

爬蟲框架 colly



# 爬蟲框架 colly

網路爬蟲(Web crawler)或叫網路蜘蛛(Spider)，是一套能搜集、解析網路上的訊息、資料，自動獲取該網域網站的摘要、重點甚至內容的機器人。有沒有想過為什麼打開google搜尋引擎，鍵入關鍵字、按一下Enter，就能搜尋相關的資料？因為這就是Google這外星科技公司厲害的地方！因為Google就是個世界上最大之一的網路爬蟲。

簡單地說，爬蟲是擷取網頁的重點內容，擷取並剖析。而Google搜尋引擎就是到各個IP、各個網站擷取、解析資料並存進他們的資料庫，而使用者在下關鍵字時等同是進Google的引擎資料庫搜尋相關資料、跑出搜尋結果，點擊結果時會再導到該網站去。

其實所寫的爬蟲程式並不完全地叫作爬蟲，畢竟個人所寫的程式只能解析特定網站的格式，充其量只能叫做網頁剖析而已，很難做到公司規模等級的自動網路爬蟲。寫爬蟲通常會用一套已經有完整工具的框架，而這裡介紹的colly是golang中爬蟲的主流框架。

# 爬蟲框架 colly

## 安裝 colly

- Colly這個套件 使用到 GoQuery selector來做html parser剖析網頁的語法。以下路徑最末端要加上v2才會指到最新版本(2.x.x)：github.com/gocolly/colly/v2

```
go get -u github.com/gocolly/colly
```

- 在專案底下 放colly1.go程式

```
package main

import (
    "github.com/gocolly/colly"
)

func main() {
}
```

# 爬蟲框架 colly

## OnResponse

- 僅用到以下短短三行程式碼，就能看到文章列表的網頁原始碼：

```
package main

import (
    "fmt"

    "github.com/gocolly/colly"
)

func main() {
    c := colly.NewCollector() // 在colly中使用 Collector 這類物件 來做事情

    c.OnResponse(func(r *colly.Response) { // 當Visit訪問網頁後，網頁響應(Response)時候執行的事情
        fmt.Println(string(r.Body)) // 返回的Response物件r.Body 是[]Byte格式，要再轉成字串
    })

    c.OnRequest(func(r *colly.Request) { // 加入 User-Agent
        r.Headers.Set("User-Agent", "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/86.0.4240.75 Safari/537.36")
    })

    c.Visit("https://tw.yahoo.com/?guccounter=1") // Visit 要放最後
}
```

- 原始碼有了，距離所謂的自製爬蟲 就只差網頁剖析了，也就是要解析在原始碼中出現的標籤html tag。

# 爬蟲框架 colly

## OnHTML

- 搜尋qa-list\_\_title-link這個class就能找到列表中第一頁的文章標題了。
- 透過count這個變數更了解colly.OnHTML的運作。

```
package main

import (
    "fmt"

    "github.com/gocolly/colly"
)

var count = 0

func main() {

    c := colly.NewCollector()

    // 當Visit訪問網頁後，在網頁響應(Response)之後、發現這是HTML格式 執行的事情
    c.OnHTML(".qa-list__title-link", func(e *colly.HTML_Element) { // 每找到一個符合 goquerySelector字樣的結果，便會進這個OnHTML一次
        fmt.Println(e.Text)
        count++
    })

    c.OnRequest(func(r *colly.Request) { // 它幫忙需要寫這一段 User-Agent才給爬
        r.Headers.Set("User-Agent", "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/86.0.4240.75 Safari/537.36")
    })

    c.Visit("https://ithelp.ithome.com.tw/users/20125192/ironman/5938")

    fmt.Println(count) // count值為10，代表原始碼中 有10個符合規則的結果，總共進了OnHTML func 10次
}
```

# 爬蟲框架 colly

## 爬蟲選取器 query Selector

- 在 colly OnHTML 的參數goquerySelector 中可以用條件來篩選所要的HTML內容。可以依照**tag**、**attr**、**class**當作搜尋的條件來選取。會依照以下網頁中的其中這些元素來做範例：

```
<title>...</title>
<meta name="..." content="...">
<h3 class="qa-list__title qa-list__title--ironman">Go繁不及備載<span> 系列</span></h3>
<a href="/users/20125192" id="account" data-account="gjlMOTEa">我的主頁</a>
```

- Tag 標籤
  - 想抓文章標題 title的tag，直接在OnHTML中輸入tag名稱：

```
c.OnHTML("title", func(e *colly.HTMLElement) {
    fmt.Println(e.Text)
})
```

```
<head>
  <style type="text/css">...</style>
  <meta charset="utf-8">
  <meta http-equiv="X-UA-Compatible" content="IE=edge">
  <meta name="viewport" content="width=device-width, initial-scale=1">
  <title>Angular牙起來 :: 2022 iThome 鐵人賽</title>
  <meta name="description" content="Angular學到要牙起來了
不只設定多，要學的東西也很多
文章編排會由入門開始，詳細介紹再慢慢深入
希望讀者在學習時能獲得成就感，能有動力繼續前進

文章...">
  <meta name="keywords" content="it幫幫忙,iThome">
```

# 爬蟲框架 colly

## 爬蟲選取器 query Selector

- Attr 屬性

- 想抓 meta tag 中，有 name 這個屬性的相關訊息：

```
c.OnHTML("meta[name]", func(e *colly.HTMLElement) {  
    fmt.Println(e)  
})
```

- AttrVal 屬性值

- 想抓圖片中的文字，name="description"這串屬性的content：

```
c.OnHTML("meta[name='description']", func(e *colly.HTMLElement) {  
    fmt.Println(e.Attr("content")) // 抓此Tag中的name屬性 來找出此Tag，再印此Tag中的content屬性  
})
```

```
<head>  
  <style type="text/css"></style>  
  <meta charset="utf-8">  
  <meta http-equiv="X-UA-Compatible" content="IE=edge">  
  <meta name="viewport" content="width=device-width, initial-scale=1">  
  <title>Angular 牙起來 :: 2022 iThome 鐵人賽</title>  
  <meta name="description" content="Angular 學到要牙起來了  
不只設定多，要學的東西也很多  
文章編排會由入門開始，詳細介紹再慢慢深入  
希望讀者在學習時能獲得成就感，能有動力繼續前進  
文章...">  
  <meta name="keywords" content="it 邦幫忙, iThome">
```

# 爬蟲框架 colly

## 爬蟲選取器 query Selector

- CSS Class 名稱
- 想以 CSS來抓該 class底下的字：

```
c.OnHTML(".qa-list__title--ironman", func(e *colly.HTMLElement) {  
    fmt.Println(e.Text)  
})
```

```
▶ <div class="ir-profile-years clearfix">...</div>  
▼ <div class="ir-profile-series ">  
    <a href="https://ithelp.ithome.com.tw/2020-12th-ironman/software-dev" class=  
        "group__badge group__badge--software-dev ir-profile-list__badge">  
        Software Development  
    </a>  
▼ <h3 class="qa-list__title qa-list__title--ironman">  
    "  
    Go 繁不及備載" == $0  
    <span> 系列</span>  
    </h3>  
▶ <p class="qa-list__description">...</p>  
▶ <div class="text-right">...</div>
```



