

CH14 使用Go的HTTP客戶端

1-1 前言

- Go程式能互動的資料不只限於本機，更能透過網路和遠端的程式交換資訊
- 更精確地說，你能使用go語言撰寫HTTP客戶端(client)與伺服器(server)，且只要使用標準函式庫就能做到
- HTTP客戶端即用來從網路伺服器接收資料，或傳送資料給伺服器的程式，最有名的例子大概是網頁瀏覽器(如google chrome)

- 當你在瀏覽器輸入網址時，它內建的HTTP客戶端就會向該網址的伺服器請求(request)資料
- 伺服器會蒐集資料(網頁)並回傳給客戶端，後者會將結果顯示在瀏覽器
- 同樣的，當你在瀏覽器填寫表單和按下送出，瀏覽器也會用他的HTTP客戶端將資料傳到伺服器，再將伺服器的回應抓回來
- 在本章，我們要來了解go的HTTP客戶端是什麼，以及他能跟伺服器收發資料的各種方式及案例

- 註：

為了示範，這一章也會看到基本的go伺服器程式，但細節會留至下一章講解

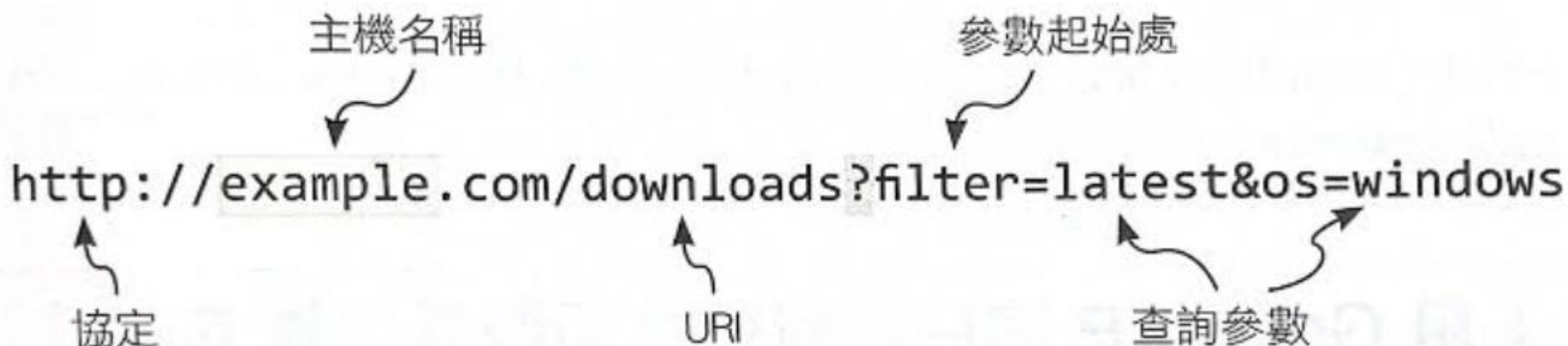
14-2 Go語言的HTTP客戶端

- 若想在go程式中使用HTTP客戶端，可以使用內建的net/http函式庫
- 這有兩種方式：一是直接沿用net/http內建的標準HTTP客戶端，這樣不僅簡單，也能快速打造可用的程式；其次是建立自訂的HTTP客戶端，讓你調整送出請求的行為和其他功能，這樣雖然花時間，但你會有更自由的網路請求控制權

- 使用HTTP客戶端時，你能送出幾種不同的HTTP請求，本章只會討論最主要的兩種請求：**GET** 和 **POST**：
 - 當你在瀏覽器輸入網址，它會對那個位址的伺服器送出**GET**請求，以取得傳回的資料並顯示出來
 - 當你填寫表格和送出(例如登入)時，瀏覽器會用**POST**請求將資料提交給伺服器

14-3 對伺服器傳送GET請求

- GET是HTTP請求中最常見的一種，它使用的URL會描述資源所在的位址，並藉由查詢參數來附帶額外的資訊
- GET請求的URL能拆成幾個部分：



The diagram illustrates the components of the URL `http://example.com/downloads?filter=latest&os=windows`. Hand-drawn arrows point from labels to specific parts of the URL:

- 主機名稱** (Host Name) points to `example.com`.
- 參數起始處** (Parameter Start) points to the `?` character.
- 協定** (Protocol) points to `http`.
- URI** points to `downloads`.
- 查詢參數** (Query Parameter) points to the entire query string `filter=latest&os=windows`.

