## 爬虫开发

### 引入

在具体讲解爬虫知识之前，我们首先简单回顾下前面Http编程。一种是我们自己编写一个服务器，让浏览器请求我们的服务器。接受请求后分析协议，回发相应数据给浏览器。

另一种，我们可以编写程序充当客户端或者说是用程序模拟浏览器行为，向互联网上现有的服务器发送请求，获取服务器上的数据。例如请求百度，网易，腾讯 等。

实现手段，主要使用的还是net/http这个包。它不仅可以接收浏览器发送过来的请求，实现服务器的功能，也可以模拟浏览器向其它的服务器发送请求。基本的流程如下：

1. 构建、发送请求链接
2. 获取服务器返回的响应数据
3. 过滤、保存、使用得到的数据
4. 关闭请求链接。

打印出完整的网页内容，和浏览器获取的内容是一样的。只不过我们写的 .go程序是直接将服务器返回的所有数据内容打印出来，而浏览器是将服务器返回的内容（代码）按照既定的方式加以执行并显示给用户看，所以我们在浏览器上会看到文字，图片等信息。

### 爬虫简介

**其实我们编写的这个模拟浏览器行为的客户端程序，就是一个爬虫。**可以获取网络服务器数据到本地。只不过，我们简单粗暴的直接将服务器回发的所有数据获取下来，并没有做任何筛选和处理。

**爬虫的定义：**网络爬虫（又被称为网页蜘蛛，网络机器人），是一种按照一定的规则自动地抓取万维网信息的程序。

简单来说，就是编写程序，模拟浏览器发送请求，获取到和浏览器一模一样的数据。因此，我们能获取的是浏览器能够接收到的数据。

**爬虫获取的数据的用途：**

- 呈现数据，呈现在app或者网站上

- 进行数据分析，获得结论

**爬虫的分类：**

- 通用爬虫：搜索引擎的爬虫

- 聚焦爬虫：针对特定网站的爬虫

**聚焦爬虫的工作流程：**

1. 明**确**URL (请求的地址，明确爬什么)
2. **发**送请求，获取响应数据
3. 保存响应数据，提**取**有用信息
4. 处理数据（存储、使**用**）

**爬虫爬取哪些数据：**

- 资讯公司：特定领域的新闻数据的爬虫

- 金融公司：关于各个公司的动态的信息，

- 酒店/旅游：携程，去哪儿的酒店价格信息/机票，景点价格，其他旅游公司价格信息

- 房地产、高铁：10大房地产楼盘门户网站，政府动态等

- 强生保健医药：医疗数据，价格，目前的市场的行情

对爬虫整体做了简单了解后，我们实现一个爬虫案例，尝试爬取百度贴吧中一些讯息。

### 爬取百度贴吧

在爬取之前，再回顾一下爬虫的步骤：

第一：请求的URL地址，也就是明确目标 (要知道你准备在哪个范围或者网站去搜索)，这里我们以“吃鸡”游戏这个贴吧为例，分析地址规律如下：

<http://tieba.baidu.com/f?kw=%E7%BB%9D%E5%9C%B0%E6%B1%82%E7%94%9F&ie=utf-8&pn=0> //第一页

http://tieba.baidu.com/f?kw=%E7%BB%9D%E5%9C%B0%E6%B1%82%E7%94%9F&ie=utf-8&pn=50 //第二页

http://tieba.baidu.com/f?kw=%E7%BB%9D%E5%9C%B0%E6%B1%82%E7%94%9F&ie=utf-8&pn=100 //第三页

总结规律：“下一页”地址是“前一页”地址 + 50

第二：发送请求，获取响应 (将所有的网站的内容全部爬下来)