# 爬蟲框架 colly

## 爬蟲選取器 query Selector

- CSS ID 唯一識別
- 想以 CSS來抓該 id底下的字：

```
c.OnHTML("#read_more", func(e *colly.HTMLElement) {  
    fmt.Println(e.Text)  
})
```

```
▶ <h3 class="qa-list__title qa-list__title--ironman">...</h3>  
▶ <p class="qa-list__description">...</p>  
▼ <div class="text-right">  
  ▼ <button id="read_more" class="btn btn-qa-list-open"> == $0  
    "展開"  
    "
```

# 爬蟲框架 colly

colly函式作用順序(Call order of callbacks)

1. OnRequest：在發起請求之前，可以預先對Header的參數進行設定
2. OnError：如果在請求的時候發生錯誤
3. OnResponseHeaders：收到回應標頭的時候
4. OnResponse：收到回應回復(內容)的時候
5. OnHTML：收到的響應是HTML格式時(時間點比 OnResponse還晚)，進行goquerySelector篩選
6. OnXML：收到的響應是XML格式時(時間點比 OnHTML還晚)，進行xpathQuery篩選
7. OnScraped：抓取網頁(與OnResponse相仿)，但在最後才進行使用

```
package main

import (
    "fmt"
    "github.com/gocolly/colly/v2"
)

func main() {
    var url = "https://member.ithome.com.tw/login"
    c := colly.NewCollector()

    c.OnRequest(func(r *colly.Request) {
        fmt.Println("1")
    })

    c.OnError(func(_ *colly.Response, err error) {
        fmt.Println("2")
    })

    c.OnResponseHeaders(func(r *colly.Response) {
        fmt.Println("3")
    })

    c.OnResponse(func(r *colly.Response) {
        fmt.Println("4")
    })

    c.OnHTML("body", func(e *colly.HTMLElement) {
        fmt.Println("5")
    })

    c.OnXML("//footer", func(e *colly.XMLElement) {
        fmt.Println("6")
    })

    c.OnScraped(func(r *colly.Response) {
        fmt.Println("7")
    })

    c.Visit(url)
}
```

# 爬蟲框架 colly

colly實作小專案範例：爬文章

- 先打開開發者工具找到文章的標題：

The screenshot displays a web browser window with the DevTools element inspector open. The browser shows a forum page with a blue header and a sidebar. The main content area lists articles, with the first one titled "# Day01 Angular牙起來 - 開場簡介". The DevTools element inspector is open on the right, showing the DOM tree. The selected element is a link with the class "qa-list\_title-link" and the href "https://ithelp.ithome.com/articles/10292161". The link text is "Angular牙起來系列".

```
DevTools is now available in Chinese! Always match Chrome's language Switch DevTools to Chinese Don't show again
```

Select an element in the page to inspect it - Ctrl + Shift + C

```
<div class="container index-top">
  ::before
  <div class="row">
    ::before
    <div class="col-md-12">
      <div class="board">
        <div class="board leftside profile-main">
          <h2 class="ir-profile-header">
            <div class="ir-profile-content">
              <div class="ir-profile-years clearfix">
                <div class="ir-profile-series ir-profile-series--done ">
                  <!-- articles -->
                <div class="qa-list profile-list ir-profile-list">
                  <div class="profile-list__condition">
                    <div class="profile-list__content">
                      <div class="ir-qa-list__status">
                        <h3 class="qa-list_title">
                          <a href="https://ithelp.ithome.com/articles/10292161" class="qa-list_title-link">
                            Angular牙起來系列
                          </a>
                        </h3>
                      <p class="qa-list__desc">
                        Day01 Angular牙起來 - 開場簡介 歡迎你，勇者 首先要恭喜你開局五星級難度 Vue、React 什麼前端框架不遍，偏偏選擇了Angular 惹所害...
                      </p>
                      <div class="qa-list__info">
                        2022-09-16: 由 gilmotea 分享
                      </div>
                    </div>
                  </div>
                </div>
              </div>
            </div>
          </h2>
        </div>
      </div>
    </div>
  </div>
</div>
```

- 首要任務是找到進到各個文章的連結的tag。

# 爬蟲框架 colly

colly 實作小專案範例：爬文章

- `class=".qa-list__title-link"`，從這個標籤開始下手：

```
package main

import (
    "fmt"

    "github.com/gocolly/colly"
)

func main() {
    c := colly.NewCollector()

    c.OnHTML(".qa-list__title-link", func(e *colly.HTML_Element) {
        fmt.Println(e.Text, e.Attr("href"))
        // e.Text 印出 <a> tag 裡面的文字，也就是文章標題
        // e.Attr("href") 則是找到 <a> tag裡面的 href元素
    })

    c.OnRequest(func(r *colly.Request) { // iT邦幫忙需要寫這一段 User-Agent才給爬
        r.Headers.Set("User-Agent", "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/86.0.4240.75 Safari/537.36")
    })

    c.Visit("https://ithelp.ithome.com.tw/users/20125192/ironman/5938?page=1")
    c.Visit("https://ithelp.ithome.com.tw/users/20125192/ironman/5938?page=2")
    c.Visit("https://ithelp.ithome.com.tw/users/20125192/ironman/5938?page=3")
}
```

```
# Day01 Angular 牙起來 - 開場簡介
https://ithelp.ithome.com.tw/articles/10292161

# Day02 Angular 牙起來 - 安裝環境
https://ithelp.ithome.com.tw/articles/10292909

# Day03 Angular 牙起來 - ng與node_module包的關係
https://ithelp.ithome.com.tw/articles/10293692

# Day04 Angular 牙起來 - CLI入門指令及元件
https://ithelp.ithome.com.tw/articles/10294450
```

# 爬蟲框架 colly

## colly 實作小專案範例：爬文章

- 有了<a>中的網址連結字串，想辦法點進去每個連結裡面：

```
package main

import (
    "fmt"
    "strings"

    "github.com/gocolly/colly"
)

func main() {
    c := colly.NewCollector()

    c.OnHTML(".qa-list__title-link", func(e *colly.HTMLElement) {
        // fmt.Println(e.Text, e.Attr("href"))
        // e.Text 印出 <a> tag 裡面的文字，也就是文章標題
        // e.Attr("href") 則是找到 <a> tag裡面的 href元素

        linksStr := e.Attr("href")
        linksStr = strings.Replace(linksStr, " ", "", -1) // 把空白以""取代掉
        links := strings.Split(linksStr, "\n")           // 以換行符號(\n)做為分隔來切割字串

        for _, link := range links {
            c.OnResponse(func(r *colly.Response) {
                fmt.Println(string(r.Body)) // 印出所有返回的Response物件r.Body
            })

            c.Visit(link)
        }
    })

    c.OnRequest(func(r *colly.Request) { // iT邦幫忙需要寫這一段 User-Agent才給爬
        r.Headers.Set("User-Agent", "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/86.0.4240.75 Safari/537.36")
    })

    c.Visit("https://ithelp.ithome.com.tw/users/20125192/ironman/5938?page=1")
    c.Visit("https://ithelp.ithome.com.tw/users/20125192/ironman/5938?page=2")
    c.Visit("https://ithelp.ithome.com.tw/users/20125192/ironman/5938?page=3")
}
```

