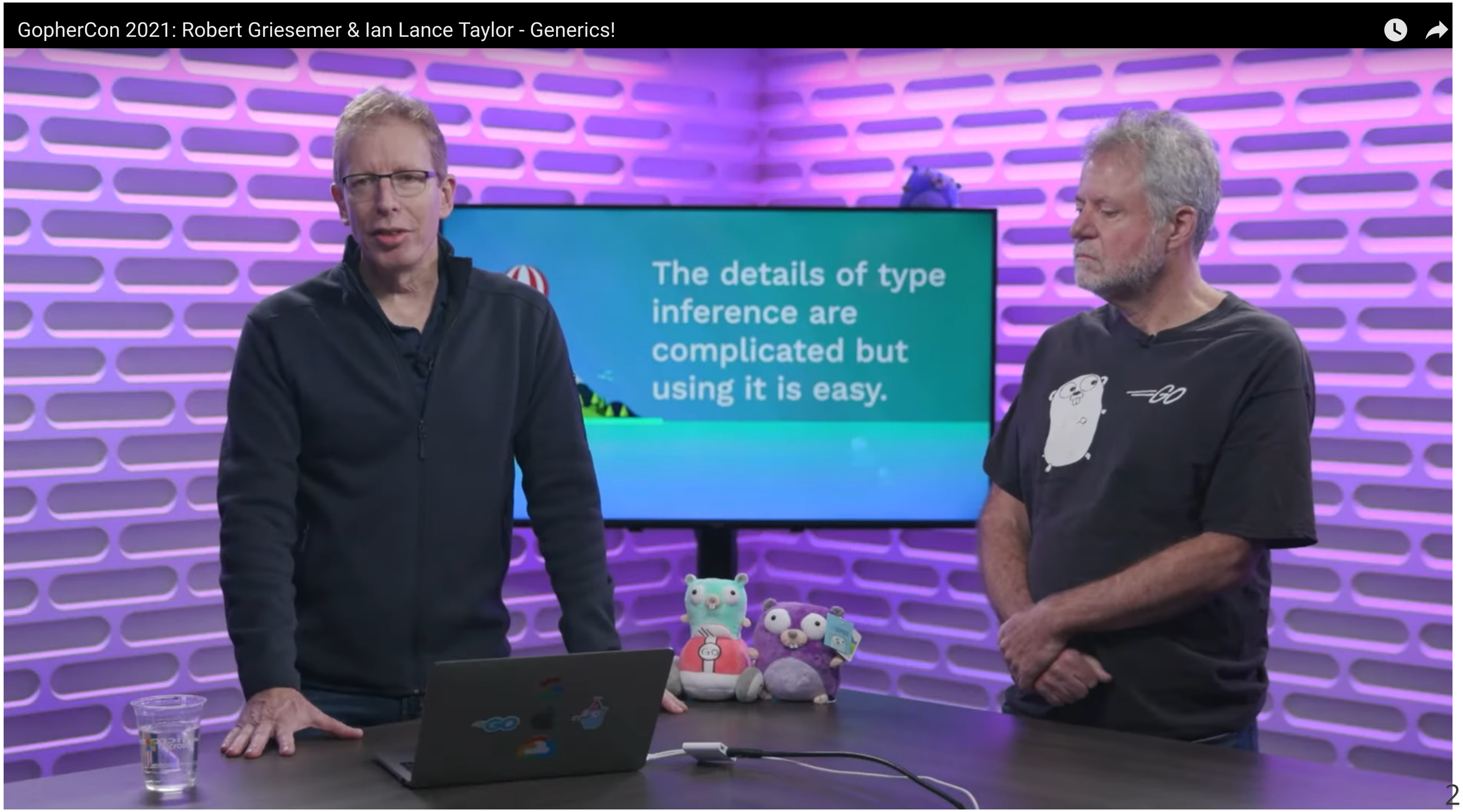
RIP

REST in Peace

3 Feb 2024

Tanguy Herrmann
Senior Software Engineer, Tuxago

Genesis

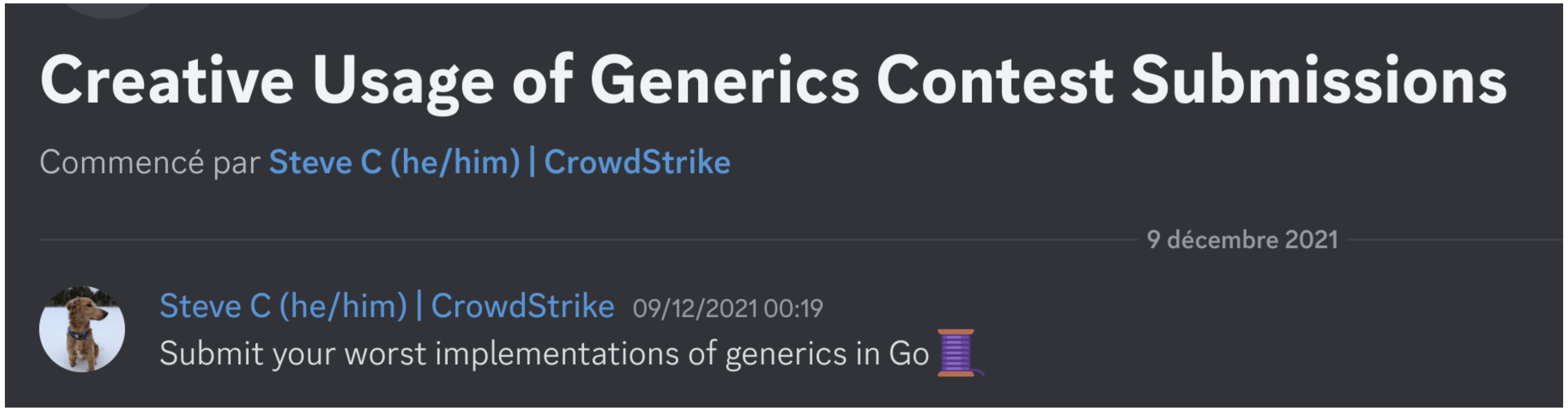

GopherCon 2021: Robert Griesemer & Ian Lance Taylor - Generics!

The details of type inference are complicated but using it is easy.

Ian's advice for generics in Go

0:00 / 0:13

CrowdStrike's Generics Challenge