1. 協定(protocol) : 描述客戶端如何跟伺服器連線，最常見的協定為HTTP(無加密)和HTTPS(有加密)
2. 主機名稱(hostname) : 要連上的伺服器位址
3. URL : 全名為“統一資源識別碼(Uniform Resource Identifier)”，為伺服器資源所在路徑
4. 查詢參數(query parameters) : 要傳給伺服器的額外資訊，可以看到參數與URL是以?分開的，好讓伺服器能解析出參數；而各參數之間則用&分隔

14-3-1 使用http.Get()發送GET請求

- 若想在go語言對一個URL送出GET請求，並接收伺服器的回應，可以使用http.Get()：

```
func Get(url string) (resp *Response, err error)
```

- 回傳的Response結構中，屬性Body(即request body, 回應主體)會包含傳回的內容，事後應該用Body.Close()關閉它，且它符合io.Writer介面的規範，可以用io.ReadAll來讀取內容

- `Response`的屬性`StatusCode`則是請求的HTTP狀態碼(整數), 2XX(通常是200)代表請求成功
- 但務必留意的是, 即使狀態非2XX, `http.Get()`也不會傳回錯誤, 你必須檢查`StatusCode`才能知道請求是否成功

練習：用Go HTTP客戶端對網路伺服器傳送GET請求

- 首先匯入需要的套件：

```
1  package main
2
3  import (
4      "fmt"
5      "io"
6      "log"
7      "net/http"
8  )
9
```

```
9
10 func getDataAndReturnResponse() string {
11     //送出GET請求和取得會應
12     resp, err := http.Get("http://google.com")
13     if err != nil {
14         log.Fatal(err)
15     }
16     defer resp.Body.Close() //在結束時釋放r占用的連線資源
17
18     if resp.StatusCode != http.StatusOK { //檢查HTTP狀態碼
19         log.Fatal(resp.Status)
20     }
21
22     data, err := io.ReadAll(resp.Body)
23     if err != nil {
24         log.Fatal(err)
25     }
26
27     return string(data) //回傳回應內容
28 }
29
30 func main() {
31     data := getDataAndReturnResponse()
32     fmt.Print(data) //印出回應內容
33 }
34
```

執行結果

```
PS D:\git\Golang\ch14\14-3-1> go run .
<!doctype html><html itemscope="" itemtype="http://schema.org/WebPage" lang="zh-TW"><head><meta content="text/html" type="Content-Type"><meta content="/images/branding/googlelog/1x/googlelog_standard_color_128dp.png" itemprop="image" nonce="8ePWulRsNzvom_s-Ag4Prw">(function(){window.google={kEI:'9LUJY5-KE8HcmAXaqpUQDQ',kEXPI:'0,1302536,568,5,5367,1123753,1197785,609,380097,16114,28684,17572,4859,1361,9291,3029,17579,4998,13228,3847,10622,22741,508,40,1983,4314,3514,606,2025,1775,520,14670,3227,2845,7,5599,19390,8228,553,1851,2614,13142,3,346,230,6460,148,658,4164,3193,13658,4437,16786,5797,2560,4097,4049,3,3541,1,14261,27893,2,14022,6001,8115,11623,5679,1020,238,370,2465,14967,4333,20,7464,445,2,2,1,17312,9320,8155,6582,799,1401,13279,1290,872,19634,7,1922,9779,23,11327,5761,2173,3667,1224,3,2652,552,983,123,700,4,1,2,2,2,2,3752,3339,1,761,549,1322,2951,1242,1206,5329,493,3359
```

- 以上練習對 `http://www.google.com/` 送出GET請求，並將回應列印在主控台
- 執行結果乍看像亂碼，但若將它存在一個HTML檔和用瀏覽器打開，就會出現Google搜尋首頁
- 這其實就是使用網頁時瀏覽器暗中做的事 --- 從伺服器取得並解讀結構化的資料，然後顯示成網頁

