链滴

golang+elasticsearch 爬取网易云音乐评论(整理版本 1)

作者：xhaoxiong

原文链接：https://ld246.com/article/1523811621913

来源网站：链滴

稍微整理了一下的爬取，在文件V2.0中

入口文件main.go

```
package main

import (
    "go-wyy/models"
    "go-wyy/v2.0/engin"
    "io"
    "github.com/PuerkitoBio/goquery"
    "sync"
    "go-wyy/v2.0/fetcher"
    "go-wyy/v2.0/parse"
)

func main() {
    //ticker := time.Tick(200 * time.Millisecond)

    engin.Run(
        engin.Request{
            Url:           "",
            ParserComment: engin.NilParser,
            ParserSong: func(reader io.Reader) engin.ParseResult {
                doc, err := goquery.NewDocumentFromReader(reader)

                if err != nil {
                    panic(err)
                }

                wg := &sync.WaitGroup{}
                result := engin.ParseResult{}
                doc.Find("ul[class=f-hide] a").Each(func(i int, selection *goquery.Selection) {
                    /*开启协程插入数据库，并且开启协程请求每首歌的评论*/
                    songIdUrl, _ := selection.Attr("href")
                    title := selection.Text()
                    var song models.Song
                    //歌曲id
                    songId := songIdUrl[9:len(songIdUrl)]
                    song.SongId = songId

                    ///song?id=歌曲id
                    song.SongUrlId = songIdUrl

                    //歌曲标题
                    song.Title = title
                    //fmt.Printf("歌曲题目:%s\n", title)
                    result.Items = append(result.Items, "歌曲信息:", song)
                    offset := 0
                    songComment := make(chan [][]byte, 100)
                    go func(offset int) {
                        for {
                            fetcher.GetComments(songId, offset, offset+40, songComment, wg)
```

```
                offset += 40
            }

        }(offset)
        go parse.ReceiveComment(songComment, wg)
        wg.Add(2)
    })
    wg.Wait()
    return result
},
})
}
```

这里写的很不简洁

本来ParseSong后面的闭包函数应该写在parse文件中的，进行抽象化

这里很明显的是在engine中传入了用户id

然后获取到页面reader 然后解析进行goquery遍历

获取歌曲id

最后fetcher评论数据

关于网易云的其他逻辑知乎上一抓一大把的最难部分还是加密破解那一块，

现在仍然存在请求过多ip被封的情况

v2.0文件夹中直接运行main.go就能看到测试获取数据情况，如果需要存入数据库则需要在receivecoment函数中进行处理

参考简书部分

</br>

Github

----------分割线（以下是整理版本1）----------

目的和准备

目的：

为了知道自己歌单中每首歌的评论，然后通过歌曲id来检索这首歌的所有评论，并且想熟悉运用golan中的channel以及整个爬虫架构

准备：

语言：golang

编辑器: goland

处理：goquery/正则

存储：mysql（elasticsearch）

V1版本（单机版，mysql存储）

models

评论struct[comment.go]

首先通过查看某一首音乐的请求获到ajax((eg: https://music.163.com/weapi/v1/resource/comments/R_SO_4_340394?csrf_token=)
`)