第三：提取数据，去掉对我们没用处的数据

第四：处理数据（按照我们想要的方式存储和使用）

示例代码：

**package** main  
  
**import** (  
 **"fmt"  
 "strconv"  
 "net/http"  
 "os"**)  
  
*// 实现 读取一个网页内容函数***func** HttpGet(url string) (result string, err error) {  
  
 resp, err1 := http.Get(url) *// 借助 http包的 Get()函数 获取网页数据* **if** err1 != nil {  
 err = err1 *// 将错误传出* **return** }  
 **defer** resp.Body.Close() *// 读取结束，关闭resp.Body* buf := make([]byte, 4096)  
 **for** { *// 读取Body内容* n, err := resp.Body.Read(buf)  
 **if** n == 0 {  
 fmt.Println(**"读完！err:"**, err)  
 **break** }  
 result += string(buf[:n]) *// 拼接每次buf中读到的数据，到result中，返回* }  
 **return**}  
  
**func** working(start, end int) {  
 fmt.Printf(**"正在爬取 %d 到 %d 页\n"**, start, end) *// 测试  
 // 明确目标：url* **for** i:=start; i<=end; i++ {  
 url := **"https://tieba.baidu.com/f?kw=%E7%BB%9D%E5%9C%B0%E6%B1%82%E7%94%9F&ie=utf-8&pn="** +  
 strconv.Itoa((i-1) \* 50)  
 fmt.Printf(**"正在爬：%d页，%s\n"**, i, url)  
  
 result, err := HttpGet(url) *// 封装函数，读取一页内容，存至 result* **if** err != nil {  
 fmt.Println(**"HttpGet err:"**, err)  
 **continue** }  
  
 fileName := strconv.Itoa(i) + **".html"** *// 将读到的一个网页内容，写出成一个文件。用i.html命名文件* f, err := os.Create(fileName) *// 每个网页保存成一个文件* **if** err != nil {  
 fmt.Println(**"Create err:"**, err)  
 **continue** }  
 f.WriteString(result)  
 f.Close() *// 写完一个文件，关闭一个文件。* }  
}  
  
**func** main() {  
 *// 指定爬取的起始、终止页面* **var** start, end int  
 fmt.Printf(**"请输入爬取的起始页( >= 1 )："**)  
 fmt.Scan(&start)  
 fmt.Printf(**"请输入爬取的终止页( >= 起始页)："**)  
 fmt.Scan(&end)  
  
 *// 封装函数，专门完成爬取工作。* working(start, end)  
}

### 并发版网络爬虫

上面实现的案例中，只有一个主协程在爬取网页内容。爬完第一页，再去爬取第二页，再去爬取第三页……这样效率显然很低。学习并发时，我们了解到go语言的goroutine十分轻量级，且能很好的实现并发目的。那么要爬取N页数据，我们可以直接定义N个goroutine分别去爬取，大大提高程序的并发性，执行效率也会高出很多。

将根据URL爬取网页内容、保存生成HTML文件的相关功能封装到函数中。如：

**func** SpiderPage(idx int) {  
 url := **"https://tieba.baidu.com/f?kw=%E7%BB%9D%E5%9C%B0%E6%B1%82%E7%94%9F&ie=utf-8&pn="** +  
 strconv.Itoa((idx-1) \* 50)  
 fmt.Printf(**"正在爬取：%d页，%s\n"**, idx, url)  
  
 result, err := HttpGet2(url) *// 封装函数，读取一页内容，存至 result* **if** err != nil {  
 fmt.Println(**"HttpGet err:"**, err)  
 **return** }  
  
 fileName := strconv.Itoa(idx) + **".html"** *// 将读到的一个网页的内容，写出成一个文件。用i.html命名文件* f, err := os.Create(fileName) *// 每个网页保存成一个文件* **if** err != nil {  
 fmt.Println(**"Create err:"**, err)  
 **return** }  
 f.WriteString(result)  
 f.Close() *// 写完一个文件，关闭一个文件。*}

working()函数只需循环启动goroutine，调用该函数即可。

**func** working(start, end int) { *// 明确目标：url* **for** i:=start; i<=end; i++ {  
 **go** SpiderPage(i) *// 起 go 程并发处理* }  
}

但，这样处理有一个问题。主协程很快创建N个goroutine，working()函数调用完毕退出了，而此时子goroutine还没有爬取完网页内容保存成html文件。这里需要主goroutine等待所有子协程调用完成再退出。