Creative Usage of Generics Contest Submissions

Commencé par Steve C (he/him) | CrowdStrike

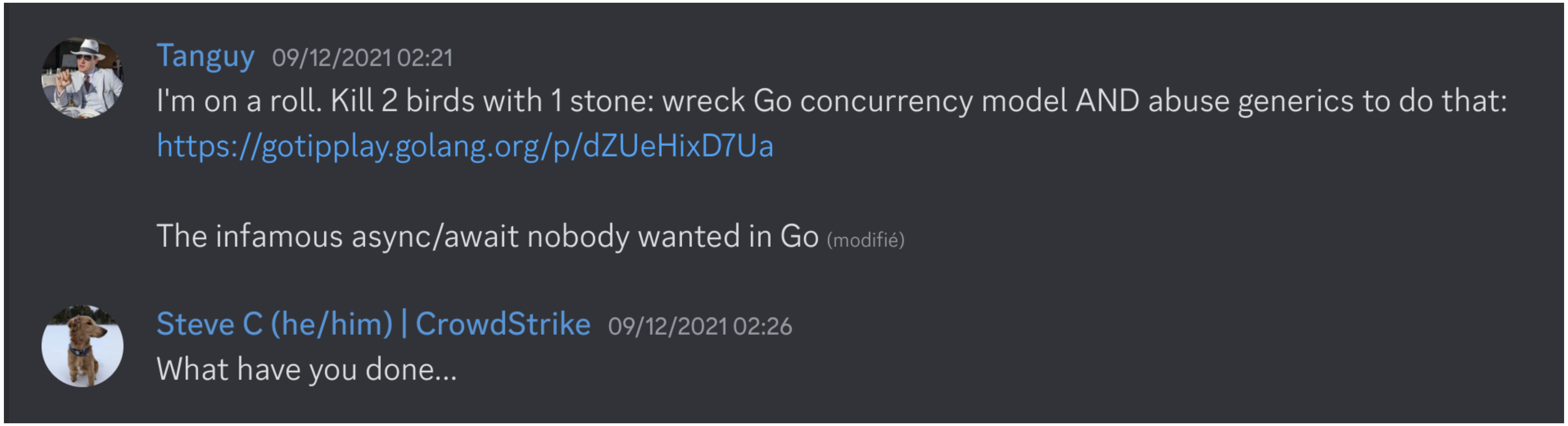
9 décembre 2021

Steve C (he/him) | CrowdStrike 09/12/2021 00:19

Submit your worst implementations of generics in Go 🧵

Original Post (https://discord.com/channels/755435423177638059/918280718013054986/918280720005337158)