但想抓的並不是整個文章的原始碼，  
而是想要精華的內文。

# 爬蟲框架 colly

colly 實作小專案範例：爬文章

- 再稍微修改一下：

```
package main

import (
    "fmt"
    "strings"
    "github.com/gocolly/colly"
)

func main() {
    c := colly.NewCollector()

    c.OnHTML(".qa-list__title-link", func(e *colly.HTMLElement) {
        // fmt.Println(e.Text, e.Attr("href"))
        // e.Text 印出 <a> tag 裡面的文字，也就是文章標題
        // e.Attr("href") 則是找到 <a> tag 裡面的 href 元素

        linksStr := e.Attr("href")
        linksStr = strings.Replace(linksStr, " ", "", -1) // 把空白以 "" 取代掉
        links := strings.Split(linksStr, "\n")           // 以換行符號(\n)做為分隔來切割字串

        for _, link := range links {
            c2 := colly.NewCollector() // 這邊要在迴圈一開始再宣告一個 Collector，才不會與原本的混合到
            c2.OnHTML(".qa-markdown", func(e2 *colly.HTMLElement) {
                fmt.Println(e2.Text) // 印出 qa-markdown class 中的文字（文章的內文）
            })

            c2.OnRequest(func(r *colly.Request) { // 它幫幫忙需要寫這一段 User-Agent 才給爬
                r.Headers.Set("User-Agent", "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/86.0.4240.75 Safari/537.36")
            })

            c2.Visit(link) // 找到 <a> 連結網址後，點進去訪問
        }
    })

    c.OnRequest(func(r *colly.Request) { // 它幫幫忙需要寫這一段 User-Agent 才給爬
        r.Headers.Set("User-Agent", "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/86.0.4240.75 Safari/537.36")
    })

    c.Visit("https://ithelp.ithome.com.tw/users/20125192/ironman/5938?page=1")
    c.Visit("https://ithelp.ithome.com.tw/users/20125192/ironman/5938?page=2")
    c.Visit("https://ithelp.ithome.com.tw/users/20125192/ironman/5938?page=3")
}
```

<https://go.dev/play/p/bAA6xJ2zxCG>

# 爬蟲框架 colly

## colly實作小專案範例：爬文章

- 另外可以像以下這樣再做字數計算，爬下來就能方便知道自己的文章總共佔了多少字數：
- 有些HTML元素在會員登入後才會顯示，像是右上角的使用者名稱。這部分就需要帶值(帳號密碼)做**模擬登入**，會比較麻煩。

```
package main

import (
    "fmt"
    "strings"

    "github.com/gocolly/colly"
)

var wordCount = 0
var chineseCount = 0

func main() {
    c := colly.NewCollector()

    c.OnRequest(func(r *colly.Request) { // iT邦幫忙需要寫這一段 User-Agent才給爬
        r.Headers.Set("User-Agent", "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/86.0.4240.75 Safari/537.36")
    })

    c.OnHTML(".qa-list__title-link", func(e *colly.HTMLElement) {
        // fmt.Println(e.Text, e.Attr("href"))
        // e.Text 印出 <a> tag 裡面的文字，也就是文章標題
        // e.Attr("href") 則是找到 <a> tag裡面的 href元素

        linksStr := e.Attr("href")
        linksStr = strings.Replace(linksStr, " ", "", -1) // 把空白以""取代掉
        links := strings.Split(linksStr, "\n")           // 以換行符號(\n)做為分隔來切割字串

        for _, link := range links {
            c2 := colly.NewCollector() // 這邊要在迴圈一開始再宣告一個 Collector，才不會與原本的混合到
            c2.OnHTML(".qa-markdown", func(e2 *colly.HTMLElement) {
                fmt.Println(e2.Text) // 印出 qa-markdown class中的文字 (文章的內文)

                countWord(e2.Text)
            })
            c2.OnRequest(func(r *colly.Request) { // iT邦幫忙需要寫這一段 User-Agent才給爬
                r.Headers.Set("User-Agent", "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/86.0.4240.75 Safari/537.36")
            })
            c2.Visit(link) // 找到<a>連結網址後，點進去訪問
        }
    })

    c.Visit("https://ithelp.ithome.com.tw/users/20125192/ironman/5938?page=1")
    c.Visit("https://ithelp.ithome.com.tw/users/20125192/ironman/5938?page=2")
    c.Visit("https://ithelp.ithome.com.tw/users/20125192/ironman/5938?page=3")

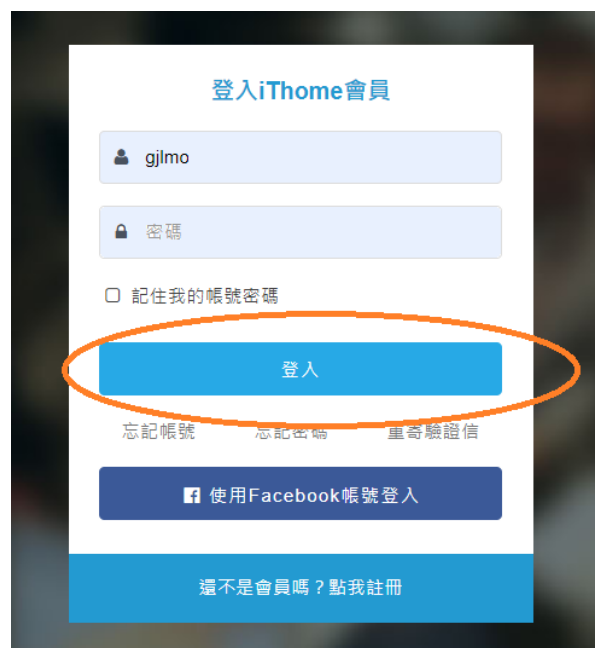
    fmt.Println("英文+中文+數字 共", wordCount, "字")
    fmt.Println("純中文字 共", chineseCount, "字")
}

func countWord(input string) {
    for _, word := range input {
        if word != 32 && word != 10 { // 計算有多少非空白(space)以及換行(\n)的字數
            wordCount++
        }
        if word > 256 { // 計算有多少中文字數(編碼比ASCII大的字)
            chineseCount++
        }
    }
}
```