可以借助channel 来达到这一目的。

定义一个名为page的通道，将通道引用和循环因子i一起传递到了SpiderPage方法中。

**func** working(start, end int) {  
 page := make(**chan** int) *// 使用 channel 防止主协程提前退出。  
 // 明确目标：url* **for** i:=start; i<=end; i++ {  
 **go** SpiderPage(i, page) *// 起 go 程并发处理* }  
  
 **for** i:=start; i<=end; i++ {  
 fmt.Printf(**"爬取%d页面完成！\n"**, <-page)  
 }  
}

同时在SpiderPage函数内爬取网页数据完成后，将i值（代表爬取的第几页）写入page。主协程循环创建N个goroutine之后，要依次读取每一个goroutine借助channel写回的i值。在读取期间，如果page上没有写端写入，主goroutine则会阻塞等待，直到有子协程写入，读取打印第i个页面爬取完毕。

示例代码：

**package** main  
  
**import** (  
 **"fmt"  
 "net/http"  
 "strconv"  
 "os"**)  
  
*// 实现 读取一个网页内容函数***func** HttpGet2(url string) (result string, err error) {  
  
 resp, err1 := http.Get(url) *// 借助 http包的 Get()函数 获取网页数据* **if** err1 != nil {  
 err = err1 *// 将错误传出* **return** }  
 **defer** resp.Body.Close() *// 读取结束，关闭resp.Body* buf := make([]byte, 4096)  
 **for** { *// 读取Body内容* n, err := resp.Body.Read(buf)  
 **if** n == 0 {  
 fmt.Println(**"读完！err:"**, err)  
 **break** }  
 result += string(buf[:n]) *// 拼接每次buf中读到的数据，到result中，返回* }  
 **return**}  
  
**func** SpiderPage(idx int, page **chan**<- int) {  
 url := **"https://tieba.baidu.com/f?kw=%E7%BB%9D%E5%9C%B0%E6%B1%82%E7%94%9F&ie=utf-8&pn="** +  
 strconv.Itoa((idx-1) \* 50)  
 fmt.Printf(**"正在爬取：%d页，%s\n"**, idx, url)  
  
 result, err := HttpGet2(url) *// 封装函数，读取一页内容，存至 result* **if** err != nil {  
 fmt.Println(**"HttpGet err:"**, err)  
 **return** }  
  
 fileName := strconv.Itoa(idx) + **".html"***// 将读到的一个网页的内容，写出成一个文件。用i.html命名文件* f, err := os.Create(fileName) *// 每个网页保存成一个文件* **if** err != nil {  
 fmt.Println(**"Create err:"**, err)  
 **return** }  
 f.WriteString(result)  
 f.Close() *// 写完一个文件，关闭一个文件。* page <- idx *// 爬取一个页面完成，写入管道。*}  
  
**func** working2(start, end int) {  
 page := make(**chan** int) *// 使用 channel 防止主协程提前退出。  
 // 明确目标：url* **for** i:=start; i<=end; i++ {  
 **go** SpiderPage(i, page) *// 起协程并发处理* }  
  
 **for** i:=start; i<=end; i++ {  
 fmt.Printf(**"爬取%d页面完成！\n"**, <-page)  
 }  
}  
  
**func** main() {  
 *// 指定爬取的起始、终止页面* **var** start, end int  
 fmt.Printf(**"请输入爬取的起始页( >= 1 )："**)  
 fmt.Scan(&start)  
 fmt.Printf(**"请输入爬取的终止页( >= 起始页)："**)  
 fmt.Scan(&end)  
  
 *// 封装函数，专门完成爬取工作。* working2(start, end)  
}

### 正则表达式

简介

在前面的案例中，我们已经完成了将网页的内容进行爬取，并且进行保存。但是，我们并没有对获取的网页数据内容进行筛选，而直接全部保到文件中了。

那如何对爬取到的网页内容进行筛选提取呢？之前学过的string包中的一些字符串操作函数可以完成这类任务，如：搜索(Contains、Index)、替换(Replace)和解析(Split、Join)，但是处理网页数据实现起来相对而言复杂度较高。实际在工作中，对于这类字符串拆分提取操作，我们通常使用正则表达式来实现。通过正则表达式提取网页内容要方便许多。