14-3-2 取得並解析伺服器的JSON資料

- 儘管HTML, JavaScript等原始碼適合拿來顯示網頁, 但並不適合在機器間交換資料
- 網路API很常用JSON格式資料, 因為對人或機器來說, JSON資料結構良好, 不管是人或機器都能輕易讀懂
- 我們稍後會來看如何以go語言從Web API取得並解析JSON資料
- 在以下練習中, 我們要來在go程式從伺服器取得結構化的JSON資料, 並用`json.Unmarshal()`解析成結構形式

練習：以Go HTTP客戶端存取JSON資料

- 為了模擬JSON資料交換，本練習的伺服器和客戶端會由兩支不同的程式負責
- 你必須在專案資料夾下建立兩個子目錄, `server` 和 `client`

伺服器程式

- 首先，在`serve`子目錄建立`server.go`
- 這支程式會建立一個非常簡單的伺服器，它會在收到客戶端的請求後傳回一段簡單的**JSON**資料
- 在下一章才會解釋這支程式的原理；目前就當作是配合客戶端的程式

```
1  package main
2
3  import (
4      "log"
5      "net/http"
6  )
7
8  type server struct{} //伺服器結構
9
10 //收到請求時要執行的伺服器服務
11 func (srv server) ServeHTTP(w http.ResponseWriter, r *http.Request) {
12     msg := `{"message": "hello world"}` //要傳回給客戶端的JSON資料
13     w.Write([]byte(msg))                //將JSON字串寫入回應主體
14 }
15
16 func main() {
17     //啟動本地端伺服器，監聽port 8080(即http://localhost:8080)
18     log.Fatal(http.ListenAndServe(":8080", server{}))
19 }
20
```

客戶端程式

```
1  package main
2
3  import (
4      "encoding/json"
5      "fmt"
6      "io"
7      "log"
8      "net/http"
9  )
10
11 type messageData struct { //對應JSON資料的結構
12     Message string `json:"message"`
13 }
14
```

```
15 func getDataAndReturnResponse() messageData {
16     //對http://localhost:8080送出GET請求
17     r, err := http.Get("http://localhost:8080")
18     if err != nil {
19         log.Fatal(err)
20     }
21     defer r.Body.Close()
22
23     //從回應主體讀出所有內容字串
24     data, err := io.ReadAll(r.Body)
25     //解析JSON字串並將資料存入\message(messageData結構)
26     if err != nil {
27         log.Fatal(err)
28     }
29
30     message := messageData{}
31     err = json.Unmarshal(data, &message)
32     if err != nil {
33         log.Fatal(err)
34     }
35
36     return message
37 }
38
39 func main() {
40     data := getDataAndReturnResponse()
41     fmt.Println(data.Message) //印出message欄位的字串
42 }
43
```

執行練習題

- 首先先開啟一個主控台，並啟動伺服器服務(這個伺服器會一直執行到你按下ctrl + c 或主控台關閉)：

```
PS D:\git\Golang\ch14\14-3-2\server> go run server.go  
□
```

- 接著開啟另一個主控台，並執行客戶端程式，以下是執行結果：

```
PS D:\git\Golang\ch14\14-3-2\client> go run .  
hello world
```

14-4 用POST請求傳送資料給伺服器

- 想傳送資料到伺服器時，最常用的方式是透過**POST**請求
- 常見的例子就是登入表格：
 - 當你按下表格的送出鈕，該表格會對某個**URL**送出**POST**請求，接著網路伺服器通常會檢查登入細節是否正確，是的話就更新我們的登入狀態，並回應**POST**請求說登入成功

- **POST**請求是如何傳送資料的呢？所有**HTTP**訊息(請求和回應)其實都包含三個部分：**URL** / 標頭(header) / 主體(body)
- **POST**請求會將要送出的資料夾帶在請求主體中，而不是像**GET**請求那樣透過**URL**參數

14-4-1 送出POST請求並接收回應

- 在go語言要送出POST請求，可以使用`http.Post()`函式：

```
func Post(url, contentType string, body io.Reader) (resp *Response, err error)
```

- `url` 參數即請求的網址，而`contentType`參數為請求標頭中的Content-Type欄位要指定的內容類型，以一般的文字或網頁資料來說就是`text/html`
 - 這回我們要傳送JSON資料，因此將之定義為 `'application/json'`
- 最後一個參數`body`是`io.Reader`介面型別，即我們要讓POST請求夾帶的資料；在下面的練習中會使用`bytes.NewBuffer()`來建立一個`bytes.Buffer`結構，他同樣時做了`io.Reader`