Generics Challenge: Async/Await



Tanguy 09/12/2021 02:21

I'm on a roll. Kill 2 birds with 1 stone: wreck Go concurrency model AND abuse generics to do that: https://gotipplay.golang.org/p/dZUeHixD7Ua

The infamous async/await nobody wanted in Go (modifié)

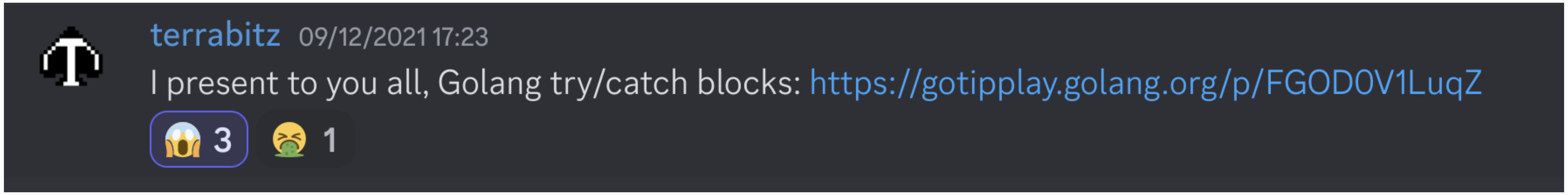
Steve C (he/him) | CrowdStrike 09/12/2021 02:26

What have you done...

Original Post(https://discord.com/channels/755435423177638059/918280718013054986/918311335144587304)

go.dev/play link(https://gotipplay.golang.org/p/dZUeHixD7Ua)

Generics Challenge: Try/Catch



terrabitz 09/12/2021 17:23

I present to you all, Golang try/catch blocks: https://gotipplay.golang.org/p/FGOD0V1LuqZ

😱 3 🤮 1

Original Post (https://discord.com/channels/755435423177638059/918280718013054986/918538237306339348)

go.dev/play link (https://gotipplay.golang.org/p/FGOD0V1LuqZ)

Generics Challenge: success

Generics Challenge: all the submissions

Original Post (https://discord.com/channels/755435423177638059/775877786698776576/918957768533245952)

Element Extraction (@Tanguy) (https://gotipplay.golang.org/p/rtSs9zdtQnl)

New() (@Tanguy) (https://gotipplay.golang.org/p/3UoOMsQZMlW)

ProMess, Async, Await (@Tanguy) (https://gotipplay.golang.org/p/dZUeHixD7Ua)

Print (@Tanguy) (https://gotipplay.golang.org/p/1T5Yn1MFZL6)

Try, Catch, Finally (@terrabitz) (https://gotipplay.golang.org/p/87QFmuQ-OVA)

Monads (@danicat) (https://gotipplay.golang.org/p/lQBorcFUTW5)

Fluent Method Chains (@danicat) (https://gotipplay.golang.org/p/8rqkQPt0CJS)

Perl (@Andy Walker - (he/him)) (https://go.dev/play/p/Yidoeekdtlo?v=gotip)

AJAX (@Ben Woodward | CrowdStrike) (https://gotipplay.golang.org/p/Xryrf6yR9bE)

HTTP Request/Response (@Tanguy) (https://go.dev/play/p/SV-aeh51526?v=gotip)

About me

- Tanguy
- from France
- 17 years in IT
- CEO of HTMX

About me

About me

- freelancer specialized in Go since 2015
- mostly classic RESTful API backends
- some blockchain
- containers in CI/CD as code @dagger_io (https://dagger.io)
- interested in pushing Go in more areas: GUI, video games, AI, embedded, ...