当然如果strings包提供的函数能解决你的问题，那么就尽量使用它来解决。因为他们足够简单、而且性能和可读性都要比正则好。

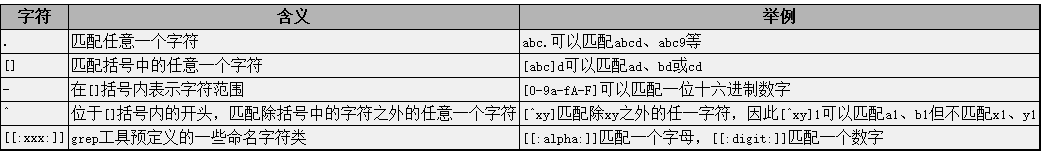
**正则表达式是一种进行模式匹配和文本操纵**的复杂而又强大的工具。虽然正则表达式比纯粹的文本匹配效率低，但是它却更灵活。按照它的语法规则，随需构造出的匹配模式就能够从原始文本中筛选出几乎任何你想要得到的字符组合。

对于初学正则表达式的学者，最困难的地方就是它语法中繁多杂乱的符号。所以大多数工作者，都在记忆中保存正则表达式的整体规范，而做不到完全记忆。确保自己手边有一套可靠的正则查找资料，或者能保证随时上网查询即可。对于正则的语法，这里我们捡常用的一些，加以分类介绍：

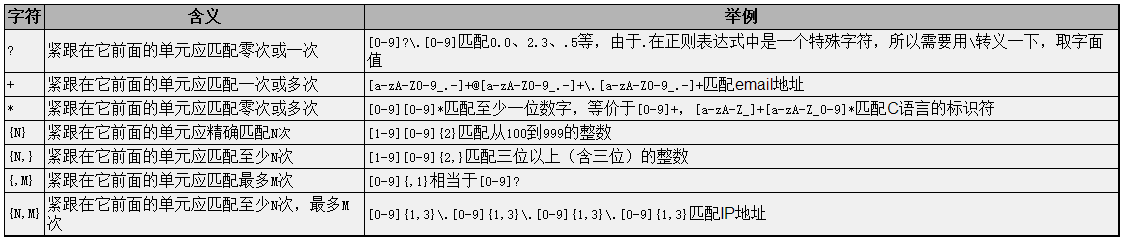
基本语法

可以借助这个在线网站，测试学习正则表达式基础语法： http://www.itlookit.com/regextest.html

**字符类：**



**数量限定符：**



**其它特殊字符：**



**Go语言通过regexp（regular expression）标准包为正则表达式提供了官方支持**，如果你已经使用过其他编程语言提供的正则相关功能，那么你应该对Go语言版本的不会太陌生。但是它们之间也有一些小的差异，因为Go实现的是RE2标准，详细的语法描述可参考：<http://code.google.com/p/re2/wiki/Syntax> 如遇无法打开，也可参看：

<http://www.sun190.com/2015/01/re2-%E6%AD%A3%E5%88%99%E8%A1%A8%E8%BE%BE%E5%BC%8F/>

Go语言使用正则

简单来说，Go语言中使用正则表达式只需要两步即可：

1. **解析、编译正则表达式。**使用 regexp.MustCompile() 函数

**func MustCompile(str string) \*Regexp**

函数的主要作用是将正则表达式中，奇形怪状的符号（如.\*?\[ ...）转换成 Go语言能识别的格式，并将其存成结构体格式，方便编译器识别。

**参数**：正则表达式字串。建议使用反引号。

**返回值**：编译后的结构体。解析失败时会产生panic错误。

1. 根据解析好的规则（结构体形式），**从指定字符串中提取需要的信息。**使用FindAllStringSubmatch() 函数

**func (re \*Regexp) FindAllStringSubmatch(s string, n int) [][]string**

**参数1**：待解析的字符串。

**参数2**：匹配的次数。通常传-1，表示匹配所有。

**返回值**：返回成功匹配的[ ][ ]string。

**说明**：

[

[string1 string2]

[string1 string2]

[string1 string2]

]

