Is Go 1.22's net/http ServeMux all you need?

With the introduction of Go 1.22, is there any reason now to consider 3rd party frameworks?

Introduction - Frameworks

- `net/http` from the Standard Library
- Gin - https://github.com/gin-gonic/gin
  - Fast, built on top of HTTPRouter - https://github.com/julienschmidt/httprouter instead of (`net/http`)
- Fiber - https://github.com/gofiber/fiber
  - Again, fast, but built on top of https://github.com/valyala/fasthttp but lack of HTTP/2
  - Handler's resemble gRPC (I'll come back to this)
- Go Chi - https://github.com/go-chi/chi
  - Built on `net/http`, Good middleware
- Echo - https://echo.labstack.com
  - Built on `net/http`, Good middleware
  - Handler's resemble gRPC (I'll come back to this)

My pre-Go1.22 choice

As an engineer who has been writing gRPC microservices, I'm used to writing "handlers" that tend to follow this pattern:

```
func DoThing(ctx context.Context, req *Request) (*Response, error)
```

Also, I would like some useful syntactic suger. I'm also risk adverse to diverging from `net/http` code.

So, that led me to Echo.

Sample Echo code

Let's look at a simple server implementation.
There's no real implemention code. We'll just implement the router and some middleware only.

Features we may consider

- 1: Routing (functions for specific http Methods): https://echo.labstack.com/docs/quick-start#routing
- 2: Path Params: https://echo.labstack.com/docs/quick-start#path-parameters
- 3: Groups in Routing: https://echo.labstack.com/docs/routing#group
- 4: Middleware: https://echo.labstack.com/docs/quick-start#middleware
    - Convenient and easily understandable interface to add middlware

CODE DEMO

The Go Blog

Routing Enhancements for Go 1.22

Jonathan Amsterdam, on behalf of the Go team
13 February 2024

Go 1.22 brings two enhancements to the `net/http` package's router: method matching and wildcards. These features let you express common routes as patterns instead of Go code. Although they are simple to explain and use, it was a challenge to come up with the right rules for selecting the winning pattern when several match a request.

We made these changes as part of our continuing effort to make Go a great language for building production systems. We studied many third-party web frameworks, extracted what we felt were the most used features, and integrated them into `net/http`. Then we validated our choices and improved our design by collaborating with the community in a GitHub discussion and a proposal issue. Adding these features to the standard library means one fewer dependency for many projects. But third-party web frameworks remain a fine choice for current users or programs with advanced routing needs.

Enhancements

The new routing features almost exclusively affect the pattern string passed to the two `net/http.ServeMux` methods `Handle` and `HandleFunc`, and the corresponding top-level functions `http.Handle` and `http.HandleFunc`. The only API changes are two new methods on `net/http.Request` for working with wildcard matches.

We'll illustrate the changes with a hypothetical blog server in which every post has an integer identifier. A request like `GET /posts/234` retrieves the post with ID 234. Before Go 1.22, the code for handling those requests would start with a line like this:

... and the Standard Library?

Although code reuse has it's benefits, there are also costs. Russ Cox (co-creator) of Go has discussed this before (https://research.swtch.com/deps)

Can we easily achieve the same using the new Standard Library only?
Ref: https://go.dev/blog/routing-enhancements

CODE DEMO

Review (1)

> "is there any reason now to consider 3rd party frameworks?"

Has the new standard library filled the gaps we had?

- 1: Routing (functions for specific http Methods)
  - Yes. It's an improvement.
  - But, lack of compile time "safety"

```
notesGroup.GET("/:id", routes.GetNote) // ECHO

notesGroup.HandleFunc("GET /notes/{id}", routes.GetNote) // Go 1.22 std lib
```

- 2: Path Params
  - Yes. This is is much needed improvement over pre Go1.22

Review (2)

