    // 取一个文件名
    filename := string(src[len(src)-8 : len(src)])
    fmt.Println(filename)

    response, _ := http.Get(src)
    picdata, _ := ioutil.ReadAll(response.Body)

    image, _ := os.Create("./files/" + filename)
    image.Write(picdata)

    defer func() {
        image.Close()
        lock.Done()
    }()
}
```



3、colly框架爬虫

首先要获取一个`*colly.Collector`对象；
然后注册处理函数`OnXxx`函数；
之后就可以访问url了。

*** OnXxx函数**

主要操作都是由OnXxx函数的参数函数进行处理的

```go
package main

import (
    "fmt"

    "github.com/gocolly/colly"
)

func main1() {

    collector := colly.NewCollector()

    collector.OnRequest(func(req *colly.Request) {
        fmt.Println("请求前调用...OnRequest")
    })

    collector.OnError(func(res *colly.Response, err error) {
        fmt.Println("发生错误调用...OnError")
    })

    collector.OnResponse(func(r *colly.Response) {
        fmt.Println("获得响应后调用...OnResponse")
    })
        主要函数，用于系统回调
    collector.OnHTML("a[href]", func(elem *colly.HTMLElement) {
        fmt.Println("OnResponse收到html内容后调用...OnHTML")
    })

    collector.OnXML("//h1", func(elem *colly.XMLElement) {
        fmt.Println("OnResponse收到xml内容后调用...OnXML")
    })

    collector.OnScraped(func(resp *colly.Response) {
        fmt.Println("结束")
    })

    collector.Visit("https://gorm.io/zh_CN/docs/")
}
```


* 完成图片的爬取

```go
package main

import (
    "fmt"
    "io/ioutil"
    "net/http"
    "os"
    "strings"
    "sync"

    "github.com/gocolly/colly"
)
```

```go
var locker sync.WaitGroup

func main() {

    col := colly.NewCollector()

    // 检测请求
    col.OnRequest(func(req *colly.Request) {
        fmt.Println("检测一个请求......")
    })

    // 检测响应
    col.OnResponse(func(r *colly.Response) {
        fmt.Println("检测一个响应......")
    })

    // 定位img标签。注册该函数，框架内部回调
    col.OnHTML("img", func(elem *colly.HTMLElement) {

        fmt.Println("ONXHTML")

        // 获取标签对应属性的值。
        // 其他对标签的操作，可以查看对应的API
        http := elem.Attr("src")

        if strings.HasSuffix(http, ".gif") {

            locker.Add(1)

            http := "http:" + http

            go DownLoad(http)
        }
    })

    col.Visit("http://games.sina.com.cn/t/n/2021-01-15/kftpnnx7445951.shtml")

    locker.Wait()
}

func DownLoad(src string) {

    fmt.Println("================================", src)

    // 取一个文件名
    filename := string(src[len(src)-8 : len(src)])
    fmt.Println(filename)

    response, _ := http.Get(src)
    picdata, _ := ioutil.ReadAll(response.Body)

    image, _ := os.Create("./files/" + filename)
    image.Write(picdata)

    defer func() {
        image.Close()
```

```
        locker.Done()
    }()
}
```

```
PS F:\tools\golang\spiders> go run .\colly.go
检测一个请求......
检测一个响应......
ONXHTML
ONXHTML
ONXHTML
============================== http://f.sinaimg.cn/spider2021115/750/w480h270/20210115/36f4-khstaxs2169075.gif
9075.gif
============================== http://n.sinaimg.cn/spider2021115/750/w480h270/20210115/0d87-khstaxs2169127.gif
9127.gif
ONXHTML
ONXHTML
============================== http://n.sinaimg.cn/spider2021115/750/w480h270/20210115/23e5-khstaxs2169131.gif
9131.gif
============================== http://n.sinaimg.cn/spider2021115/750/w480h270/20210115/4aab-khstaxs2658651.gif
8651.gif
```



```
// HTMLElement is the representation of a HTML tag.
type HTMLElement struct {
    // Name is the name of the tag
    Name       string
    Text       string
    attributes []html.Attribute
    // Request is the request object of the element's HTML document
    Request *Request
    // Response is the Response object of the element's HTML document
    Response *Response
    // DOM is the goquery parsed DOM object of the page. DOM is relative
    // to the current HTMLElement
    DOM *goquery.Selection
    // Index stores the position of the current element within all the elements matched by an OnHTML callback
    Index int
}
```