What is this?

```
var t T
err := json.NewDecoder(r.Body).Decode(&t)
if err != nil {
    http.Error(w, err.Error(), http.StatusBadRequest)
    return
}

err = validate(t)
if err != nil {
    http.Error(w, err.Error(), http.StatusBadRequest)
    return
}

resp, err := backendCall(r.Context(), t)
if err != nil {
    http.Error(w, err.Error(), http.StatusInternalServerError)
    return
}

err = json.NewEncoder(w).Encode(resp)
if err != nil {
    http.Error(w, err.Error(), http.StatusInternalServerError)
    return
}
```

What's the point?

```
var t T
err := json.NewDecoder(r.Body).Decode(&t)
if err != nil {
    http.Error(w, err.Error(), http.StatusBadRequest)
    return
}

err = validate(t)
if err != nil {
    http.Error(w, err.Error(), http.StatusBadRequest)
    return
}

resp, err := backendCall(r.Context(), t)
if err != nil {
    http.Error(w, err.Error(), http.StatusInternalServerError)
    return
}

err = json.NewEncoder(w).Encode(resp)
if err != nil {
    http.Error(w, err.Error(), http.StatusInternalServerError)
    return
}
```

Let's add a new handler

```go
var t T2
err := json.NewDecoder(r.Body).Decode(&t)
if err != nil {
    http.Error(w, err.Error(), http.StatusBadRequest)
    return
}

err = validate2(t)
if err != nil {
    http.Error(w, err.Error(), http.StatusBadRequest)
    return
}

resp, err := backendCall2(r.Context(), t)
if err != nil {
    http.Error(w, err.Error(), http.StatusInternalServerError)
    return
}

err = json.NewEncoder(w).Encode(resp)
if err != nil {
    http.Error(w, err.Error(), http.StatusInternalServerError)
    return
}
```

LET ME THINK

Let's add a new one

```
var t T2
err := json.NewDecoder(r.Body).Decode(&t)
if err != nil {
    http.Error(w, err.Error(), http.StatusBadRequest)
    return
}

err = validate2(t)
if err != nil {
    http.Error(w, err.Error(), http.StatusBadRequest)
    return
}

resp, err := backendCall2(r.Context(), t)
if err != nil {
    http.Error(w, err.Error(), http.StatusInternalServerError)
    return
}

err = json.NewEncoder(w).Encode(resp)
if err != nil {
    http.Error(w, err.Error(), http.StatusInternalServerError)
    return
}
```

MAIRIE DE PARIS
NE DÉPOSEZ RIEN SUR LA VOIE PUBLIQUE
TRIEZ VOS DÉCHETS
UTILISEZ LE BAC APPROPRIÉ
I HAVE THE
BEST GARBAGE,
IT IS SO HUGE
imgflip.com

Solution: Abstract the handler

```
type BackendFunc func(ctx context.Context, anyIn interface{}) (anyOut interface{}, err error)
```

```
func Handle(method string, f BackendFunc) http.HandlerFunc {
    return func(w http.ResponseWriter, r *http.Request) {
        var in map[string]interface{}

        err := json.NewDecoder(r.Body).Decode(&in)
        if err != nil {
            http.Error(w, fmt.Errorf("json decode: %w", err).Error(), http.StatusBadRequest)
            return
        }

        out, err := f(r.Context(), in)
        if err != nil {                // err handler behaviour?
            http.Error(w, fmt.Errorf("backend call: %w", err).Error(), http.StatusInternalServerError)
            return
        }

        err = json.NewEncoder(w).Encode(out)
        if err != nil {
            http.Error(w, fmt.Errorf("json encode: %w", err).Error(), http.StatusInternalServerError)
            return
        }
    }
}
```

Abstract the handler: backend wrapper

```
func BackendWrapper(ctx context.Context, anyIn interface{}) (anyOut interface{}, err error) {
    mapIn, ok := anyIn.(map[string]interface{})
    if !ok {
        return Output{}, ErrBadArgument
        // And we need to catch this error in our handler to send a http.StatusBadRequest
    }

    in, err := inputFromMap(mapIn)
    if err != nil {
        return Output{}, ErrBadArgument
    }

    // Do the backend-y stuff
    out, err := realBackendCall(ctx, in)
    if err != nil {
        return Output{}, ErrInternalServer
    }

    return out, nil
}
```

Abstract the handler: input converter

```
func inputFromMap(mapIn map[string]interface{}) (Input, error) {
    ma, ok := mapIn["A"]
    if !ok {
        return Input{}, errors.New("inputFromMap: no A")
    }
    mb, ok := mapIn["B"]
    if !ok {
        return Input{}, errors.New("inputFromMap: no B")
    }
    fa, ok := ma.(float64)
    if !ok {
        return Input{}, fmt.Errorf("inputFromMap: A is not an int: %T", ma)
    }
    // check the number range
    a := int(fa)
    b, ok := mb.(string)
    if !ok {
        return Input{}, fmt.Errorf("inputFromMap: B is not an int: %T", mb)
    }
    return Input{
        A: a,
        B: b,
    }, nil
}
```

