Context使用

tony

Outline

1. Context取消goroute任务
2. Context进行超时控制
3. Context传递通用参数

Context使用

1. context.WithCancel

   a. 返回一个cancel函数，调用cancel函数的时候，会触发context.Done()函数

```
package main

import (
    "context"
    "log"
    "os"
    "time"
)

var logg *log.Logger

func someHandler() {
    ctx, cancel := context.WithCancel(context.Background())
    go doStuff(ctx)
    time.Sleep(10 * time.Second)
    cancel()

}

//每1秒work一下，同时会判断ctx是否被取消了，如果是就退出
func doStuff(ctx context.Context) {
    for {
        time.Sleep(1 * time.Second)
        select {
        case <-ctx.Done():
            logg.Printf("done")
            return
        default:
            logg.Printf("work")
        }
    }
}

func main() {
    logg = log.New(os.Stdout, "", log.Ltime)
    someHandler()
    logg.Printf("down")
}
```

Context使用

2. context. WithTimeout

   a. 超过指定时间之后，会触发context.Done函数

示例1

```
package main

import (
    "context"
    "log"
    "os"
    "time"
)

var logg *log.Logger

func timeoutHandler() {
    // ctx, cancel := context.WithTimeout(context.Background(), 5*time.Second)
    ctx, cancel := context.WithDeadline(context.Background(), time.Now().Add(5*time.Second))
    // go doTimeOutStuff(ctx)
    go doStuff(ctx)

    time.Sleep(10 * time.Second)
    cancel()
}

//每1秒work一下，同时会判断ctx是否被取消了，如果是就退出
func doStuff(ctx context.Context) {
    for {
        time.Sleep(1 * time.Second)
        select {
        case <-ctx.Done():
            logg.Printf("done")
            return
        default:
            logg.Printf("work")
        }
    }
}

func main() {
    logg = log.New(os.Stdout, "", log.Ltime)
    someHandler()
    logg.Printf("down")
}
```

示例2

```
package main

import (
    "net/http"
    "math/rand"
    "fmt"
    "time"
)


func lazyHandler(w http.ResponseWriter, req *http.Request) {
    ranNum := rand.Intn(2)
    if ranNum == 0 {
        time.Sleep(6 * time.Second)
        fmt.Fprintf(w, "slow response, %d\n", ranNum)
        fmt.Printf("slow response, %d\n", ranNum)
        return
    }
    fmt.Fprintf(w, "quick response, %d\n", ranNum)
    fmt.Printf("quick response, %d\n", ranNum)
    return
}

func main() {
    http.HandleFunc("/", lazyHandler)
    http.ListenAndServe(":9200", nil)
}
```

示例2

```
package main

import (
    "context"
    "net/http"
    "fmt"
    "sync"
    "time"
    "io/ioutil"
)
var (
    wg sync.WaitGroup
)
type ResPack struct {
    r *http.Response
    err error
}
func work(ctx context.Context) {
    tr := &http.Transport{}
    client := &http.Client{Transport: tr}
    defer wg.Done()
    c := make(chan ResPack, 1)

    req, _ := http.NewRequest("GET", "http://localhost:9200", nil)
    go func() {
        resp, err := client.Do(req)
        pack := ResPack{r: resp, err: err}
        c <- pack
    }()
    select {
    case <-ctx.Done():
        tr.CancelRequest(req)
        <-c
        fmt.Println("Timeout!")
    case res:= <-c:
        if res.err != nil {
            fmt.Println(res.err)
            return
        }
        defer res.r.Body.Close()
        out, _ := ioutil.ReadAll(res.r.Body)
        fmt.Printf("Server Response: %s", out)
    }
    return
}
func main() {
    ctx, cancel := context.WithTimeout(context.Background(), 2 * time.Second)
    defer cancel()
    wg.Add(1)
    go work(ctx)
    wg.Wait()
    fmt.Println("Finished")
}
```

Context使用

3. 传递上下文通用参数

   a. context.WithValue(ctx, “key”, value)，把参数设置到context中。

   b. context.Value(“key”) 获取参数。