其中：string1: 表示带有匹配参考项的全部字串。string2: 表示去除匹配参考项后的字串。

**注意**，要使用前面regexp.MustCompile() 函数调用的返回值，来调用此函数。

下面是使用正则表达式处理字符串检索的几个小例子。

示例代码1：

**package** main  
  
**import** (  
 **"fmt"  
 "regexp"**)  
  
**func** main() {  
 str := **"abc a7c mfc cat 8ca azc cba"** *// 1. 解析、编译正则表达式* ret := regexp.MustCompile(**`a.c`**) *// 可以不用检查出错情况  
 //ret := regexp.MustCompile(`a[0-9]c`)  
 //ret := regexp.MustCompile(`a\dc`)  
  
 // 2. 提取需要信息* alls := ret.FindAllStringSubmatch(str, -1)  
 fmt.Println(alls)  
}

再看一个提取小数的例子：

示例代码2：

**package** main  
  
**import** (  
 **"regexp"  
 "fmt"**)  
  
**func** main() {  
 str := **"3.14 123.123 .68 haha 1.0 abc 7. ab.3 66.6 123."** *// 1. 解析、编译正则表达式  
 //ret := regexp.MustCompile(`\d\.\d`)  
 //ret := regexp.MustCompile(`\d+\.\d`)* ret := regexp.MustCompile(**`\d+\.\d+`**)  
  
 *// 2. 提取需要信息* result := ret.FindAllStringSubmatch(str, -1)  
  
 fmt.Println(result)  
 fmt.Printf(**"------------------------------------\n"**)  
}

提取网页中<div></div>标签数据的例子：

示例代码3：

**import** (  
 **"regexp"  
 "fmt"**)  
  
**func** main() {  
 str := **`   
<!DOCTYPE html>  
<html lang="zh-CN">  
<head>  
 <title>Go语言标准库文档中文版 | Go语言中文网 | Golang中文社区 | Golang中国</title>  
 <meta name="viewport" content="width=device-width, initial-scale=1, maximum-scale=1.0, user-scalable=no">  
 <meta http-equiv="X-UA-Compatible" content="IE=edge, chrome=1">  
 <meta charset="utf-8">  
 <link rel="shortcut icon" href="/static/img/go.ico">  
 <link rel="apple-touch-icon" type="image/png" href="/static/img/logo2.png">  
 <meta name="author" content="polaris <polaris@studygolang.com>">  
 <meta name="keywords" content="中文, 文档, 标准库, Go语言,Golang,Go社区,Go中文社区,Golang中文社区,Go语言社区,Go语言学习,学习Go语言,Go语言学习园地,Golang 中国,Golang中国,Golang China, Go语言论坛, Go语言中文网">  
 <meta name="description" content="Go语言文档中文版，Go语言中文网，中国 Golang 社区，Go语言学习园地，致力于构建完善的 Golang 中文社区，Go语言爱好者的学习家园。分享 Go 语言知识，交流使用经验">  
</head>  
 <title>标题</title>  
 <div>过年来吃鸡啊</div>  
 <div>hello regexp</div>  
 <div>你在吗？</div>  
 <body>呵呵</body>  
  
<frameset cols="15,85">  
 <frame src="/static/pkgdoc/i.html">  
 <frame name="main" src="/static/pkgdoc/main.html" tppabs="main.html" >  
 <noframes>  
 </noframes>  
</frameset>  
</html>  
 `***//反引号``* ret := regexp.MustCompile(**`<div>(.\*)</div>`**)  
  
 result := ret.FindAllStringSubmatch(str, -1)  
  
 fmt.Println(result)  
  
}

运行上述代码，发现可以成功提取<div></div>标签的数据。但输出数据是这样的。

[[<div>过年来吃鸡啊</div> 过年来吃鸡啊] [<div>hello regexp</div> hello regexp] [<div>你在吗？</div> 你在吗？]]

这里再解释下FindAllStringSubmatch()函数的返回值[ ][ ]string：

[

[string1 string2]

[string1 string2]

[string1 string2]

]