Abstract the handler: real backend

```go
func realBackendCall(ctx context.Context, in Input) (Output, error) {
    return Output{
        C: in.A + 2,
        D: in.B + "2",
    }, nil
}
```

Conclusion

- lot of runtime/reflect boilerplate to get back to types
- potential reuse of the handler

If only

Generics to the rescue

Generics: Pros

- better type safety
- better performance than `interface{}`/any (except for this use case)
  - Go check this article from Vicent Marti (https://planetscale.com/blog/generics-can-make-your-go-code-slower) (deprecated?)
- more readable code (`math` package, for example)
- DRY

Without generics: math.Min

```
    x := 1
    y := 2
    z := math.Min(x, y)
    fmt.Println(z)
```

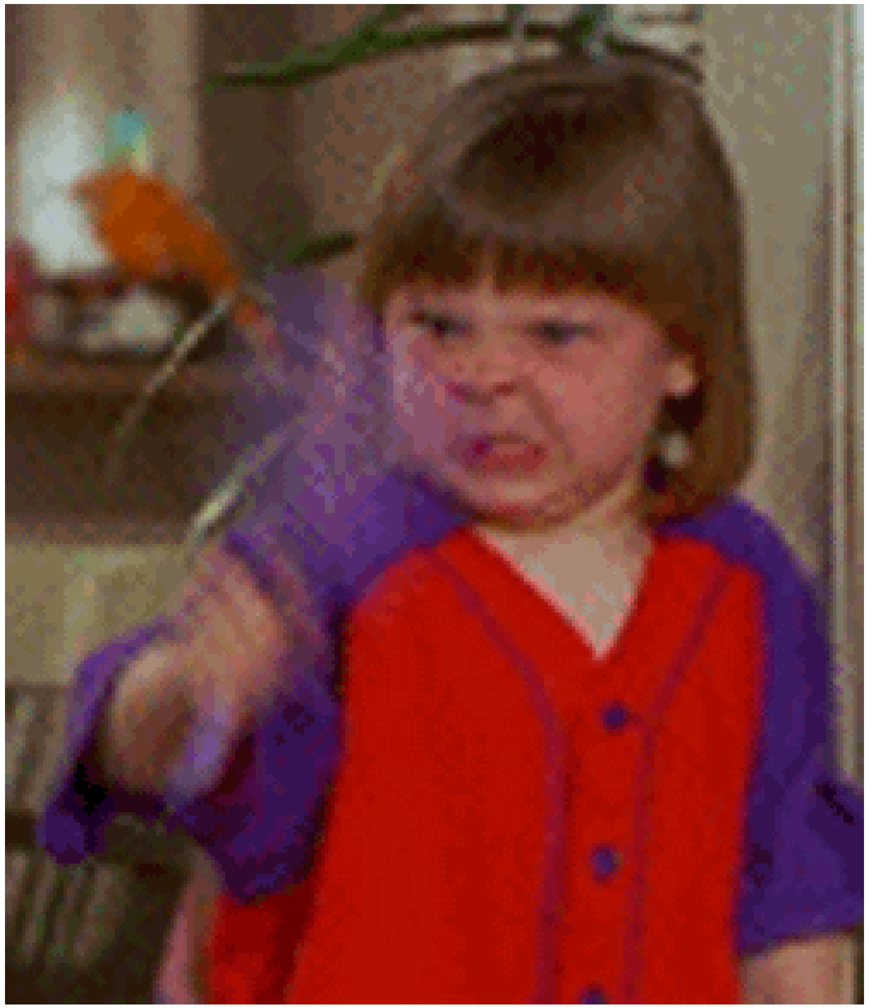
Without generics: math.Min

```
    x := 1
    y := 2
    z := math.Min(x, y)
    fmt.Println(z)
```

```
math.go:11:16: cannot use x (variable of type int) as float64 value in argument to math.Min
math.go:11:19: cannot use y (variable of type int) as float64 value in argument to math.Min
```