# 爬蟲框架 colly

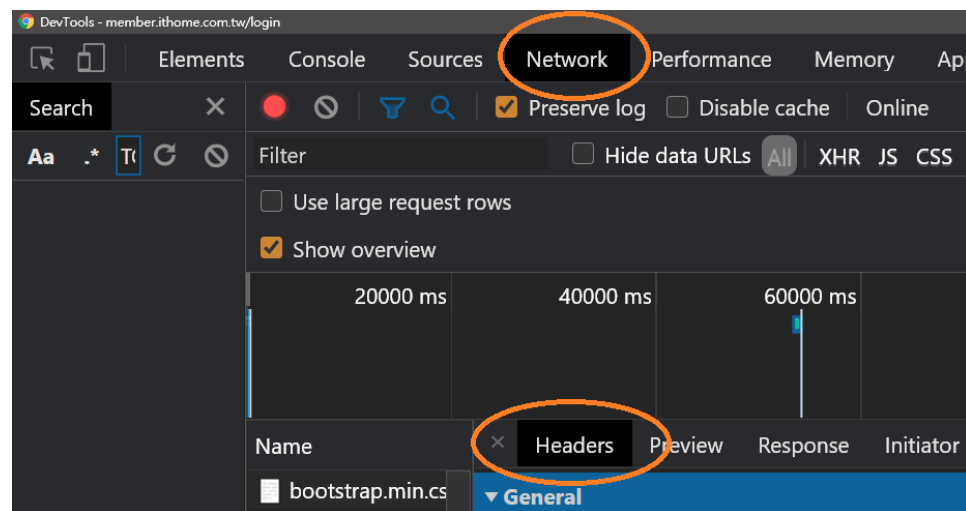
colly模擬使用者登入

- 首先要觀察登入自己的帳號密碼來看看整個流程：



按下登入按鈕。

熟練按下鍵盤上的F12，打開開發者工具，將欄位切換到Network，仔細尋找登入的頁面在哪裡。一邊尋找一邊往下拉，

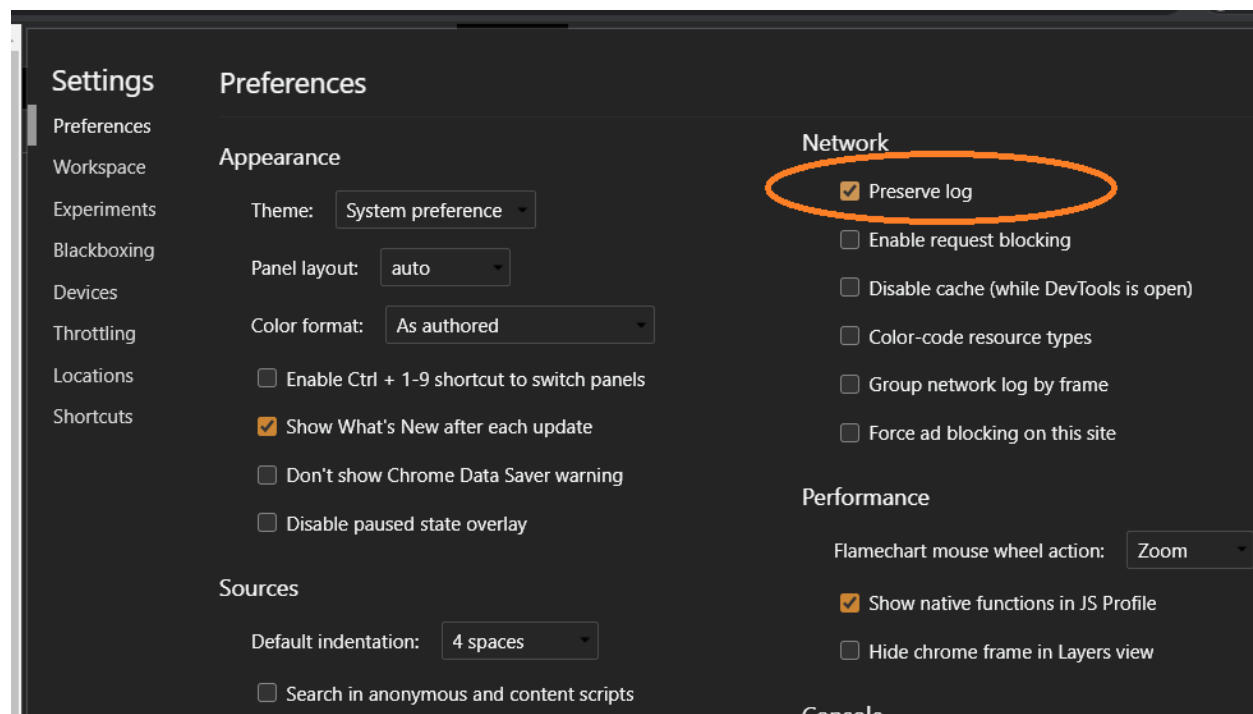
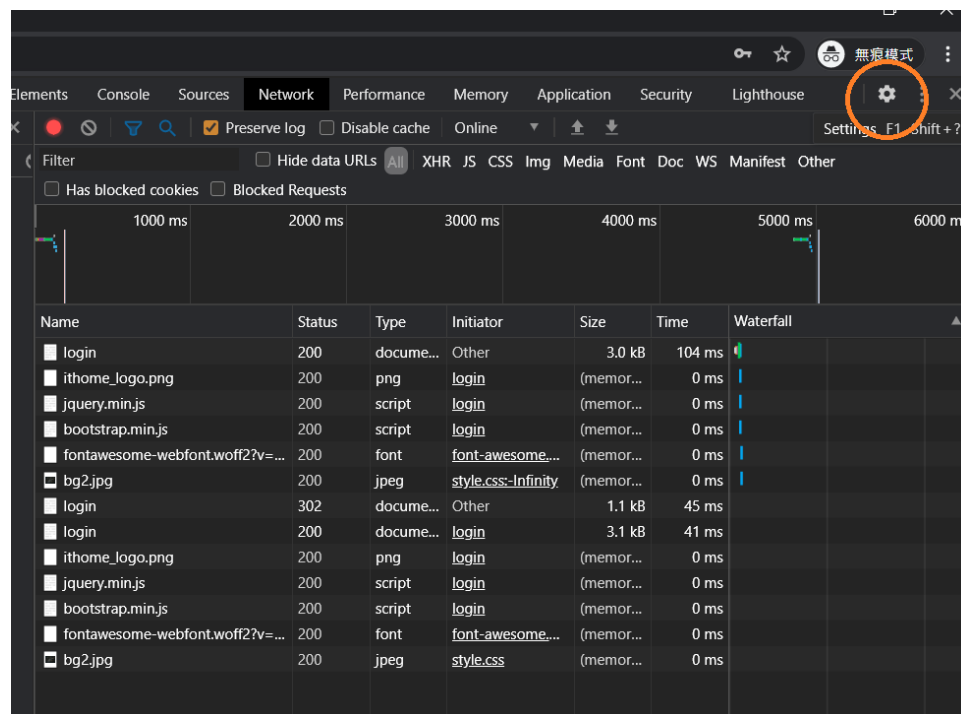




# 爬蟲框架 colly

colly 模擬使用者登入

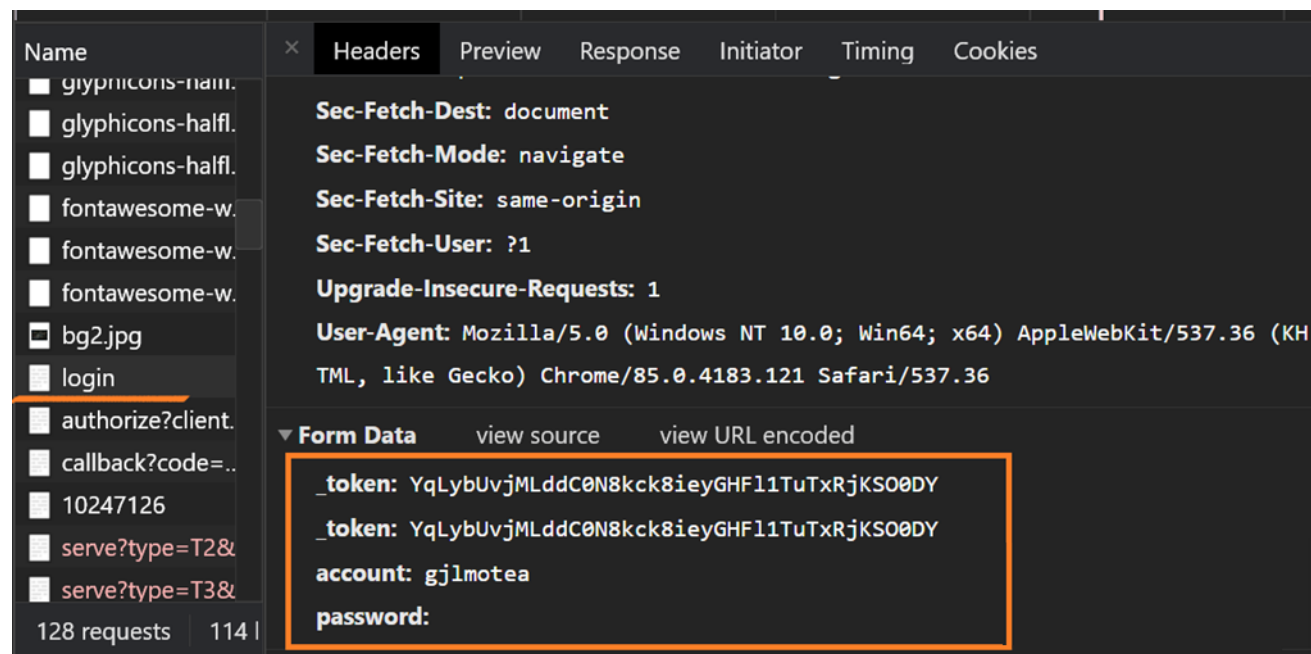
- 可以透過設定 Setting Preserve Log 來保留日誌，就算跳頁、刷新也不會消失：