練習：使用Go HTTP客戶端對網路伺服器傳送POST請求

- 在這個練習中，要讓客戶端對伺服器送出一個**POST**請求，當中包含一個**JSON**字串
- 然後伺服器會將字串中的**message**訊息轉成全大寫然後回傳

伺服器程式

- 以下程式會實作一個非常基本的網路伺服器，會接收**POST**請求的**JSON**資料並將回覆給客戶端：

```
1  package main
2
3  import (
4      "encoding/json"
5      "log"
6      "net/http"
7      "strings"
8  )
9
10 type server struct{}
11
12 type messageData struct {
13     Message string `json:"message"`
14 }
15
```

```
15
16 //伺服器服務
17 func (srv server) ServeHTTP(w http.ResponseWriter, r *http.Request) {
18     message := messageData{}
19     err := json.NewDecoder(r.Body).Decode(&message)
20     if err != nil {
21         log.Fatal(err)
22     }
23     log.Println(message) //印出收到的資料
24
25     //將訊息轉成大寫
26     message.Message = strings.ToUpper(message.Message)
27     //將message重新編碼成JSON資料
28     jsonBytes, _ := json.Marshal(message)
29     w.Write(jsonBytes) //傳回給客戶端
30 }
31
32 func main() {
33     log.Fatal(http.ListenAndServe(":8080", server{}))
34 }
35
```

- 注意到這裡改用`json.NewDecoder()`來傳回一個`Decoder`結構，並使用後者的`Decode()`來解析JSON資料
- 既然`NewDecoder()`可以接收一個`io.Reader()`介面型別，就不必再用`io.ReadAll()`先將它讀成字串了

客戶端程式

```
1  package main
2
3  import (
4      "bytes"
5      "encoding/json"
6      "fmt"
7      "log"
8      "net/http"
9  )
10
11  type messageData struct {
12      Message string `json:"message"`
13  }
14
```

```
15 func postDataAndReturnResponse(msg messageData) messageData {
16     jsonBytes, _ := json.Marshal(msg) //將要傳的結構編碼成JSON
17     buffer := bytes.NewBuffer(jsonBytes) //將JSON轉成bytes.Buffer
18
19     //送出POST請求和資料，標頭為application/json，並接收回應
20     r, err := http.Post("http://localhost:8080", "application/json", buffer)
21     if err != nil {
22         log.Fatal(err)
23     }
24     defer r.Body.Close()
25
26     message := messageData{}
27     //解碼伺服器回應的JSON資料
28     err = json.NewDecoder(r.Body).Decode(&message)
29     if err != nil {
30         log.Fatal(err)
31     }
32
33     return message
34 }
35
36 func main() {
37     msg := messageData{Message: "hi server!"} //要傳給伺服器的訊息
38     data := postDataAndReturnResponse(msg) //接收伺服器串回的訊息
39
40     fmt.Println(data.Message)
41 }
42
```

執行練習題：

- 先執行serve子目錄下的server.go：

```
PS D:\git\Golang> cd .\ch14\14-4-1\server\  
PS D:\git\Golang\ch14\14-4-1\server> go run .\server.go  
□
```

- 再用第二個終端機執行client子目錄下的main.go：

```
PS D:\git\Golang> cd .\ch14\14-4-1\client\  
PS D:\git\Golang\ch14\14-4-1\client> go run .  
HI SERVER!
```


- 執行完後再回到伺服器的主控台，可以看到伺服器印出以下訊息，顯示有收到客戶端傳來的資料：

```
PS D:\git\Golang> cd .\ch14\14-4-1\server\  
PS D:\git\Golang\ch14\14-4-1\server> go run .\server.go  
2022/08/28 18:00:20 {hi server!}
```

14-4-2 用POST請求上傳檔案

- 再看一個常見範例：你想從本地端用**POST**請求上傳一個檔案到伺服器，很多網站會用這種方法讓使用者上傳照片之類的東西
- 為此，我們得在**POST**請求標頭中指定使用**MIME**(多用途網際網路郵件擴散)格式來傳送檔案；這種標準能將檔案切割成較小的訊息傳送，且支援多媒體類型
- 可以想像，上傳檔案會比單純上傳表格資料更複雜：客戶端必須將上傳的檔案轉換成**MIME**格式，而伺服器收到後也要讀取他，在將之寫入系統中
- 幸好，**go**語言標準套件**mime/multipart**可以替我們應付**MIME**物件的建立，並傳回適當的請求標頭(**multipart/form-data**)