Without generics: math.Min: Solution

```
    x := 1
    y := 2
    z := math.Min(float64(x), float64(y))
    fmt.Println(z)
```



With generics: min

```
func min[T cmp.Ordered](x T, y ...T) T
```

With generics: min

```
func min[T cmp.Ordered](x T, y ...T) T
```

```
func Min(x, y float64) float64 {
```

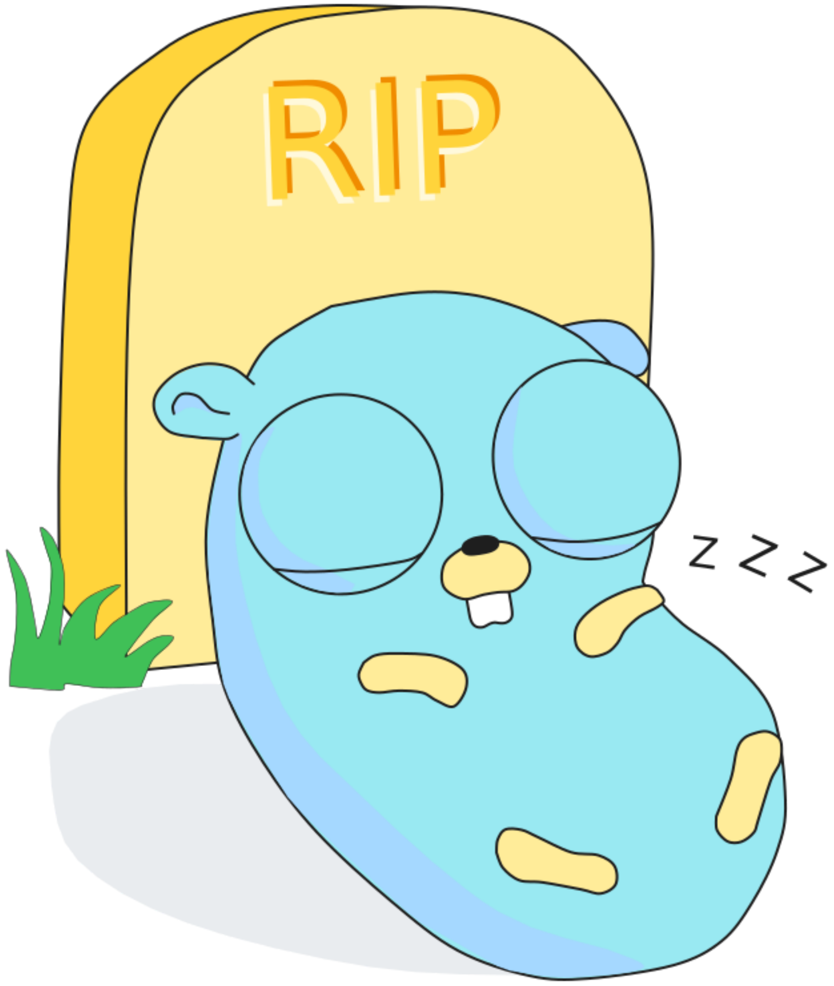
generics library code is less readable

With generics: min: user code

```
    x := 1
    y := 2
    z := min(x, y)
    fmt.Println(z)
```

This just works

RIP



RIP: User code

```go
func Uppercase(ctx context.Context, name string) (string, error) {
	return strings.ToUpper(name), nil
}
```

```go
http.HandleFunc("/uppercase", rip.Handle(http.MethodPost, Uppercase, opts))
```

RIP: Library code

```
// InputOutputFunc is a function that takes a ctx and an input, and it can return an output or an err.
type InputOutputFunc[
    Input, Output any,
] func(ctx context.Context, input Input) (output Output, err error)
```

```
// Handle is a generic HTTP handler that maps an HTTP method to a InputOutputFunc f.
func Handle[
    Input, Output any,
](
    method string, f InputOutputFunc[Input, Output],
    options *RouteOptions,
) http.HandlerFunc {
```

WE NEED TO GO
DEEPER

REST in Peace

A key concept of REST services is the notion of resource

- accessible via a URI
- action on the resource URI via HTTP methods (POST, PUT, GET, DELETE, …)
- current state sent back via HTTP response