# 爬蟲框架 colly

colly 模擬使用者登入

- 有個叫作login，點開login頁面：



- 剛剛所輸入的帳號密碼就在眼前了！

# 爬蟲框架 colly

## colly 模擬使用者登入

- 看看登入頁面的原始碼中的登入表單，彼此參照對應一下：\_token、account、password，然後method="POST"

```
<form method="POST" action="..."><input name="_token" type="hidden" value="...">
<input type="hidden" name="_token" value=".....">
<div class="form-group ...">
  <label class="sr-only">帳號</label>
  <input id="account" placeholder="帳號(非email)" name="account">
  <i class="..."></i>
</div>
<div class="...">
  <label class="sr-only">密碼</label>
  <input id="password" placeholder="密碼" name="password" value="">
  <i class="..."></i>
</div>
```

- 也就是說，登入時需要的參數有三個：帳號、密碼、\_Token

# 爬蟲框架 colly

## colly 模擬使用者登入

- Token
  - Token 常被拿來當作訪問令牌(Access token)或是會話連線(Session ID)，通常具有時效性。可以拿來做使用者重複登入時做後踢前或者使用者閒置多久做登出的動作。在有些網站中，登入前會給個Token，在登入的時候代入、驗證Token，代表這一次的連線訪問(Session)。所以必須先取得這個Token：

```
package main

import (
    "fmt"
    "github.com/gocolly/colly/v2"
)

var token string

func main() {
    var url = "https://member.ithome.com.tw/login"
    c := colly.NewCollector()

    // 拿到這次登入的Token
    c.OnHTML("input[name='_token']", func(e *colly.HTMLInputElement) {
        token = e.Attr("value")
    })

    c.OnScraped(func(r *colly.Response) {
        fmt.Println(string(r.Body))
    })

    c.Visit(url)

    fmt.Println(token)
}
```

# 爬蟲框架 colly

## colly 模擬使用者登入

- 帳號密碼
  - 再來要考慮的是，如何在一個Session裡做登入(畢竟下一次訪問網站，勢必會換個連線Session，Token肯定不會是剛剛那個。)
  - 用同一個colly Collector來做POST Payload，傳帳號密碼進去。
  - 可以在傳回的html原始碼中，找到自己的姓名、帳號，那就是成功登入。

```
package main

import (
    "fmt"
    "github.com/gocolly/colly/v2"
    "log"
)

var token string

func main() {
    var url = "https://member.ithome.com.tw/login"
    c := colly.NewCollector()

    // 拿到這次登入的Token
    c.OnHTML("input[name='_token']", func(e *colly.HTMLInputElement) {
        token = e.Attr("value")
    })

    c.Visit(url)

    c.OnRequest(func(r *colly.Request) {
        r.Headers.Set("User-Agent", "Chrome/84.0.4147.89 Safari/537.36")
        r.Headers.Set("Host", "https://member.ithome.com.tw")
        r.Headers.Set("Origin", "https://member.ithome.com.tw")
        r.Headers.Set("Referer", "https://member.ithome.com.tw/login")
        // 這幾行在這it幫忙沒有起到作用，但有些網站會照這些資訊判斷、阻擋其他來源
    })

    c.OnScraped(func(r *colly.Response) {
        fmt.Println(string(r.Body))
    })

    var formData = map[string]string{
        "account": "你的名字",
        "password": "你的30公分",
        "_token": token,
    }
    err := c.Post(url, formData) // 進到該url 執行POST
    if err != nil {
        log.Println(err)
    }
}
```

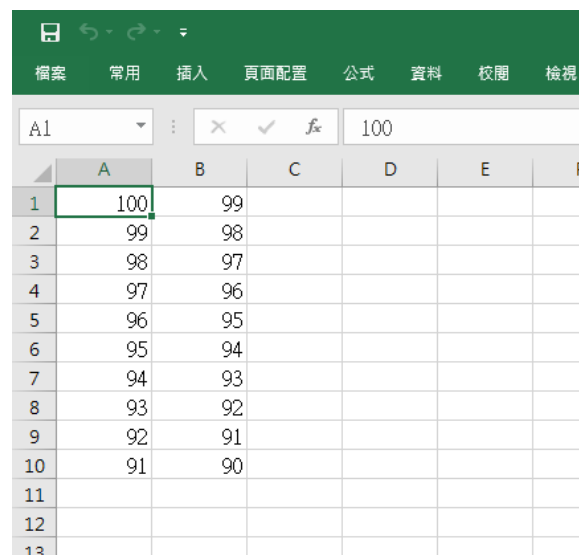
CSV與Google Sheet



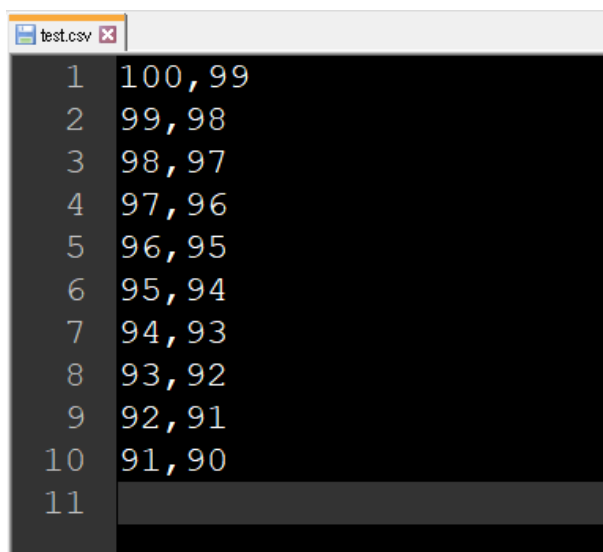
# CSV與Google Sheet

CSV 是以逗號,及換行\n分隔的資料格式，從CSV的英文(Comma-Separated Values)中就能知道。

- 用Excel開啟一個CSV檔案(副檔名要為.csv 而非.xlsx。xlsx是 Microsoft Office Excel 獨有的格式)



|    | A   | B  | C | D | E | F |
|----|-----|----|---|---|---|---|
| 1  | 100 | 99 |   |   |   |   |
| 2  | 99  | 98 |   |   |   |   |
| 3  | 98  | 97 |   |   |   |   |
| 4  | 97  | 96 |   |   |   |   |
| 5  | 96  | 95 |   |   |   |   |
| 6  | 95  | 94 |   |   |   |   |
| 7  | 94  | 93 |   |   |   |   |
| 8  | 93  | 92 |   |   |   |   |
| 9  | 92  | 91 |   |   |   |   |
| 10 | 91  | 90 |   |   |   |   |
| 11 |     |    |   |   |   |   |
| 12 |     |    |   |   |   |   |
| 13 |     |    |   |   |   |   |



```
test.csv
1 100,99
2 99,98
3 98,97
4 97,96
5 96,95
6 95,94
7 94,93
8 93,92
9 92,91
10 91,90
11
```

