GRequests: 让 HTTP 服务人类

熟悉我的读者朋友们都知道，我早期是写 Python 的，现在主力语言是 Go。开始接触 Go 语言以后，我发现 Go 自带的 `net/http` 请求库不够好用，好在我没用 Go 写过一行爬虫代码，平时 `net/http` 库用的也就比较少，不是每天都用，也就忍了。

最近我在网上冲浪时，无意间发现了 `grequests` 这个库，灵感来源于 Python 生态中大名鼎鼎的 `requests` 库。顾名思义，这个库就是用来发送 HTTP 请求的。`grequests` 的出现让我眼前一亮，我的 Go 工具包又增添了一员，本文就来带大家一起体验下 `grequests` 库的强大之处。

简介

以前写 Python 的时候，要说用起来最舒服的 HTTP 请求库，当属 `requests` 库了，没有之一。`requests` 库[官方文档](http://cxyroad.com/”https://requests.readthedocs.io/en/latest/”)标题是 `Requests: HTTP for Humans™`，有人将其翻译为`让 HTTP 服务人类`，这也是本文标题的由来，可见这个库旨在降低我们在发送 HTTP 请求时写代码的心智负担。

根据我的实际使用体验，这个标题绝对不是在吹牛，`requests` 库的 API 设计是真的符合直觉，也符合 Python 哲学。

`grequests` 即 Go 版本的 `requests`，官方介绍其为：`A Go ”clone” of the great and famous Requests library`，看来 `grequests` 库是 `requests` 库的复刻。

使用 `net/http` 发送 HTTP 请求

在具体介绍 `grequests` 库的使用方法之前，我们先来回顾下 `net/http` 发送 HTTP 请求的用法，以此来对比 `grequests` 库的强大。

使用 `net/http` 发送 HTTP `GET` 请求示例代码如下：

```
// 发起 HTTP GET 请求
resp, _ := http.Get("https://httpbin.org/get")
defer resp.Body.Close() // 确保关闭响应体

// 读取响应体内容
body, _ := io.ReadAll(resp.Body)

// 将响应体打印为字符串
fmt.Println(string(body))

```

> NOTE: 为了保持代码逻辑清晰，这里只展示了代码主逻辑，并且代码中忽略了所有错误处理。后文中所有示例代码都会如此，完整代码可以在文末给出的示例代码 GitHub 链接中获取。

> NOTE: 示例中请求的网址 `httpbin.org` 是一个叫 `httpbin` 的开源项目，它提供了一个可用于测试和调试 HTTP 客户端（如浏览器、命令行工具、自定义脚本或任何发送 HTTP 请求的软件）的 HTTP 服务。这个服务模拟了各种 HTTP 场景，可以帮助开发者测试自己的 HTTP 客户端是否正确处理各种 HTTP 方法、响应等。`httpbin` 也是 `requests` 作者 `Kenneth Reitz` 开发并维护的项目。

使用 `net/http` 发送 HTTP `POST` 请求示例代码如下：

```
// 创建一个要发送的数据结构并编码为 JSON
data := map[string]any{
    "username": "user",
    "password": "pass",
}
jsonData, _ := json.Marshal(data)

// 创建 POST 请求
url := "https://httpbin.org/post"
request, _ := http.NewRequest(http.MethodPost, url,
bytes.NewBuffer(jsonData))
```

```
// 添加请求头，指明发送的内容类型为 JSON
request.Header.Set("Content-Type", "application/json")

// 发送请求并获取响应
client := &http.Client{}
resp, _ := client.Do(request)
defer resp.Body.Close()

// 读取响应体内容
body, _ := io.ReadAll(resp.Body)

// 将响应体打印为字符串
fmt.Println(string(body))
```

可见，使用 `net/http` 发送 HTTP `GET` 请求的代码还算友好，但是发送 `POST` 请求真的是过于繁琐。

好在，现在我们有了 `grequests`。

GRequests

要使用 `grequests` 库就要先对其进行安装：

```
$ go get -u github.com/levigross/grequests
```

极速开始

使用 `grequests` 发送 HTTP `GET` 请求示例代码如下：

```
resp, _ := grequests.Get("https://httpbin.org/get", nil)
defer resp.Close() // 确保关闭响应体

fmt.Println(resp)
fmt.Println(resp.String()) // 实现了 `fmt.Stringer` 接口
```

```

没错，就是这么简单，仅需三行代码即可打印响应结果。一切都是那么的自然，符合直觉，这就是 `requests` API 的强大之处。

我们不再需要使用 `io.ReadAll` 去读取 HTTP 响应体内容，`*grequests.Response` 提供了 `String` 方法，即实现了 `fmt.Stringer` 接口。

#### QueryString

`grequests` 库发送 `GET` 请求时传递 QueryString 参数方式如下：