请求的 json, 通过返回的json数据构造出struct

```
//每首歌的评论
type Commentt struct {
    Id          int64
    IsMusician bool  `json:"isMusician"`
    UserId      int32 `json:"userId"`
    //TopComments []string        `json:"topComments";gorm:"-"`
    MoreHot     bool            `json:"moreHot"`
    HotComments []*HotComments `json:"hotComments"`
    Code        int             `json:"code"`
    Comments    []*Comments     `json:"comments"`
    Total       int64           `json:"total"`
    More        bool            `json:"more"`
    SongId      string
}



//具体到每条评论（普通评论）
type Comments struct {
    Id                  int64
    User                *User        `json:"user"`
    BeReplied           []*BeReplied `json:"-"`
    Time                int64        `json:"time"`
    LikedCount          int          `json:"likedCount"`
    Liked               bool         `json:"liked"`
    CommentId           int64        `json:"commentId"`
    Content             string       `json:"content";gorm:"type:longtext"`
    IsRemoveHotComment bool         `json:"isRemoveHotComment"`
    Commentt            *Commentt
    CommenttID          int64
}

//热门评论
type HotComments struct {
    Id          int64
    User        *User        `json:"user"`
```

```
    BeReplied  []*BeReplied `json:"-"`
    Time       int64        `json:"time"`
    LikedCount int          `json:"likedCount"`
    Liked      bool         `json:"liked"`
    CommentId  int64        `json:"commentId"`
    Content    string       `json:"content";gorm:"type:longtext"`
    Commentt   *Commentt
    CommenttID int64
}

//评论的用户信息
type User struct {
    Id           int64
    LocationInfo *LocationInfo `json:"-"`
    UserType     int           `json:"userType"`
    ExpertTags   *ExpertTag    `json:"-"`
    UserId       int64         `json:"userId"`
    NickName     string        `json:"nickName"`
    Experts      *Expert       `json:"-"`
    AuthStatus   int           `json:"authStatus"`
    RemarkName   *RemarkName   `json:"-"`
    AvatarUrl    string        `json:"avatarUrl"`
    VipType      int           `json:"vipType"`
    Comments     *Comments
    CommentsID   int64
}


//答复用户的评论信息
type BeReplied struct {
    Id           int64
    User         *User  `json:"-"`
    UserID       int64
    Content      string `json:"content"`
    Status       int    `json:"status"`
    CommentsID   int64
    HotCommentsID int64
}

type LocationInfo struct {
}

type ExpertTag struct {
}

type Expert struct {
}

type RemarkName struct {
}
```

然后又查看了歌单信息看一了下网页Dom结构构造出如下struct

歌曲列表struct[songList.go]

```
type Song struct {
    Id          int64
    SongUrlId   string
    SongId      string
    Title       string
    DownloadUrl string
    UserId      string
}

type PlayList struct {
    Id     int64
    UserId string
    Songs  []Song
}
```

然后又找了一下歌曲下载的接口发现只能在pc端下载，于是在一篇博客中找到了一些关于golang爬网易云音乐的具体实现点击此处

歌曲下载struct[download.go]

```
type DownloadData struct {
    Data []*Data `json:"data"`
    Code int     `json:"code"`
}

type Data struct {
    Id        int64   `json:"id"`
    Url       string  `json:"url"`
    Br        int64   `json:"br"`
    Md5       string  `json:"md_5"`
    Code      int     `json:"code"`
    Expi      int     `json:"expi"`
    Type      string  `json:"type"`
    Gain      float64 `json:"gain"`
    Fee       int     `json:"fee"`
    Uf        *Uf     `json:"-"`
    Payed     int     `json:"payed"`
    Flag      int     `json:"flag"`
    CanExtend bool    `json:"can_extend"`
}

type Uf struct {
}
```

推荐一个很好用的json转struct工具

以上就是所有数据库表

连接数据库[base.go]