- 用記事本開啟一個CSV檔案：(也有出現以空白字元、\t或其他字元來做分隔，只要解析時以該字元下去做區隔就行)

# CSV與Google Sheet

## 讀取 Read CSV

- Golang讀檔案時會需要用到絕對路徑，在這邊先取 `pwd` 當前路徑、再加上 `fileName` 檔案名稱：

```
package main

import (
    "encoding/csv"
    "fmt"
    "io"
    "log"
    "os"
    "path/filepath"
)

var Pwd string
var FilePath string
var FileName = "test.csv"

func init() {
    Pwd, _ = os.Getwd()
    FilePath = filepath.Join(Pwd, FileName)
}

func main() {
    file, err := os.OpenFile(FilePath, os.O_RDONLY, 0777) // os.O_RDONLY 表示只讀、0777 表示(owner/group/other)權限
    if err != nil {
        log.Fatalf("找不到CSV檔案路徑:", FilePath, err)
    }

    // read
    r := csv.NewReader(file)
    r.Comma = ',' // 以何種字元作分隔，預設為','。所以這裡可拿掉這行
    for {
        record, err := r.Read()
        if err == io.EOF {
            break
        }
        if err != nil {
            log.Fatalf(err)
        }

        var a = record[0]
        var b = record[1]

        fmt.Println(a, b)
    }
}
```

讀取結果：

```
<4 go setup calls>
100 99
99 98
98 97
97 96
96 95
95 94
94 93
93 92
92 91
91 90
```

讀寫權限的常數：

```
os.O_RDONLY 唯讀
os.O_WRONLY 唯寫
os.O_RDWR   讀/寫
```



# CSV與Google Sheet

## 寫入 Write CSV

- Write寫入單行：

```
func Write() {  
    file, err := os.OpenFile(filePath, os.O_WRONLY, 0777)  
    if err != nil {  
        log.Fatalln("找不到CSV檔案路徑:", filePath, err)  
    }  
  
    w := csv.NewWriter(file)  
    x := []string{"999"}  
    w.Write(x)  
  
    w.Flush() // 把在buffer緩存中的所有資料輸出  
}
```

- WriteAll寫入多行：

```
func Write() {  
    file, err := os.OpenFile(filePath, os.O_WRONLY, 0777)  
    if err != nil {  
        log.Fatalln("找不到CSV檔案路徑:", filePath, err)  
    }  
  
    // w := csv.NewWriter(file)  
    // x := []string{"999"}  
    // w.Write(x)  
  
    // w.Flush() // 把在buffer緩存中的所有資料輸出  
  
    w := csv.NewWriter(file)  
    x := [][]string{{"999", "1"}, {"998", "997"}}  
    w.WriteAll(x)  
  
    w.Flush() // 把在buffer緩存中的所有資料輸出  
}
```

# CSV與Google Sheet

## 寫入 Write CSV

- 若開啟一個空白檔案寫入，不會有任何問題。但若開啟已經有資料的CSV，再寫入的話會發生問題：

已有的原始資料：

|   | A  | B  | C  | D  |
|---|----|----|----|----|
| 1 | 1  | 2  | 3  | 4  |
| 2 | 5  | 6  | 7  | 8  |
| 3 | 9  | 10 | 11 | 12 |
| 4 | 13 | 14 | 15 | 16 |

如果寫入以下一行資料：

```
w := csv.NewWriter(file)
x := []string{"999"}
w.Write(x)

w.Flush() // 把在buffer緩存中的所有資料輸出
```

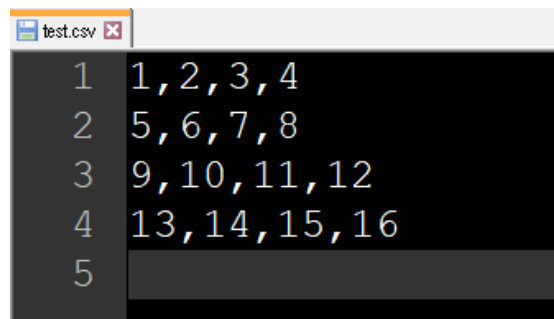
|   | A   | B  | C  | D  |
|---|-----|----|----|----|
| 1 | 999 |    |    |    |
| 2 | 3   | 4  |    |    |
| 3 | 5   | 6  | 7  | 8  |
| 4 | 9   | 10 | 11 | 12 |
| 5 | 13  | 14 | 15 | 16 |

# CSV與Google Sheet

## 寫入 Write CSV

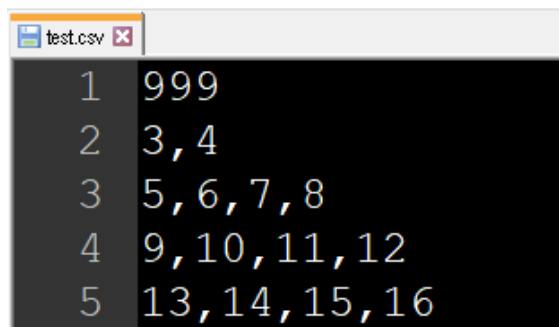
- 多了一堆莫名其妙的空格？用EXCEL開，會看不清發生了什麼事情，但以記事本開啟的話：

資料從原本的狀態：



|   |             |
|---|-------------|
| 1 | 1,2,3,4     |
| 2 | 5,6,7,8     |
| 3 | 9,10,11,12  |
| 4 | 13,14,15,16 |
| 5 |             |

變成這樣：



|   |             |
|---|-------------|
| 1 | 999         |
| 2 | 3,4         |
| 3 | 5,6,7,8     |
| 4 | 9,10,11,12  |
| 5 | 13,14,15,16 |

- 也就是說輸入的單行的資料是999'\n'，把原始資料1,2,3,4'\n'給蓋過去，將最前面四個字元覆蓋掉了，就變成999'\n'3,4'\n' ...，所以用Excel打開才會被解析成右上方那樣的圖。
- 附加 Append CSV**：打開檔案的時候以 O\_APPEND 模式開啟，就能不影響舊資料，將欲寫入的資料Append在最後了。

```
file, err := os.OpenFile(filePath, os.O_APPEND, 0777)
```

# CSV與Google Sheet

## Google Sheet 試算表應用 (Spreadsheet)

- Enable the Google Sheets API

### Step 1: Turn on the Google Sheets API

Click this button to create a new Cloud Platform project and automatically enable the Google Sheets API:

Enable the Google Sheets API

In resulting dialog click **DOWNLOAD CLIENT CONFIGURATION** and save the file `credentials.json` to your working directory.