其中：string1: 表示带有匹配参考项的全部字串。string2: 表示去除匹配参考项后的字串。测试：

result := ret.FindAllStringSubmatch(str, -1)  
**for** i := 0; i < len(result); i++ {  
 one := result[i]  
 **for** i := 0; i < len(one); i++ {  
 fmt.Printf(**"%d: %s\n"**, i, one[i])  
 }  
}  
fmt.Println(result)

因此，如果我们想得到不包含<div></div>标签的数据内容，可以使用下标[1]来直接获取。

**但是**，如果我们将测试代码适当调整。在原str中添加如下标签内容。就不能提取正常数据了。

**<title>标题</title>  
 <div>过年来吃鸡啊</div>  
 <div>hello regexp</div>  
 <div>你在吗？</div>  
 <div>  
 2块钱啥时候还？  
 过了年再说吧！  
 刚买了车，没钱。。。  
 </div>**

**<body>呵呵</body>**

测试发现，新添加的<div></div>标签中，如果出现换行，则源正则表达式 `<div>(.\*)</div>`不能正确提取数据。因此需要调整正则表达式：`<div>**(?s:(.\*?))**</div>`。分析这个表达式，有两部分内容。

(?s) 是正则表达式的模式修饰符。即Singleline(单行模式)。表示更改.的含义。使它与每一个字符匹配（包括换行 符\n）。

(.\*?) 是一个单元分组。“.”匹配任意字符。“\*?”表重复>=0次匹配。

这个语法，在正则表达式知识里是较难的应用，不必过度学习。我们可以直接记**结论：将(?s:(.\*?))元组放置于某一特征字串中，可以提取带有这一特征字串的内容。**

示例代码4：

**package** main  
  
**import** (  
 **"regexp"  
 "fmt"**)  
  
**func** main() {  
 str := **`   
<!DOCTYPE html>  
<html lang="zh-CN">  
<head>  
 <title>Go语言标准库文档中文版 | Golang中国</title>  
 <meta http-equiv="X-UA-Compatible" content="IE=edge, chrome=1">  
 <meta charset="utf-8">  
 <link rel="shortcut icon" href="/static/img/go.ico">  
</head>  
 <title>标题</title>  
 <div>过年来吃鸡啊</div>  
 <div>hello regexp</div>  
 <div>你在吗？</div>  
 <div>  
 2块钱啥时候还？  
 过了年再说吧！  
 刚买了车，没钱。。。  
 </div>  
 <body>呵呵</body>  
   
<frameset cols="15,85">  
 <noframes>  
 </noframes>  
</frameset>  
</html>  
 `***//反引号``* ret := regexp.MustCompile(**`<div>(?s:(.\*?))</div>`**)  
  
 result := ret.FindAllStringSubmatch(str, -1)  
  