我选择了gorm来操作数据库

```
/**如果缺少某些包则需要自己去go get*/

/**
    "fmt"
    "net/url"
    "github.com/jinzhu/gorm"(需要goget)
    "os"
    "time"
    "log"
    "crypto/md5"
    "encoding/hex"
    "database/sql"
*/

var DB *gorm.DB



var dbconf *conf.DbConf
var db_host,db_port,db_name,db_user,db_pass string
//初始化加载配置文件
func init() {

    dbconf, err := dbconf.Load("database.json")
    if err != nil {
        fmt.Println(err)
    }
    db_host = dbconf.DbHost
    db_port = dbconf.DbPort
    db_name = dbconf.DbName
    db_user = dbconf.DbUser
    db_pass = dbconf.DbPass
}

//自动建表
func SyncDB() {
    createDB()
    Connect()
    DB.
        Set("gorm:table_options", "ENGINE=InnoDB").
        AutoMigrate(
        &AdminUser{},
        &Commentt{},
        &Comments{},
        &HotComments{},
        &Song{},
        &User{},
```

```
    )
}

//添加后台admin_user(可以不用)
func AddAdmin() {
    var user AdminUser
    fmt.Println("please input username for system administrator")
    var name string
    fmt.Scanf("%s", &name)
    fmt.Println("please input password for system administrator")
    var password string
    fmt.Scanf("%s", &password)
    user.Username = name
    h := md5.New()
    h.Write([]byte(password))
    user.Password = hex.EncodeToString(h.Sum(nil))
    if err := DB.Create(&user).Error; err != nil {
        fmt.Println("admin create error,please run this application again")
        os.Exit(0)
    } else {
        fmt.Println("admin create finished")
    }
}

//连接数据库
func Connect() {
    dsn := fmt.Sprintf("%s:%s@tcp(%s:%s)/%s?charset=utf8mb4&loc=%s&parseTime=true",
        db_user,
        db_pass,
        db_host,
        db_port,
        db_name,
        url.QueryEscape("Asia/Shanghai"))

    var err error

    DB, err = gorm.Open("mysql", dsn)
    if err != nil {
        log.Print("master detabase connect error:", err)
        os.Exit(0)
    }

    DB.SingularTable(true)
    DB.DB().SetMaxOpenConns(2000)
    DB.DB().SetMaxIdleConns(100)
    DB.DB().SetConnMaxLifetime(100 * time.Nanosecond)
}

//创建数据库
func createDB() {

    dsn := fmt.Sprintf("%s:%s@tcp(%s:%s)/?charset=utf8mb4&loc=%s&parseTime=true", db_u
er, db_pass, db_host, db_port, url.QueryEscape("Asia/Shanghai"))
    sqlstring := fmt.Sprintf("CREATE DATABASE  if not exists `%s` CHARSET utf8mb4 COLLATE u
```

```
f8mb4_general_ci", db_name)
    db, err := sql.Open("mysql", dsn)
    if err != nil {
        panic(err.Error())
    }
    r, err := db.Exec(sqlstring)
    if err != nil {
        log.Println(err)
        log.Println(r)
    } else {
        log.Println("Database ", db_name, " created")
    }
    defer db.Close()

}
```

main.go中initArgs()函数初始化(通过命令行添加后台用户，同步数据表)

eg：go run main.go -syncdb(这样就可以同步数据库表了)

```
func main() {
    initArgs()
}
func initArgs() {
    args := os.Args

    for _, v := range args {
        if v == "-syncdb" {
            models.SyncDB()
            os.Exit(0)
        }
        if v == "-admin" {
            models.Connect()
            models.AddAdmin()
            os.Exit(0)
        }
    }
}
```

---

service

encrypt[关于爬取网易云音乐评论的关键，已经有大佬在知乎解密]

首先在看完知乎的分析之后，于是去了解一下aes加密。

参考博客

总结来说：就是获得两个加密参数params和encSecKey

```go
// EncParams 传入参数 得到加密后的参数和一个也被加密的秘钥
func EncParams(param string) (string, string, error) {
    // 创建 key
    secKey := createSecretKey(16)

    aes1, err1 := aesEncrypt(param, nonce)

    // 第一次加密 使用固定的 nonce
    if err1 != nil {
        return "", "", err1
    }
    aes2, err2 := aesEncrypt(aes1, secKey)
    // 第二次加密 使用创建的 key
    if err2 != nil {
        return "", "", err2
    }
    // 得到 加密好的 param 以及 加密好的key
    return aes2, rsaEncrypt(secKey, pubKey, modulus), nil
}
```

params获取步骤：

1、构造一个特定json参数param

```
eg: `{"rid": "` + rid + `","offset": "` + strOffset + `","total": "` + total + `","limit": "` + strLimit + `",
csrf_token": ""}`
```

2、对 param进行两次 aes加密得到 params

```
keys = "abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789+/"
```