# CSV與Google Sheet

## Google Sheet 試算表應用 (Spreadsheet)

- 下載套件

```
$ go get -u google.golang.org/api/sheets/v4  
$ go get -u golang.org/x/oauth2/google
```

### Step 2: Prepare the workspace

- Set the `GOPATH` environment variable to your working directory.
- Get the Google Sheets API Go client library and OAuth2 package using the following commands:

```
$ go get -u google.golang.org/api/sheets/v4  
$ go get -u golang.org/x/oauth2/google
```



# CSV與Google Sheet

## Google Sheet 試算表應用 (Spreadsheet)

- 執行範例程式
- 執行完畢會出現如下的網址：

```
<4 go setup calls>  
open token.json: The system cannot find the file specified.  
Go to the following link in your browser then type the authorization code:  
https://accounts.google.com/o/oauth2/auth?access\_type=offline&client\_id=385269114445-b1gsbh0do
```

- 點擊進去，點選進階、進入、授權：



得到這一組授權碼：

localhost/?state=state-token&code=4/0AfJohXmx5VymPGwfxlC0yO4ybdKot7\_Zhl-XARj5SjWGWZ6bdJ3JsYJl3H94aGsqnsslh3g&scope=https://www.googleapis.com/auth/spreadsheets.readonly

# CSV與Google Sheet

## Google Sheet 試算表應用 (Spreadsheet)

- 執行範例程式
- 最後將授權碼貼回，讓程式產生token：

```
>4 go setup calls>  
open token.json: The system cannot find the file specified.  
Go to the following link in your browser then type the authorization code:  
https://accounts.google.com/o/oauth2/auth?access\_type=offline&client\_id=385269114445-b1gsbh0do4/1AY0e-g7R...y7e-...
```

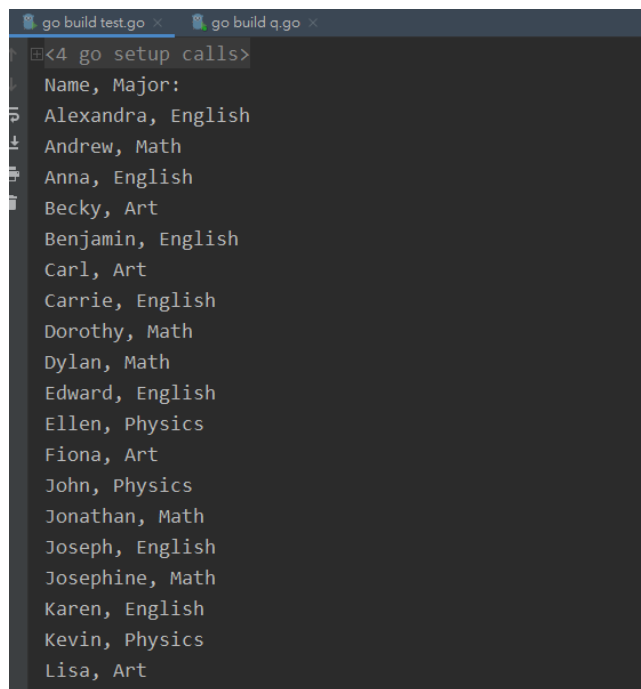
- 專案底下自動產出的 token，裡面有 access\_token：

|            |                   |                     |      |
|------------|-------------------|---------------------|------|
| test.csv   | 2021/1/13 上午 1... | Microsoft Excel ... | 1 KB |
| test.go    | 2021/1/18 下午 0... | JetBrains GoLand    | 4 KB |
| token.json | 2021/1/18 上午 1... | JSON File           | 1 KB |

# CSV與Google Sheet

## Google Sheet 試算表應用 (Spreadsheet)

- 執行範例程式
- 再執行一次程式，會印出如下的結果：



```
go build test.go x go build q.go x
<4 go setup calls>
Name, Major:
Alexandra, English
Andrew, Math
Anna, English
Becky, Art
Benjamin, English
Carl, Art
Carrie, English
Dorothy, Math
Dylan, Math
Edward, English
Ellen, Physics
Fiona, Art
John, Physics
Jonathan, Math
Joseph, English
Josephine, Math
Karen, English
Kevin, Physics
Lisa, Art
```

這等於是用同意的授權(Read權限)去開啟這一份表單，  
讀取範圍並且把其中兩個欄位的值印出來。



# CSV與Google Sheet

## Google Sheet 試算表應用 (Spreadsheet)

- 範例程式(程式碼的部分：對Value進行操作)
  - 往下看之前，先大略程式碼的流程，前幾個function都是做與Config、Token相關的事情，只有在main後半部，將試算表內的數值讀出時，才使用到 sheets API 的功能。
  - 透過option.WithHTTPClient 以及 sheets.NewService來達成。

```
srv, err := sheets.NewService(ctx, option.WithHTTPClient(client))
```

# CSV與Google Sheet

## Google Sheet 試算表應用 (Spreadsheet)

- 範例程式(程式碼的部分：對Value進行操作)
  - 變數名稱
    - spreadsheetId：用來指定哪一份表單的ID。是google sheet網址URI中的一部分
    - readRange：欲讀取哪個表的欄位範圍(Range)。Class Data!A2:E 驚嘆號!前是表(Table)的名稱
  - 讀寫權限
    - 在google.ConfigFromJSON中的<https://www.googleapis.com/auth/spreadsheets.readonly>，把readonly拿掉就有讀/寫權限  
只不過需要刪除Token再產生一次。詳細值參照[文件](#)。

# CSV與Google Sheet

## Google Sheet 試算表應用 (Spreadsheet)

- 範例程式(程式碼的部分：對Value進行操作)
  - 讀取表單欄位(Get)
  - 取得一個範圍內的值 Get：

```
resp, err := srv.Spreadsheets.Values.Get(SpreadsheetId, readRange).Do()
```

- 取得多個範圍的值 BatchGet：

```
resp, err := srv.Spreadsheets.Values.BatchGet(SpreadsheetId).Ranges(readRange1, readRange2).Do()
```

# CSV與Google Sheet

## Google Sheet 試算表應用 (Spreadsheet)

- 範例程式(程式碼的部分：對Value進行操作)
  - 更新表單欄位(Update)
  - 把剛剛讀取到的欄位數值套用寫回：

```
res, err := srv.Spreadsheets.Values.Update(SpreadsheetId, readRange, resp).ValueInputOption("RAW").Do()
```

- 其中的 RAW 可以換成 USER\_ENTERED 詳見[文件](#)。

# CSV與Google Sheet

## Google Sheet 試算表應用 (Spreadsheet)

- 範例程式(程式碼的部分：對Value進行操作)
- 物件 `sheets.ValueRange`

```
vr := sheets.ValueRange{
    MajorDimension: "",
    Range:          "",
    Values:          nil,
    ServerResponse: googleapi.ServerResponse{},
    ForceSendFields: nil,
    NullFields:      nil,
}
```

- **MajorDimension**(解析方式的)主要維度：
  - 填入 ROWS 表示以 列 方式讀取(先由左至右 再上到下)，以圖為例值為[[1,2],[3,4]]
  - 填入 COLUMNS 表示以 欄 方式讀取(先由上到下 再左至右)，以圖為例值為[[1,3],[2,4]]

|   | A | B | C |
|---|---|---|---|
| 1 | 1 | 2 |   |
| 2 | 3 | 4 |   |
| 3 |   |   |   |
| 4 |   |   |   |
| 5 |   |   |   |

# CSV與Google Sheet

## Google Sheet 試算表應用 (Spreadsheet)

- 範例程式(程式碼的部分：對Value進行操作)
- 物件 `sheets.ValueRange`

```
vr := sheets.ValueRange{
    MajorDimension: "",
    Range:          "",
    Values:          nil,
    ServerResponse: googleapi.ServerResponse{},
    ForceSendFields: nil,
    NullFields:      nil,
}
```

