Error handling in Go

Golang Vietnam Meetup #12

Nguyễn Mậu Quang Vũ
Software Engineer at Giao Hang Nhanh

Error handling in Go

- Go turns 5 with Go 1.9 (Mar 2012 – Sep 2017)
- Patterns for handling error in Go
- What people are doing?

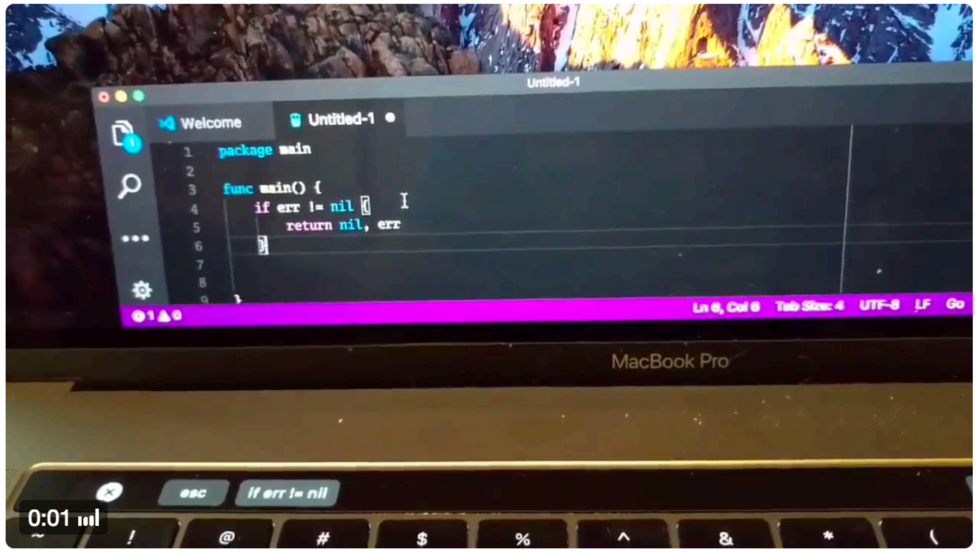
Error handling

```
type error interface {
  Error() string
}

result, err := doSomething()
if err != nil {
  return nil, err
}
```

- Every IO function returns error
- Every API defines error code

```
if err != nil {
  return nil, err
}
```



```
package main

func main() {
    if err != nil {
        return nil, err
    }
```

Example: API flow

API flow

```go
func CreateOrders(ctx context.Context, req *CreateOrdersRequest)
(*CreateOrdersResponse, error) {
    if userID, err := getUserFromContext(ctx); err != nil {
        return nil, err
    }
    if err := verifyPermission(thisFunction, userID); err != nil {
        return nil, err
    }
    if err := validateRequest(req); err != nil {
        return nil, err
    }
    cost, err := calculateCost(req)
    if err != nil {
        return nil, err
    }
    orders, err := newOrders(req)
    if err != nil {
        return nil, err
    }
    err := insertToDB(orders)
    if err != nil {
        return nil, err
```

What can go wrong?

```
Everything.
```

What can go wrong?

- Validate input
- Validate permission
- Verify pre-condition
- Call external services
- Store to DB
- …

What we need to do?

- Validate input -> Respond error code, which fields are invalid.
- Validate permission -> Response error code, why it didn’t work.
- Verify pre-condition -> Response error code
- Call external services -> Network? Rejected? Retry or not?
- Store to DB -> Hmm, internal error? Unique index?
- …

Example: External service

What can go wrong?

SendNotification(deviceTokens, msg) error

What can go wrong?

- Network error
- Protocol error
- Argument error
- Error code
  - Token is invalid
  - Token expired, should delete
  - Device unreachable, should retry later
  - …