伺服器程式

- 下面同樣來實作一個簡單的伺服器，它會從請求主體讀取使用者傳送的**MIME**檔案，並將其內容寫入系統
- 稍後在客戶端程式部分，會看到如何產生傳送用的**MIME**檔案

```
1  package main
2
3  import (
4      "fmt"
5      "io"
6      "log"
7      "net/http"
8      "os"
9  )
10
11  type server struct{}
12
```

```
13 func (srv server) ServeHTTP(w http.ResponseWriter, r *http.Request) {
14     //從請求主體取出名稱為myFile的檔案 (multipart.File 型別)
15     file, fileHeader, err := r.FormFile("myFile")
16     if err != nil {
17         log.Fatal(err)
18     }
19     defer file.Close()
20
21     //multipart.File符合io.Reader介面，故可用io.ReadAll()讀取內容
22     fileContent, err := io.ReadAll(file)
23     if err != nil {
24         log.Fatal(err)
25     }
26
27     //將檔案寫入伺服器端的系統
28     err = os.WriteFile(fmt.Sprintf("./%s", fileHeader.FileName), fileContent, 0666)
29     if err != nil {
30         log.Fatal(err)
31     }
32
33     //顯示並回傳已上傳檔案的訊息
34     log.Printf("%s uploaded!", fileHeader.FileName)
35     w.Write([]byte(fmt.Sprintf("%s uploaded!", fileHeader.FileName)))
36 }
37
38 func main() {
39     log.Fatal(http.ListenAndServe(":8080", server{}))
40 }
41
```

- 這裡使用了 `os.WriteFile()` 函式，它能建立一個新檔案，並將一個 `[]byte` 切片寫入到其內容，不需要另外開啟檔案物件
- 接著我們看看，客戶端是如何建立 **MIME** 檔案來給伺服器解讀的

客戶端程式

```
1  package main
2
3  import (
4      "bytes"
5      "fmt"
6      "io"
7      "log"
8      "mime/multipart"
9      "net/http"
10     "os"
11 )
12
13 func postFileAndReturnResponse(filename string) string {
14     fileDataBuffer := bytes.Buffer{} //建立一個buffer
15     mpWritter := multipart.NewWriter(&fileDataBuffer) //建立multipart.Writer
16
17     file, err := os.Open(filename)
18     if err != nil {
19         log.Fatal(err)
20     }
21     defer file.Close()
22
23     //用multipart.Writer 建立準備傳送的MIME檔案
24     formFile, err := mpWritter.CreateFormFile("myFile", file.Name())
25     if err != nil {
26         log.Fatal(err)
27     }
28 }
```

```
28
29 //將原始檔案的內容拷貝到MIME
30 if _, err := io.Copy(formFile, file); err != nil {
31     log.Fatal(err)
32 }
33 mpWriter.Close()
34
35 //用POST請求送出MIME檔案並讀取回應
36 //使用multipart.Writer來指定標頭內的内容類型為 multipart/form-data
37 r, err := http.Post("http://localhost:8080", mpWriter.FormDataContentType(), &fileDataBuffer)
38 if err != nil {
39     log.Fatal(err)
40 }
41 defer r.Body.Close()
42
43 data, err := io.ReadAll(r.Body)
44 if err != nil {
45     log.Fatal(err)
46 }
47
48 return string(data)
49 }
50
51 func main() {
52     data := postFileAndReturnResponse("./test.txt")
53     fmt.Println(data)
54 }
55
```

執行練習題

- 先執行serve子目錄的server.go :

```
PS D:\git\Golang\ch14\14-4-2\server> go run server.go
```



- 接著在client子目錄新增一個test.txt, 並用第二個終端機執行client子目錄的main.go :

```
PS D:\git\Golang\ch14\14-4-2\client> go run .  
test.txt uploaded!
```