- **Range(範圍)**：例如 A2:E5
- **Values(值)**：例如 `[[test1 testtest 12] [test2 testtest 4]]`，型別為`[][]interface{}`
- **ServerResponse(伺服器回傳的東西)**：如 `HTTPStatusCode:200`、`Header`、`Cache-Control...`等等

# CSV與Google Sheet

## Google Sheet 試算表應用 (Spreadsheet)

- 範例程式(程式碼的部分：對Value進行操作)
  - 為了將值代入Values中，需要讓 2D slice string 轉成 2D slice interface，這邊提供兩個不一樣的方法：

```
data1D := []string{"1111", "2222"}
s1D := make([]interface{}, len(data1D))
for i, v := range data1D {
    s1D[i] = v
}

s2D := [][]interface{}{}
s2D = append(s2D, s1D)
```

```
data2D := [][]string{{"AAAA", "1222"}, {"CCCC", "9999,999,1234"}}
s2D := make([][]interface{}, len(data2D))
for i, v := range data2D {
    for j, x := range v {
        if j == 0 {
            s2D[i] = make([]interface{}, len(data2D[0]))
        }
        s2D[i][j] = x
    }
}
```

# CSV與Google Sheet

## Google Sheet 試算表應用 (Spreadsheet)

- 範例程式(程式碼的部分：對Value進行操作)
- 建立一個Test資料表來試試看：

|   | A  | B  | C  | D  | E  |
|---|----|----|----|----|----|
| 1 | 1  | 2  | 3  | 4  | 5  |
| 2 | 3  | 4  | 5  | 6  | 7  |
| 3 | 5  | 6  | 7  | 8  | 9  |
| 4 | 7  | 8  | 9  | 10 | 11 |
| 5 | 9  | 10 | 11 | 12 | 13 |
| 6 | 11 | 12 | 13 | 14 | 15 |

Update前

|   | A    | B    | C  | D  | E  |
|---|------|------|----|----|----|
| 1 | 1    | 2    | 3  | 4  | 5  |
| 2 | 1111 | 2222 | 5  | 6  | 7  |
| 3 | 5    | 6    | 7  | 8  | 9  |
| 4 | 7    | 8    | 9  | 10 | 11 |
| 5 | 9    | 10   | 11 | 12 | 13 |
| 6 | 11   | 12   | 13 | 14 | 15 |

Update後

- 再來將 data1D(s2D) 用Update更新回去：

```
vr := sheets.ValueRange{
    MajorDimension: "ROWS",
    Values:         s2D,
}

res, err := Srv.Spreadsheets.Values.Update(SpreadsheetId, "Test!A2:E5", &vr).ValueInputOption("USER_ENTERED").Do()
```



# CSV與Google Sheet

## Google Sheet 試算表應用 (Spreadsheet)

- 範例程式(程式碼的部分：對Value進行操作)
  - 一次修改多個範圍 **BatchUpdate**：

```
var valueRanges = []*sheets.ValueRange{}

for _, ... := range ... {
    ...
    valueRanges = append(valueRanges, &valueRange)
}

var batchUpdateRequest = sheets.BatchUpdateValuesRequest{
    Data:          valueRanges,
    ValueInputOption: "USER_ENTERED",
}

_, err := Srv.Spreadsheets.Values.BatchUpdate(spreadsheetId, &batchUpdateRequest).Do()

if err != nil {
    log.Println(err)
}
```

# CSV與Google Sheet

## Google Sheet 試算表應用 (Spreadsheet)

- 範例程式(程式碼的部分：對Value進行操作)
  - 附加表單欄位(Append)：
    - 代入的欄位、用法與Update大同小異，但Append不會覆寫欄位、而是附加欄位上去，如果帶入的Range範圍內已有值，則會append到最後一行(Row)，否則會填入Range中。

|   | A  | B  | C  | D  | E  |
|---|----|----|----|----|----|
| 1 | 1  | 2  | 3  | 4  | 5  |
| 2 |    |    | 5  | 6  | 7  |
| 3 | 5  | 6  | 7  | 8  | 9  |
| 4 | 7  | 8  | 9  | 10 | 11 |
| 5 | 9  | 10 | 11 | 12 | 13 |
| 6 | 11 | 12 | 13 | 14 | 15 |

Append前

|   | A    | B    | C  | D  | E  |
|---|------|------|----|----|----|
| 1 | 1    | 2    | 3  | 4  | 5  |
| 2 | 1111 | 2222 | 5  | 6  | 7  |
| 3 | 5    | 6    | 7  | 8  | 9  |
| 4 | 7    | 8    | 9  | 10 | 11 |
| 5 | 9    | 10   | 11 | 12 | 13 |
| 6 | 11   | 12   | 13 | 14 | 15 |

Append後

```
vr := sheets.ValueRange{
    MajorDimension: "ROWS",
    Values:         s2D,
}

res, err := Srv.Spreadsheets.Values.Append(SpreadsheetId, "Test!A2:B2", &vr).ValueInputOption("USER_ENTERED").Do()
// Range也可以只帶 "Test!A2"
```

# CSV與Google Sheet

## Google Sheet 試算表應用 (Spreadsheet)

- 範例程式(程式碼的部分：對Value進行操作)
- 刪除表單欄位(Clear)：

```
cr := sheets.ClearValuesRequest{}  
res, err := Srv.Spreadsheets.Values.Clear(SpreadsheetId, "Test!A2:B2", &cr).Do()
```

|   | A  | B  | C  | D  | E  |
|---|----|----|----|----|----|
| 1 | 1  | 2  | 3  | 4  | 5  |
| 2 |    |    | 5  | 6  | 7  |
| 3 | 5  | 6  | 7  | 8  | 9  |
| 4 | 7  | 8  | 9  | 10 | 11 |
| 5 | 9  | 10 | 11 | 12 | 13 |
| 6 | 11 | 12 | 13 | 14 | 15 |

Clear後

# CSV與Google Sheet

## Google Sheet 試算表應用 (Spreadsheet)

- 表單能只讀寫單一個欄位嗎？
  - 可以透過Range限制範圍在單一個欄位內，來達成這件事情。
- 能做到SQL那樣修改特定欄位嗎？
  - 目前不行，只能透過操作變數來修改指定Range範圍。
  - 但查詢功能可以透過表單內建函式 Google Sheet Query 達成SQL語法的查詢操作，只不過沒辦法像SQL那麼便利。

```
=QUERY(C2:C5, "select *")
```

| fx   =QUERY(C2:C5, "select*") |       |          |    |      |      |
|-------------------------------|-------|----------|----|------|------|
|                               | A     | B        | C  | D    | E    |
| 1                             | 帳號    | 密碼       | 積分 | 創帳日期 | 可用狀態 |
| 2                             | test1 | testtest | 12 |      |      |
| 3                             | test2 | testtest | 4  |      |      |
| 4                             |       | 12       |    |      |      |
| 5                             |       | 4        |    |      |      |
| 6                             |       |          |    |      |      |
| 7                             |       |          |    |      |      |

```
=QUERY(C2:C, "select avg(C), max(C)")
```

| =QUERY(C2:C, "select avg(C), max(C)") |          |    |      |      |     |     |
|---------------------------------------|----------|----|------|------|-----|-----|
| A                                     | B        | C  | D    | E    | F   | G   |
| 帳號                                    | 密碼       | 積分 | 創帳日期 | 可用狀態 | avg | max |
| test1                                 | testtest | 12 |      |      | 8   | 12  |
| test2                                 | testtest | 4  |      |      |     |     |
|                                       |          |    |      |      |     |     |
|                                       |          |    |      |      |     |     |