 **for** \_, subStr := **range** result {  
 fmt.Println(subStr[1])  
 }  
}

### 段子爬虫

目标分析

首先打“捧腹网”网站首页，然后单击菜单中“段子”。测试翻页特性是“下一页”+1。

https://www.pengfu.com/xiaohua\_1.html 下一页 +1

<https://www.pengfu.com/xiaohua_2.html>

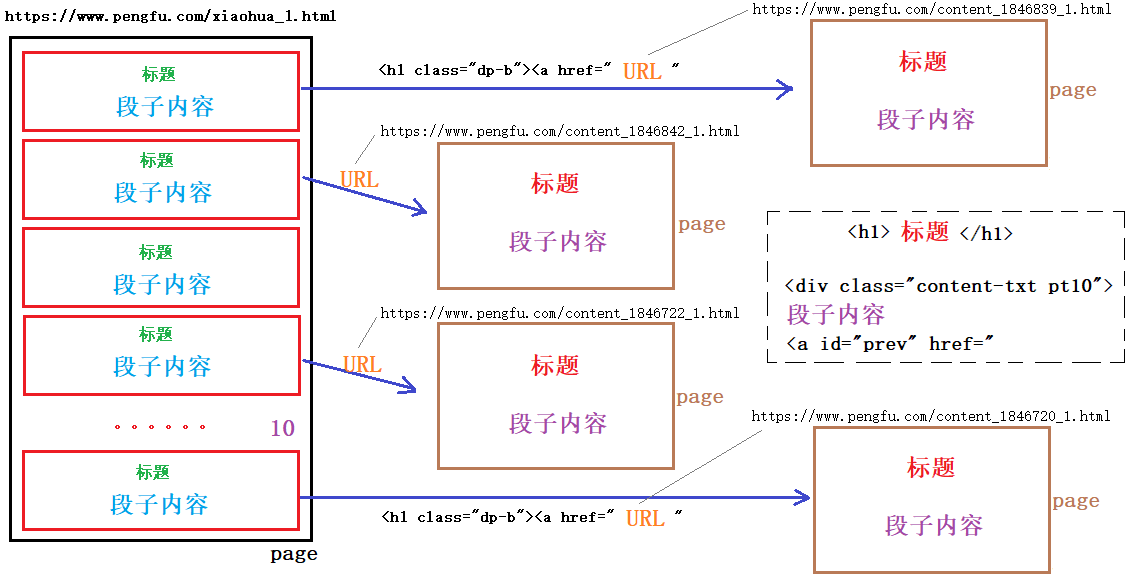
<https://www.pengfu.com/xiaohua_3.html>

使用浏览器自带功能，查看网页源码，每一个网页中共有 10 条段子。每个标题都对应有一个独立的URL链接，该URL以 “<h1 class="dp-b"><a href=”开头，以“**"**”结尾。搜索发现，共有10处。

点击该链接，可以打开一个独立的页面，包含该段子对应的标题及内容。也就是说，主页中的10条段子，可以在10个网页中分别呈现。

查看一个段子网页源码，找寻“标题”和“正文内容”规律。发现：标题被包裹在 <h1> 标题 </h1> 中。而正文内容被包裹在 <div class="content-txt pt10"> 正文内容 <a id="prev" href=" 中。

**整体页面组织结构如下图所示：**



根据上述分析，爬取“捧腹网”段子内容，大致可以分为三步实现。

1. 先获取网页规律，根据用户指定起始、终止页打开要爬取的网页。并获取每一个网页中单个段子所对应页面的URL。
2. 依次将每一个段子对应的网页打开，读取标题和正文内容。
3. 将一个网页内容（10个段子）保存成一个 .txt文件。

爬取段子

按照上述分析，依次编码代码实现。首先提取一个网页中10个段子所对应页面的URL。

**package** main  
  
**import** (  
 **"fmt"  
 "strconv"  
 "net/http"  
 "regexp"**)  
  
**func** HttpGet(url string)(result string, err error) {  
 resp, err1 := http.Get(url)  
 **if** err1 != nil {  
 err = err1  
 **return** }  
 **defer** resp.Body.Close()  
 buf := make([]byte, 4096)  
 **for** {  
 n, \_ := resp.Body.Read(buf)  
 **if** n == 0 {  
 **break** }  
 result += string(buf[:n])  
 }  
 **return**}  
  
**func** SpiderPage(idx int) {  
 *// https://www.pengfu.com/xiaohua\_6.html* url := **"https://www.pengfu.com/xiaohua\_"** + strconv.Itoa(idx) + **".html"** result, err := HttpGet(url)  
 **if** err != nil {  
 fmt.Println(**"HttpGet err:"**, err)  
 **return** }  
 *// fmt.Println(result)  
 // 从 result 中提取各个段子的url, <h1 class="dp-b"><a href=" 一个段子的URL " 使用正则表达式：  
 // 解析编译正则表达式* ret := regexp.MustCompile(**`<h1 class="dp-b"><a href="(?s:(.\*?))"`**)  
 **if** ret == nil {  
 fmt.Println(**"regexp.MustCompile err:"**, err)  
 **return** }  
 *// 2.取需要信息* alls := ret.FindAllStringSubmatch(result, -1)  
  
 *// 提取一个段子的 URL* **for** \_, jokeURL := **range** alls {  
 fmt.Println(**"url="**, jokeURL[1])  
 }  
}  
  
**func** working(start, end int) {  
 fmt.Printf(**"正在爬取 %d 到 %d 页\n"**, start, end) *// 测试* **for** i:=start; i<=end; i++ {  
 SpiderPage(i)  
 }  
}  
  