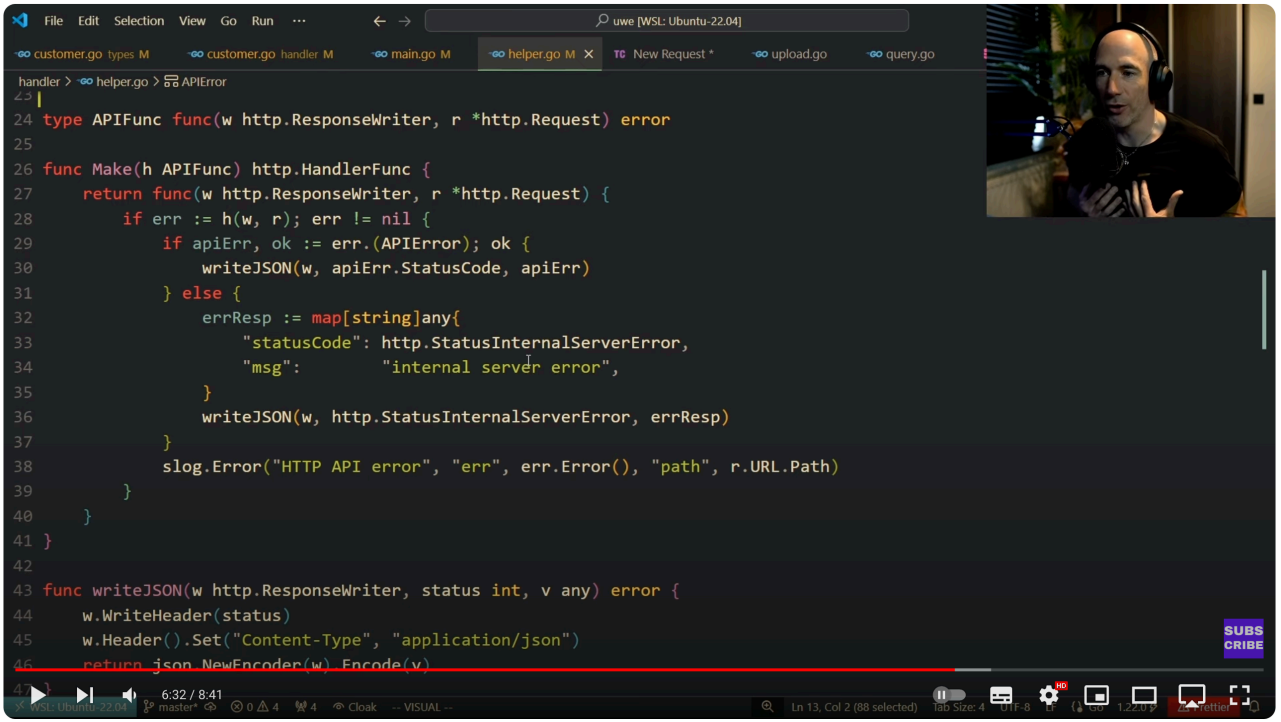
- 3: Groups in Routing
  - This has always been possible pre-Go1.22
  - But the way to setup is not as obvious
    - We could write our own abstraction?
- 4: Middlware
  - Again, this has always been possible pre-Go1.22
    - We could write our own abstraction?

What about centralised error handling?

From https://echo.labstack.com/docs/error-handling

> Echo advocates for centralized HTTP error handling by returning error from middleware and handlers. Centralized error handler allows us to log errors to external services from a unified location and send a customized HTTP response to the client.

I agree with this, I like my handler functions taking the form `func CreateNote(c echo.Context) error`



... something like this?

- I won't take the credit for this code
- This was code presented by Anthony GG
- It shows how we can wrap the Handler functions ourselves instead of relying on a library

echo / echo.go ↑ Top

Code Blame 1021 lines (905 loc) · 34.6 KB Raw

```go
}

// CONNECT registers a new CONNECT route for a path with matching handler in the
// router with optional route-level middleware.
func (e *Echo) CONNECT(path string, h HandlerFunc, m ...MiddlewareFunc) *Route {
    return e.Add(http.MethodConnect, path, h, m...)
}

// DELETE registers a new DELETE route for a path with matching handler in the router
// with optional route-level middleware.
func (e *Echo) DELETE(path string, h HandlerFunc, m ...MiddlewareFunc) *Route {
    return e.Add(http.MethodDelete, path, h, m...)
}

// GET registers a new GET route for a path with matching handler in the router
// with optional route-level middleware.
func (e *Echo) GET(path string, h HandlerFunc, m ...MiddlewareFunc) *Route {
    return e.Add(http.MethodGet, path, h, m...)
}

// HEAD registers a new HEAD route for a path with matching handler in the
// router with optional route-level middleware.
func (e *Echo) HEAD(path string, h HandlerFunc, m ...MiddlewareFunc) *Route {
    return e.Add(http.MethodHead, path, h, m...)
}

// OPTIONS registers a new OPTIONS route for a path with matching handler in the
// router with optional route-level middleware.
func (e *Echo) OPTIONS(path string, h HandlerFunc, m ...MiddlewareFunc) *Route {
    return e.Add(http.MethodOptions, path, h, m...)
}
```

Echo's current status

- Version 5 is taking time; ticket reference from 2001
  - It's router is not taking advantage of the new features from `net/http`
    - The code is well tested, but ...
  - This would simplify Echo so we can easily take advantage of it's "quality of life" abstractions such as middlware and grouping etc.

Conclusions

- It's going to be a personal or pragmatic choice.
  - For a simple personal project with minimal routing requirements, I will now stick with net/http alone
  - Larger projects with more complex middleware and routing requirements, I would still probably stick with Echo for now, and watch what else emerges.
- Recently (in the last 4 days), I found "Michi".
  - https://github.com/go-michi/michi
  - May be worth considering
  - I've also asked the Echo developers about this approach.