```
ro := &grequests.RequestOptions{
    Params: map[string]string{"Hello": "Goodbye"},
}
resp, _ := grequests.Get("https://httpbin.org/get?Hello=World", ro)
defer resp.Close()

fmt.Println(resp)
fmt.Println(resp.RawResponse.Request.URL)
```

执行以上示例代码，打印结果如下：

```
{
 "args": {
   "Hello": "Goodbye"
 },
 "headers": {
   "Accept-Encoding": "gzip",
   "Host": "httpbin.org",
   "User-Agent": "GRequests/0.10",
   "X-Amzn-Trace-Id": "Root=1-6657cca7-
715b811a08dff4420f487570"
 },
 "origin": "69.28.52.250",
```
```

```
  "url": "https://httpbin.org/get?Hello=Goodbye"
}

https://httpbin.org/get?Hello=Goodbye

```

可以发现，现在 `url` 已经被替换成了 `https://httpbin.org/get?Hello=Goodbye` ，可见 `ro` 会覆盖 `url` 中的同名参数。

略显遗憾的是，打印请求 URL 的代码稍显冗长：`resp.RawResponse.Request.URL`，如果能够像 `requests` 库那样提供 `resp.Url` 属性直接返回请求的 `url` 值就更友好了。

POST 请求

发送 `POST` 请求更能体现 `grequests` 库的方便之处，示例代码如下：

```
// 创建一个要发送的数据结构
postData := map[string]string{
    "username": "user",
    "password": "pass",
}

// 将数据结构编码为 JSON 并准备请求选项
ro := &grequests.RequestOptions{
    JSON: postData, // grequests 自动处理 JSON 编码
    // Data: postData,
}

// 发起 POST 请求
resp, _ := grequests.Post("https://httpbin.org/post", ro)
defer resp.Close()

// 输出响应体内容
fmt.Println("Response:", resp.String())

```

这个示例代码明显比使用 `net/http` 发送 `POST` 请求的示例代码整洁不少。

`grequests` 会根据 `*grequests.RequestOptions` 的属性值，自动处理请求头中的 `Content-Type`。

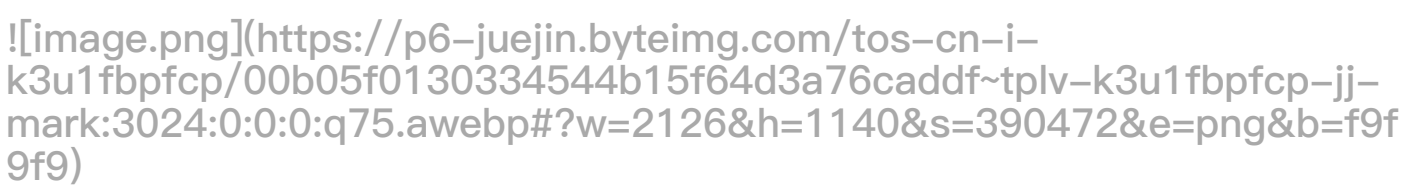
你可以自行尝试把 `grequests.RequestOptions` 中的 `JSON` 属性改为 `Data`，即如下写法：

```
ro := &grequests.RequestOptions{
    Data: postData,
}
```

看看打印结果如何。

HTTP Basic Auth

在 `requests` 库[官方文档](http://cxyroad.com/”https://requests.readthedocs.io/en/latest/”)首页，有一个示例展示了如何使用 `requests` 设置 Basic Auth 认证，简洁优雅，截图如下：