**func** main() {  
 *// 指定爬取的起始、终止页面* **var** start, end int  
 fmt.Printf(**"请输入爬取的起始页( >= 1 )："**)  
 fmt.Scan(&start)  
 fmt.Printf(**"请输入爬取的终止页( >= 起始页)："**)  
 fmt.Scan(&end)  
  
 *// 封装函数，专门完成爬取工作。* working(start, end)  
}

接下来完成第二步， 封装函数将每一个段子所对应的页面中的标题和正文内容取出。

**func** SpiderJokePage(jokeURL string) (title, content string, err error) {  
 *// 读取段子页面内容* result, err1 := HttpGet(jokeURL)  
 **if** err1 != nil {  
 err = err1  
 **return** }  
 *//解析、编译正则表达式， 处理 title —— <h1> 段子标题 </h1>* ret1 := regexp.MustCompile(**`<h1>(?s:(.\*?))</h1>`**)  
 **if** ret1 == nil {  
 err = fmt.Errorf(**"%s"**, **"MustCompile err"**)  
 **return** }  
 *// 提取 title* tmpTitle := ret1.FindAllStringSubmatch(result, 1) *// 有两处，取第一处* **for** \_, data := **range** tmpTitle {  
 title = data[1] *// 存至返回值 title* title = strings.Replace(title, **"\t"**, **""**, -1)  
 **break** *// 取一个即可。* }  
  
 *//解析、编译正则表达式， 处理 content —— <div class="content-txt pt10"> 段子内容 <a id="prev" href="* ret2 := regexp.MustCompile(**`<div class="content-txt pt10">(?s:(.\*?))<a id="prev" href="`**)  
 **if** ret2 == nil {  
 err = fmt.Errorf(**"%s"**, **"MustCompile err"**)  
 **return** }  
 *// 提取 Content* tmpContent := ret2.FindAllStringSubmatch(result, -1) *// 只有一处* **for** \_, data := **range** tmpContent {  
 content = data[1] *// 存至返回值 content* content = strings.Replace(content, **"\t"**, **""**, -1)  
 **break** *// 提取一个即可。* }  
 **return**}

第三步，封装函数，将每页的10个段子标题及内容保存成一个.txt文件，以页号命名此文件。由于反复打开追加实现较为复杂，首先我们先将读到的所有标题和正文内容保存到 []string 中，然后一次性写入文件。

**func** SaveJoke2File(idx int, fileTitle, fileContent []string) {  
 f, err := os.Create(strconv.Itoa(idx) + **".txt"**)  
 **if** err != nil {  
 fmt.Println(**"Create err:"**, err)  
 **return** }  
 **defer** f.Close()  
  
 n := len(fileTitle)  
 **for** i:=0; i<n; i++ {  
 *// 写入标题* f.WriteString(fileTitle[i] + **"\n"**)  
 *// 写入内容* f.WriteString(fileContent[i] + **"\n"**)  
 *// 写一个华丽分割线* f.WriteString(**"--------------------------------------------------------------\n"**)  
 }  
}

测试，可以完成从指定网页中提取段子标题和正文，存至文件中。但由于需要写出成文件，操作IO，较少网页的保存尚可，当需要保存的网页过多时，运行速度很慢。

与百度贴吧的处理思路相同。我们可以借助 goroutine 和 channel 完成并发，提高爬取速度。

并发实现

很简单，只需要起一个 goroutine 去调用SpiderPage函数即可。但同样需要借助channel，控制主goroutine在创建子协程完成后，不会立即结束。

**func** working(start, end int) {  
 fmt.Printf(**"正在爬取 %d 到 %d 页\n"**, start, end) *// 测试* page := make(**chan** int)  
  
 **for** i:=start; i<=end; i++ {  
 **go** SpiderPage(i, page)  
 }  
  
 **for** i:=start; i<=end; i++ {  
 fmt.Printf(**"第%d个页面爬取完毕\n"**, <-page)  
 }  
}

同时当然也需要在 SpiderPage 函数结尾处，向page通道中写入数据。完成同步。