- 回到server子目錄, 可以發現該資料夾下面多了一個test.txt, 顯示檔案上傳成功, 且伺服器的主控台也會顯示上傳訊息:

```
PS D:\git\Golang\ch14\14-4-2\server> go run server.go  
2022/08/29 15:04:06 test.txt uploaded!  
█
```

14-5 在客戶端使用自訂標頭做為請求選項

- 前面我們看到**POST**請求可以設定標頭的內容類型
- 但對於任何請求，你當然也可以設定其他標頭，甚至是加入自訂的標頭，作為對伺服器的請求選項
- 一個常見的例子就是授權標頭：當你註冊了一個服務時，它會回傳一個授權碼，然後你之後呼叫該服務的**API**時，請求裡都必須包含這個授權碼，好讓伺服器來驗證你的身分

- 為了自定標頭，需要使用`http.NewRequest()`產生一個`http.Request`結構：

```
func NewRequest(method, url string, body io.Reader) (*Request, error)
```

- `method`為HTTP方法(即使GET或POST等等)，接著`url`是網址，`body`則是請求主體
- 而取得物件後，就可以對它指定標頭(以下程式碼說明)
- 接著，你要使用一個`http.Client`結構的`Do()`方法來執行這個請求：

```
func (c *Client) Do(req *Request) (*Response, error)
```

- 你可以隨意建立`Client`結構，不過這個練習會使用http套件的預設客戶端
`DefaultClient`

練習：在GET請求加入授權標頭

- 在這個練習裡，要來打造自己的HTTP客戶端，並加入授權標頭
- 伺服器只有在請求包含正確的標頭和授權碼時，才會傳回你需要的訊息

伺服器程式碼

```
1  package main
2
3  import (
4      "log"
5      "net/http"
6      "time"
7  )
8
9  type server struct{}
10
11 func (srv server) ServeHTTP(w http.ResponseWriter, r *http.Request) {
12     //讀取授權標頭Authorization
13     auth := r.Header.Get("Authorization")
14     if auth != "superSecretToken" { //若授權碼不符就拒絕授權
15         w.WriteHeader(http.StatusUnauthorized) //回應設為 HTTP code 401
16         w.Write([]byte("Authorization token not recognized"))
17         return
18     }
19
20     //授權成功，等待兩秒後回應
21     time.Sleep(time.Second * 2)
22     msg := "Hello client!" //回傳通過授權的訊息
23     w.Write([]byte(msg))
24 }
25
26 func main() {
27     log.Fatal(http.ListenAndServe(":8080", server{}))
28 }
29
```

註：

- Authorization是HTTP協定中已經存在的標頭欄位
- `http.ResponseWriter`介面的`WriteHeader()`方法能用來設定回應的HTTP狀態碼，如果沒有呼叫它就會預設傳回`http.StatusOK`(即200)

客戶端程式

```
1  package main
2
3  import (
4      "fmt"
5      "io"
6      "log"
7      "net/http"
8      "time"
9  )
10
11 type test struct{}
12
13 func getDataWithCustomOptionsAndReturnResponse() string {
14     //建立一個GET請求
15     req, err := http.NewRequest("GET", "http://localhost:8080", nil)
16     if err != nil {
17         log.Fatal(err)
18     }
19     req.Header.Set("Authorization", "superSecretToken")
20
21     //設定預設客戶端的請求逾時時間為5秒
22     http.DefaultClient.Timeout = time.Second * 5
23 }
```

```
24 //將授權碼放入授權標頭 Authorization, 值為 "superSecretToken"
25 resp, err := http.DefaultClient.Do(req) //讓預設客戶端送出請求
26 if err != nil {
27     log.Fatal(err)
28 }
29 defer resp.Body.Close()
30
31 data, err := io.ReadAll(resp.Body) //讀取回應主體
32 if err != nil {
33     log.Fatal(err)
34 }
35
36 return string(data) //傳回伺服器的回應
37 }
38
39 func main() {
40     data := getDataWithCustomOptionsAndReturnResponse()
41     fmt.Println(data)
42 }
43
```


執行練習題：

- 先執行serve子目錄的server.go：

```
PS D:\git\Golang\ch14\14-5\server> go run .\server.go  
□
```

- 接著在第二個終端機執行client子目錄的main.go，等兩秒後會看到：

```
PS D:\git\Golang\ch14\14-5\client> go run .  
Hello client!
```

本章結束