User code: Entity handler

```
    up := memuser.NewUserProvider(logger)

    http.HandleFunc(rip.HandleEntities("/users/", up, ro))
```

```
// EntityProvider provides identifiable resources.
type EntityProvider[Ent Entity] interface {
    EntityCreater[Ent]
    EntityGetter[Ent]
    EntityUpdater[Ent]
    EntityDeleter[Ent]
    EntityLister[Ent]
}
```

Lib code

```
// HandleEntities associates an urlPath with an entity provider, and handles all HTTP requests in a REST
//
//    POST   /entities/    : creates the entity
//    GET    /entities/:id : get the entity
//    PUT    /entities/:id : updates the entity (needs to pass the full entity data)
//    DELETE /entities/:id : deletes the entity
//    GET    /entities/    : lists the entities
//
// It also handles fields
//
//    GET    /entities/:id/name : get only the name field of the entity
//    PUT    /entities/:id/name : updates only the name entity field
func HandleEntities[
    Ent Entity,
    EP EntityProvider[Ent],
](
    urlPath string,
    ep EP,
    options *RouteOptions,
) (path string, handler http.HandlerFunc) {
```

What you get

- creation of CRUD HTTP endpoints
- content negociation for many encodings (json, xml, protobuf, msgpack, HTML, HTML Forms, ...)
- automated resource web pages that can edit the resource
- harmonious way of handling common scenarios (unknown resource: return a 404, etc)

Encoding: JSON

```go
package json

import (
    "encoding/json"

    "github.com/dolanor/rip/encoding"
)

var Codec = encoding.WrapCodec(json.NewEncoder, json.NewDecoder, MimeTypes...)

var MimeTypes = []string{
    "application/json",
}
```

Quote

```
RIP is to HTTP what an ORM is to SQL
  - me, probably
```

Demo

github.com/dolanor/rip (https://github.com/dolanor/rip)

Demo

User Code

Route Option

```
ro := rip.NewRouteOptions().
    WithCodecs(
        json.Codec,
        xml.Codec,
        html.Codec,
        html.FormCodec,
    ).
    WithMiddlewares(loggerMW(logWriter))
```

User Code: Entity

```go
type User struct {
    ID           int       `json:"id" xml:"id"`
    BirthDate    time.Time `json:"birth_date" xml:"birth_date"`
    Name         string    `json:"name" xml:"name"`
    EmailAddress string    `json:"email_address" xml:"email_address"`
}

func (u User) IDString() string {
    return strconv.Itoa(u.ID)
}

func (u *User) IDFromString(s string) error {
    n, err := strconv.Atoi(s)
    if err != nil {
        return err
    }
    u.ID = n
    return nil
}
```

User Code: Entity Provider

```
type UserProvider struct {
    mem    map[int]*User
    logger *log.Logger
}
```

```
func (up *UserProvider) Update(ctx context.Context, u *User) error {
    up.logger.Printf("UpdateUser: %+v", u.IDString())
    _, ok := up.mem[u.ID]
    if !ok {
        return rip.ErrNotFound
    }
    up.mem[u.ID] = u

    return nil
}
```

Future

- ~~per-route options (encoding, middleware)~~
- ~~access/update fields independantly (GET/POST/PUT/DELETE `/users/1/name`~~)
- nested resources (eg, `/users/1/posts/1` also points to `/posts/1`)
- pagination
- OpenAPI autogeneration
- more HATEOAS (Hypermedia As The Engine Of Application State)
  - links
  - API auto documentation/explorability
  - support for JSON-LD
- improve the API

Future

- ~~protobuf encoding~~
- use of `log/slog` logger interface
- better error handling
    - ~~better error type~~
    - nice standard way to return errors to user
- customization of HTML template
- authorization of HTML pages
- generation of GUI apps based on your API

Call to action

- feedback
- discussion
- contribution

github.com/dolanor/rip (https://github.com/dolanor/rip)

Thanks

- The Go Team
- Go SXB Go Meetup (Strasbourg)
- Thierry Pfeiffer for the logo
- You for watching that talk
- FOSDEM and the Go devroom organizers
- HTMX

Thank you

REST in Peace

3 Feb 2024

Tags: web, REST, RESTful, golang, Go, generics (#ZgotmplZ)

Tanguy Herrmann
Senior Software Engineer, Tuxago⧓
https://hachyderm.io/@dolanor (https://hachyderm.io/@dolanor)

https://github.com/dolanor/rip (https://github.com/dolanor/rip)