- 从该　keys中先获得一个16位的随机字符串 key密钥aesKey

```go
// 创建指定长度的key(size=16)
func createSecretKey(size int) string {
    // 也就是从 a~9 以及 +/ 中随机拿出指定数量的字符拼成一个 size长的key
    rs := ""
    for i := 0; i < size; i++ {
        pos := rand.Intn(len([]rune(keys)))
        rs += keys[pos: pos+1]
    }
    return rs
}
```

```
nonce = "0CoJUm6Qyw8W8jud"
```

- 第一次　aes加密得到 aes1(param参数与固定的nonce密钥)
- 第二次　aes加密(第一次加密param后得到的aes1与之前得到的16位aesKey)

下面是具体的aes加密过程

```go
// 通过 CBC模式的AES加密 用 sKey 加密 sSrc
```

```go
func aesEncrypt(sSrc string, sKey string) (string, error) {
    iv := []byte(iv)
    block, err := aes.NewCipher([]byte(sKey))
    if err != nil {
        return "", err
    }
    //需要padding的数目
    padding := block.BlockSize() - len([]byte(sSrc))%block.BlockSize()
    //只要少于256就能放到一个byte中，默认的blockSize=16(即采用16*8=128, AES-128长的密钥)
    //最少填充1个byte，如果原文刚好是blocksize的整数倍，则再填充一个blocksize

    src := append([]byte(sSrc), bytes.Repeat([]byte{byte(padding)}, padding)...)


    model := cipher.NewCBCEncrypter(block, iv)
    cipherText := make([]byte, len(src))
    model.CryptBlocks(cipherText, src)
    // 最后使用base64编码输出
    return base64.StdEncoding.EncodeToString(cipherText), nil
}
```

- 获取 encKey（模仿python的思路做了一遍）

```python
python:
def rsaEncrypt(text, pubKey, modulus):

    text = text[::-1]

    rs = int(text.encode('hex'), 16)**int(pubKey, 16) % int(modulus, 16)

    return format(rs, 'x').zfill(256)
```

```go
// 将 key 也加密
func rsaEncrypt(key string, pubKey string, modulus string) string {
    // 倒序 key
    rKey := ""
    for i := len(key) - 1; i >= 0; i-- {
        rKey += key[i: i+1]
    }
    // 将 key 转 ascii 编码 然后转成 16 进制字符串
    hexRKey := ""
    for _, char := range []rune(rKey) {
        hexRKey += fmt.Sprintf("%x", int(char))
    }
    // 将 16进制 的 三个参数 转为10进制的 bigint
    bigRKey, _ := big.NewInt(0).SetString(hexRKey, 16)
    bigPubKey, _ := big.NewInt(0).SetString(pubKey, 16)
    bigModulus, _ := big.NewInt(0).SetString(modulus, 16)
    // 执行幂乘取模运算得到最终的bigint结果
    bigRs := bigRKey.Exp(bigRKey, bigPubKey, bigModulus)
    // 将结果转为 16进制字符串
```

```
    hexRs := fmt.Sprintf("%x", bigRs)
    // 可能存在不满256位的情况，要在前面补0补满256位
    return addPadding(hexRs, modulus)
}

// 补0步骤
func addPadding(encText string, modulus string) string {
    ml := len(modulus)
    for i := 0; ml > 0 && modulus[i:i+1] == "0"; i++ {
        ml--
    }
    num := ml - len(encText)
    prefix := ""
    for i := 0; i < num; i++ {
        prefix += "0"
    }
    return prefix + encText
}
```