现在我们尝试用 `grequests` 来实现此功能：

```
ro := &grequests.RequestOptions{Auth: []string{”user”, ”pass”}}
resp, _ := grequests.Get(”https://httpbin.org/basic-auth/user/pass”, ro)
defer resp.Close()

if resp.Ok != true {
    log.Println(”Request did not return OK”)
}

fmt.Println(resp.StatusCode)
```

```
fmt.Println(resp.Header.Get("content-type"))
fmt.Println(resp.String())

m := make(map[string]any)
_ = resp.JSON(&m)
fmt.Println(m)
```

可以发现，`grequests` 库同样能够通过简洁友好的方式来设置 Basic Auth 认证信息。

`*grequests.Response` 提供了 `JSON` 方法，可以将响应体的 `JSON` 信息反序列化到对象中。

相较于 `requests` 示例，`grequests` 唯一不足的是没有提供 `r.encoding` 属性可以方便获取响应结果编码格式。

上传和下载文件

`grequests` 库上传和下载文件同样非常简单。

下载内容到文件：

```
resp, _ := grequests.Get("https://httpbin.org/get", nil)
defer resp.Close()

if resp.Ok != true {
    log.Println("Request did not return OK")
}

// 下载响应体内容到 result.json
_ = resp.DownloadToFile("result.json")
```

`*grequests.Response` 提供了 `DownloadToFile` 方法可以直接将响应体内容写入指定文件。

上传文件内容到服务端：

```
// 从 result.json 读取文件内容并上传到服务端
fd, _ := grequests.FileUploadFromDisk("result.json")

// This will upload the file as a multipart mime request
resp, _ := grequests.Post("https://httpbin.org/post",
    &grequests.RequestOptions{
        Files: fd,
        Data:  map[string]string{"One": "Two"},
    })
defer resp.Close()

if resp.Ok != true {
    log.Println("Request did not return OK")
}

fmt.Println(resp)
```

上传文件可以通过 `*grequests.RequestOptions` 的 `Files` 属性来传递，并且可以同时使用 `Data` 参数传递表单数据（`form`）。

总结

本文带大家一起体验了下 `grequests` 库，展示了其强大之处。

相较于 Go 语言内置的 `net/http` 库，`grequests` 的 API 设计更为优雅易用，尤其从 `POST` 请求的对比中可以发现 `grequests` 的 API 要简洁不少。

`grequests` 库能够以简洁友好的方式设置 Basic Auth 认证信息。并且，上传和下载文件同样只需要很少的代码即可实现。

这些强大之处的灵感，都来源于 Python 生态中的 `requests` 库。

当然，`grequests` 也有一些小遗憾，它并没有将 `requests` 库的全部便捷功能复制过来。但总体而言，`grequests` 的出现让我们在 Go 语言中发送 HTTP 请求方便了不少。

`grequests` 库更多用法可以参考[官方文档](http://cxyroad.com/ ”https://pkg.go.dev/github.com/levigross/grequests”)。

本文示例源码我都放在了 [GitHub](http://cxyroad.com/ ”https://github.com/jianghushinian/blog-go-example/tree/main/grequests”) 中，欢迎点击查看。

希望此文能对你有所启发。

延伸阅读

* GRequests 源码: [github.com/levigross/g...](http://cxyroad.com/ ”https://github.com/levigross/grequests”)
* GRequests 文档: [pkg.go.dev/github.com/...](http://cxyroad.com/ ”https://pkg.go.dev/github.com/levigross/grequests”)
* requests 源码 [github.com/psf/request...](http://cxyroad.com/ ”https://github.com/psf/requests”)
* requests 文档: [requests.readthedocs.io/en/latest/](http://cxyroad.com/ ”https://requests.readthedocs.io/en/latest/”)
* httpbin 官网: [httpbin.org/](http://cxyroad.com/ ”https://httpbin.org/”)
* 本文 GitHub 示例代码：[github.com/jianghushin...](http://cxyroad.com/ ”https://github.com/jianghushinian/blog-go-example/tree/main/grequests”)

联系我

* 公众号：[Go编程世界](http://cxyroad.com/ ”https://jianghushinian.cn/about/”)
* ：jianghushinian
* 邮箱：[jianghushinian007@outlook.com](http://cxyroad.com/ ”mailto:jianghushinian007@outlook.com”)
* 博客：[jianghushinian.cn](http://cxyroad.com/ ”https://jianghushinian.cn”)

原文链接: https://juejin.cn/post/7375802292820623379