GABRIELE ROMANATO

Effettuare richieste HTTP in Go con i package core

Go offre un supporto eccellente per lavorare con richieste HTTP grazie al suo pacchetto `net/http`. In questo articolo vediamo come effettuare richieste GET e POST utilizzando esclusivamente i package inclusi nella libreria standard di Go.

Effettuare una richiesta GET

Per inviare una richiesta GET, possiamo utilizzare la funzione `http.Get`. Ecco un esempio:

```
package main

import (
    "fmt"
    "io"
    "net/http"
)

func main() {
    resp, err := http.Get("https://example.com")
    if err != nil {
        panic(err)
    }
    defer resp.Body.Close()

    body, err := io.ReadAll(resp.Body)
    if err != nil {
        panic(err)
    }

    fmt.Println(string(body))
}
```

Nel codice sopra:

- Chiamiamo `http.Get` con l'URL desiderato.
- Controlliamo eventuali errori e ricordiamo di chiudere il body della risposta con `defer`.
- Usiamo `io.ReadAll` per leggere tutto il contenuto della risposta.

Effettuare una richiesta POST

Per inviare dati con una richiesta POST, possiamo utilizzare `http.Post` oppure creare manualmente una richiesta con `http.NewRequest`. Ecco un esempio semplice con `http.Post`:

```
package main

import (
    "bytes"
    "fmt"
    "io"
    "net/http"
)

func main() {
    data := []byte(`{"name":"John","age":30}`)
    resp, err := http.Post("https://example.com/api",
"application/json", bytes.NewBuffer(data))
    if err != nil {
        panic(err)
    }
    defer resp.Body.Close()

    body, err := io.ReadAll(resp.Body)
    if err != nil {
        panic(err)
    }

    fmt.Println(string(body))
}
```

Qui:

- Prepariamo i dati da inviare come JSON.
- Chiamiamo `http.Post` specificando URL, tipo di contenuto e body.
- Leggiamo e stampiamo la risposta come nel caso della GET.

Gestire richieste personalizzate

Per maggiore controllo, possiamo creare manualmente una richiesta usando `http.NewRequest` e inviarla con `http.Client`:

```
package main

import (
    "bytes"
    "fmt"
    "io"
    "net/http"
)

func main() {
    client := &http.Client{}

    req, err := http.NewRequest("PUT",
"https://example.com/resource",
bytes.NewBuffer([]byte(`{"update":"true"}`)))
    if err != nil {
        panic(err)
    }
    req.Header.Set("Content-Type", "application/json")

    resp, err := client.Do(req)
    if err != nil {
        panic(err)
    }
    defer resp.Body.Close()

    body, err := io.ReadAll(resp.Body)
    if err != nil {
        panic(err)
```

```
    }

    fmt.Println(string(body))
}
```

Questo approccio consente di:

- Specificare il metodo HTTP (come PUT, PATCH, DELETE).
- Aggiungere intestazioni personalizzate.
- Utilizzare un `http.Client` personalizzato.

Conclusione

Con il pacchetto `net/http` e le funzioni di `io`, Go rende semplice inviare richieste HTTP in modo robusto e performante, senza dover ricorrere a librerie esterne.