最后得到了params和encKey

最重要的一步就做完了[参考了知乎里许多大佬的代码和分析过程，收获颇多]

comment(获取评论)

```
/**
    id:歌曲id
    limit:每次请求条数
    offset:请求起始点
 */
func GetComments(id string, offset int, limit int) (comment *models.Commentt, err error) {

    rid := ""
    strOffset := strconv.Itoa(offset)
    strLimit := strconv.Itoa(limit)
    total := "true"
    initStr1 := `{"rid": "` + rid + `","offset": "` + strOffset + `","total": "` + total + `","limit": "` + str
imit + `","csrf_token": ""}`
    params1, key1, err := encrypt.EncParams(initStr1)
    if err != nil {
        panic(err)
    }
    // 发送POST请求得到最后包含url的结果
    comment, err = Comments(params1, key1, id)

    if err != nil {
        fmt.Println(err)
        return comment, err
    }
    return comment, err
}

//获取某首歌的评论
```

```
func Comments(params string, encSecKey string, id string) (comment *models.Commentt, err error) {
    client := &http.Client{}
    form := url.Values{}
    form.Set("params", params)
    form.Set("encSecKey", encSecKey)
    body := strings.NewReader(form.Encode())
    request, _ := http.NewRequest("POST", "http://music.163.com/weapi/v1/resource/comment/R_SO_4_"+id+"?csrf_token=", body)
    request.Header.Set("Content-Type", "application/x-www-form-urlencoded")
    request.Header.Set("Referer", "http://music.163.com")
    request.Header.Set("Content-Length", (string)(body.Len()))
    request.Header.Set("User-Agent", "Mozilla/5.0 (Windows NT 10.0; WOW64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/53.0.2785.101 Safari/537.36")
    request.Header.Set("Cookie","_ntes_nnid=f2c441d1440900d6daa9611bab3dc027,151512235101; _ntes_nuid=f2c441d1440900d6daa9611bab3dc027; __utmz=94650624.1515122355.1.1.tmcsr=(direct)|utmccn=(direct)|utmcmd=(none); _iuqxldmzr_=32; __remember_me=true; JSESIONID-WYYY=YXZtk7tOBJ4b3gOVrX2hl5%2BBriZyYVR5kNX3D3G5oWFRcY3J1cvGnMJRZx6JXVSRNhFKO3O%5CmRiRACwWjrhBnkmK3dgGyTawDSAAmF%2Fct5T%2BhYVRy1BnxCgx%5CYAUrjnQ8jEJQ1VHJTdNhqS4p9jVxHdRcc7iv5cQn649a%5CsBTc46WR%3A1515402120148; __uta=94650624.753127024.1515122355.1515218466.1515400320.9; __utmc=94650624; MUSIC_=0120a3f48157438f759f2034b3925668ae731f8ae462a842927650798e0d663c97d1b459676c0c693e926b3c390b8ba205ba14613b02d6c02d1ccf53040f6087d9739a0cccfd7eebf122d59fa1e6a2; __csrf=5aa926378397ed694496ebf6486c5dfc; __utmb=94650624.5.10.1515400320")
    // 发起请求
    response, reqErr := client.Do(request)
    // 错误处理
    if reqErr != nil {
        fmt.Println("Fatal error ", reqErr.Error())
        return comment, reqErr
    }
    defer response.Body.Close()

    if response.StatusCode!=http.StatusOK{
        fmt.Println("Error:status code", response.StatusCode)
        return nil, fmt.Errorf("wrong status code:%d", response.StatusCode)
    }


    resBody, _ := ioutil.ReadAll(response.Body)

    err = json.Unmarshal(resBody, &comment)
    if err != nil {
        fmt.Println(err)
        return comment, err
    }
    return comment, nil

}
```

conf(加载数据库配置文件)

```go
type DbConf struct {
    DbHost string `json:"db_host"`
    DbPort string `json:"db_port"`
    DbUser string `json:"db_user"`
    DbPass string `json:"db_pass"`
    DbName string `json:"db_name"`
}

func (this *DbConf) Load(filename string,) (dbconf *DbConf,err error) {
    var dbconff *DbConf
    data, err := ioutil.ReadFile(filename)
    if err != nil {
        fmt.Println("read file error")
        return dbconf,err
    }
    datajson := []byte(data)

    err = json.Unmarshal(datajson, &dbconff)

    if err != nil {

        return dbconff, err

    }
    return dbconff,nil
}
```

