carlmjohnson / **requests** Public

HTTP requests for Gophers

blog.carlmjohnson.net/post/2021/requests-golang-http-client/

MIT license

555 stars 22 forks

Star Watch

Code Pull requests Discussions Actions Wiki Security Insigh

main

**dependabot[bot]** ... ✓ 2 weeks ago

View code

README.md

# Requests GO reference go report A+ Gocover.io mentioned in awesome

## *HTTP requests for Gophers.*

**The problem**: Go's net/http is powerful and versatile, but using it correctly for client requests can be extremely verbose.

**The solution**: The requests.Builder type is a convenient way to build, send, and handle HTTP requests. Builder has a fluent API with methods returning a pointer to the same struct, which allows for declaratively describing a request by method chaining.

Requests also comes with tools for building custom http transports, include a request recorder and replayer for testing.

## Examples

### Simple GET into a string

| code with net/http | code with re |
|---|---|
| see code below | see code below |
| 11+ lines | 5 lines |

```go
req, err := http.NewRequestWithContext(ctx,
        http.MethodGet, "http://example.com", nil)
if err != nil {
        // ...
}
res, err := http.DefaultClient.Do(req)
if err != nil {
        // ...
}
defer res.Body.Close()
b, err := io.ReadAll(res.Body)
if err != nil {
        // ...
}
s := string(b)
```

```go
var s string
err := requests.
        URL("http:/
        ToString(&s
        Fetch(ctx)
```

### POST a raw body

| code with requests | cc |
|---|---|

```go
err := requests.
        URL("https://postman-echo.com/post").
        BodyBytes([]byte(`hello, world`)).
        ContentType("text/plain").
        Fetch(ctx)
```

```go
body := bytes.NewReader(
req, err := http.NewRequ
        "https://postman
if err != nil {
        // ...
}
req.Header.Set("Content-
res, err := http.Default
if err != nil {
        // ...
}
```

| code with requests | cc |
|---|---|
| | `defer res.Body.Close()` `_, err := io.ReadAll(res` `if err != nil {` `// ...` `}` |
| 5 lines | 12+ lines |

## GET a JSON object

| code with requests | |
|---|---|
| `var post placeholder` `err := requests.` `URL("https://jsonplaceholder.typicode.com").` `Pathf("/posts/%d", 1).` `ToJSON(&post).` `Fetch(ctx)` | `var post placehol` `u, err := url.Par` `if err != nil {` `// ...` `}` `u.Path = fmt.Spri` `req, err := http.` `http.Meth` `if err != nil {` `// ...` `}` `res, err := http.` `if err != nil {` `// ...` `}` `defer res.Body.Cl` `b, err := io.Read` `if err != nil {` `// ...` `}` `err := json.Unmar` `if err != nil {` `// ...` `}` |
| 7 lines | 18+ lines |

## POST a JSON object and parse the response

```
var res placeholder
req := placeholder{
        Title:  "foo",
        Body:   "baz",
        UserID: 1,
}
err := requests.
        URL("/posts").
        Host("jsonplaceholder.typicode.com").
        BodyJSON(&req).
        ToJSON(&res).
        Fetch(ctx)
// net/http equivalent left as an exercise for the reader
```

## Set custom headers for a request

```
// Set headers
var headers postman
err := requests.
        URL("https://postman-echo.com/get").
        UserAgent("bond/james-bond").
        ContentType("secret").
        Header("martini", "shaken").
        Fetch(ctx)
```

## Easily manipulate query parameters

```
var params postman
err := requests.
        URL("https://postman-echo.com/get?a=1&b=2").
        Param("b", "3").
        Param("c", "4").
        Fetch(ctx)
        // URL is https://postman-echo.com/get?a=1&b=3&c=4
```

## Record and replay responses

```
// record a request to the file system
var s1, s2 string
err := requests.URL("http://example.com").
        Transport(requests.Record(nil, "somedir")).
        ToString(&s1).
        Fetch(ctx)
check(err)

// now replay the request in tests
err = requests.URL("http://example.com").
        Transport(requests.Replay("somedir")).
        ToString(&s2).
        Fetch(ctx)
```

```
check(err)
assert(s1 == s2) // true
```

## FAQs

See wiki for more details.

### Why not just use the standard library HTTP client?

Brad Fitzpatrick, long time maintainer of the net/http package, wrote an extensive list of problems with the standard library HTTP client. His four main points (ignoring issues that can't be resolved by a wrapper around the standard library) are:

> - Too easy to not call Response.Body.Close.
> - Too easy to not check return status codes
> - Context support is oddly bolted on
> - Proper usage is too many lines of boilerplate

Requests solves these issues by always closing the response body, checking status codes by default, always requiring a `context.Context`, and simplifying the boilerplate with a descriptive UI based on fluent method chaining.

### Why requests and not some other helper library?

There are two major flaws in other libraries as I see it. One is that in other libraries support for `context.Context` tends to be bolted on if it exists at all. Two, many hide the underlying `http.Client` in such a way that it is difficult or impossible to replace or mock out. Beyond that, I believe that none have acheived the same core simplicity that the requests library has.

### How do I just get some JSON?

```
var data SomeDataType
err := requests.
        URL("https://example.com/my-json").
        ToJSON(&data).
        Fetch(ctx)
```

### How do I post JSON and read the response JSON?

```
body := MyRequestType{}
var resp MyResponseType
err := requests.
```

```
        URL("https://example.com/my-json").
        BodyJSON(&body).
        ToJSON(&resp).
        Fetch(ctx)
```

### How do I just save a file to disk?

It depends on exactly what you need in terms of file atomicity and buffering, but this will work for most cases:

```
        err := requests.
                URL("http://example.com").
                ToFile("myfile.txt").
                Fetch(ctx)
```

For more advanced use case, use `ToWriter` .

### How do I save a response to a string?

```
var s string
err := requests.
        URL("http://example.com").
        ToString(&s).
        Fetch(ctx)
```

### How do I validate the response status?

By default, if no other validators are added to a builder, requests will check that the response is in the 2XX range. If you add another validator, you can add `builder.CheckStatus(200)` or `builder.AddValidator(requests.DefaultValidator)` to the validation stack.

To disable all response validation, run `builder.AddValidator(nil)` .

## Contributing

Please create a discussion before submitting a pull request for a new feature.

### Releases

23 tags

## Sponsor this project

**carlmjohnson** Carl Johnson

Sponsor

Learn more about GitHub Sponsors

## Used by 80

+ 72

## Contributors 4

**carlmjohnson** Carl Johnson

**favadi** Diep Pham

**ykalchevskiy** Yan Kalchevskiy

**dependabot[bot]**

## Languages

● **Go** 100.0%