songs(歌曲下载和获取用户歌单)

```go
/**
ids: [歌曲id]
rate :320000 普通品质
     640000 高级品质
     160000 低级品质
*/
func GetDownloadUrl(id string, rate string) (data *models.DownloadData, err error) {
    var DownloadData *models.DownloadData
    initStr := `{"ids": "` +"["+ id +"]"+ `", "br": "` + rate + `", "csrf_token": ""}`
    params, key, err := encrypt.EncParams(initStr)
    if err != nil {
        panic(err)
    }
    DownloadData, err = Download(params, key)
    return DownloadData, err
}

// Download 根据传入id返回生成的mp3地址
func Download(params string, encSecKey string) (data *models.DownloadData, err error) {
    var DownloadData *models.DownloadData
    client := &http.Client{}
    form := url.Values{}
    form.Set("params", params)
    form.Set("encSecKey", encSecKey)
```

```
    body := strings.NewReader(form.Encode())
    time.Sleep(500*time.Microsecond)
    request, _ := http.NewRequest("POST", "http://music.163.com/weapi/song/enhance/player
url?csrf_token=", body)
    request.Header.Set("Content-Type", "application/x-www-form-urlencoded")
    request.Header.Set("Content-Length", (string)(body.Len()))
    request.Header.Set("Referer", "http://music.163.com")
    // 发起请求
    response, reqErr := client.Do(request)
    // 错误处理
    if reqErr != nil {
        fmt.Println("Fatal error ", reqErr.Error())
        return DownloadData, err
    }
    defer response.Body.Close()
    resBody, _ := ioutil.ReadAll(response.Body)
    err = json.Unmarshal(resBody, &DownloadData)
    if err != nil {
        panic(err)
    }
    return DownloadData, err
}





/**
    userId 用户Id 该函数用于获取到用户歌单后
    获取每首歌的评论
*/


func Songs(userId string) {
    req, err := http.NewRequest("GET", "http://music.163.com/playlist?id="+userId, nil)

    if err != nil {
        panic(err)
    }
    req.Header.Set("User-Agent", "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/53
.36 (KHTML, like Gecko) Chrome/63.0.3239.84 Safari/537.36")
    req.Header.Set("Referer", "http://music.163.com/")
    req.Header.Set("Host", "music.163.com")

    c := &http.Client{}
    res, err := c.Do(req)
    if err != nil {
        panic(err)
    }

    doc, err := goquery.NewDocumentFromReader(res.Body)

    if err != nil {
```

```
        panic(err)
    }

    g := 0
    wg := &sync.WaitGroup{}

    doc.Find("ul[class=f-hide] a").Each(func(i int, selection *goquery.Selection) {
        /*开启协程插入数据库，并且开启协程请求每首歌的评论*/
        songIdUrl, _ := selection.Attr("href")
        title := selection.Text()
        var song models.Song
        //歌曲id
        songId := songIdUrl[9:len(songIdUrl)]
        song.SongId = songId

        ///song?id=歌曲id
        song.SongUrlId = songIdUrl

        //歌曲标题
        song.Title = title


        song.UserId = userId

        go comment.GetAllComment(songId, wg)

        g++
        wg.Add(1)
    })
    wg.Wait()
}
```

以上是之前写爬去网易云音乐评论的第一个版本

项目目录结构为

```
go-wyy
    |-models
        admin_user.go
        base.go     (连接数据库，建立数据库，自动建表)
        comment.go(评论的struct)
        download.go(下载的struct)
        songList.go(歌单的struct)
    |-service
        |-comment
            comment.go(获取评论的api)
        |-conf
            load.go(加载配置数据库文件)
        |-encrypt
            encrypt.go(params和encKe y的加密)
        |-songs
            download.go(下载歌曲的api)
            songs.go(获取歌单列表每首歌id)
    main.go (自定义逻辑，如果想要存入数据库必须要有models.connect(),以及initArgs()自动建表
```

者自己手动建表)