Problem with error handling

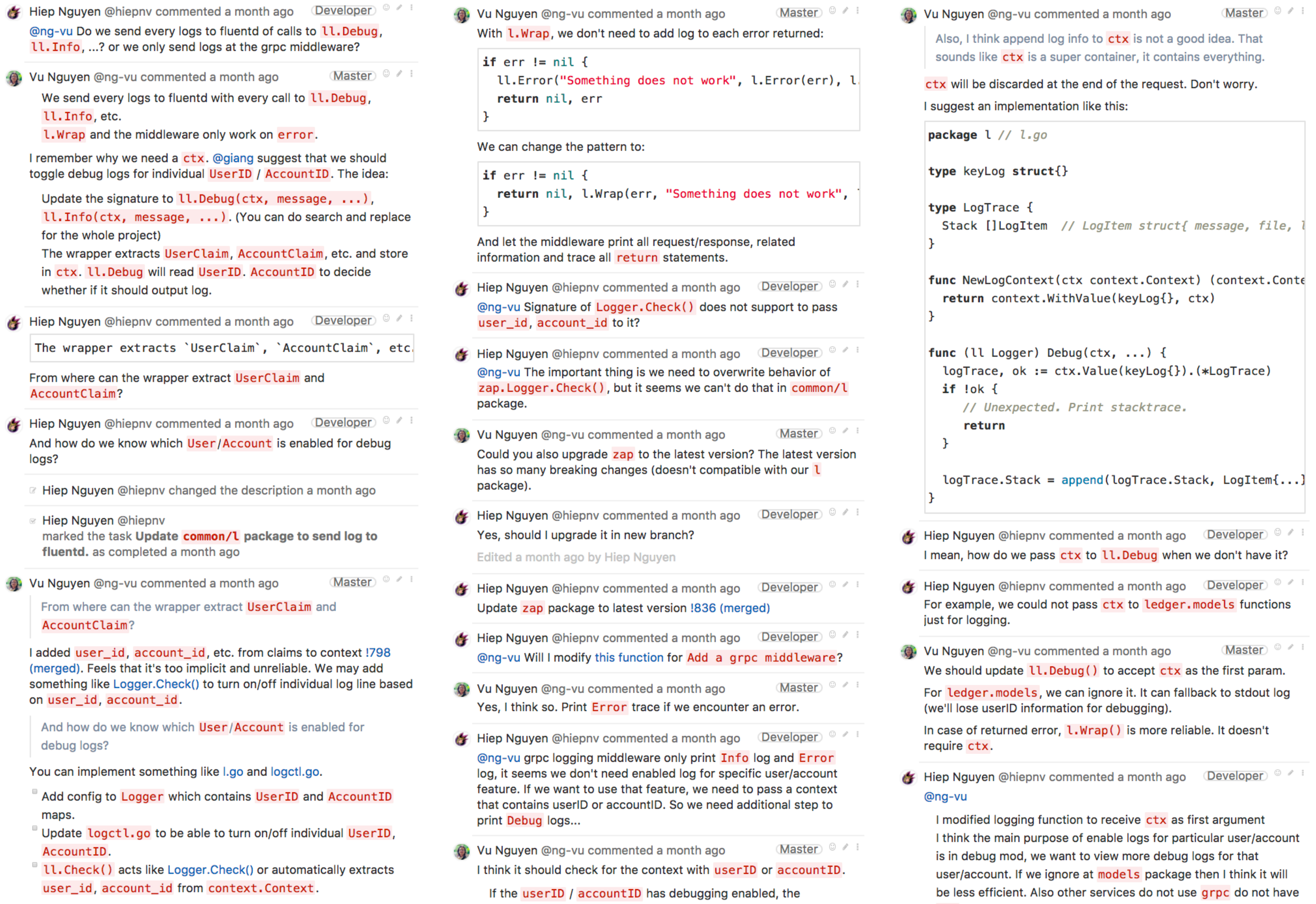
How to deal with them?

So you’ve got an error, what to do?

- Error() returns string
- How do we know what happened?

```go
type error interface {
  Error() string
}


result, err := doSomething()
if err != nil {
  return nil, err
}
```



Hiep Nguyen @hiepnv commented a month ago Developer

@ng-vu Do we send every logs to fluentd of calls to `ll.Debug`, `ll.Info`, ...? or we only send logs at the grpc middleware?

Vu Nguyen @ng-vu commented a month ago Master

We send every logs to fluentd with every call to `ll.Debug`, `ll.Info`, etc.
`l.Wrap` and the middleware only work on `error`.

I remember why we need a `ctx`. @giang suggest that we should toggle debug logs for individual `UserID` / `AccountID`. The idea:

Update the signature to `ll.Debug(ctx, message, ...)`, `ll.Info(ctx, message, ...)`. (You can do search and replace for the whole project)
The wrapper extracts `UserClaim`, `AccountClaim`, etc. and store in `ctx`. `ll.Debug` will read `UserID`. `AccountID` to decide whether if it should output log.

Hiep Nguyen @hiepnv commented a month ago Developer

```
The wrapper extracts `UserClaim`, `AccountClaim`, etc.
```

From where can the wrapper extract `UserClaim` and `AccountClaim`?

Hiep Nguyen @hiepnv commented a month ago Developer

And how do we know which `User`/`Account` is enabled for debug logs?

Hiep Nguyen @hiepnv changed the description a month ago

Hiep Nguyen @hiepnv
marked the task **Update `common/l` package to send log to fluentd.** as completed a month ago

Vu Nguyen @ng-vu commented a month ago Master

> From where can the wrapper extract `UserClaim` and `AccountClaim`?

I added `user_id`, `account_id`, etc. from claims to context !798 (merged). Feels that it's too implicit and unreliable. We may add something like Logger.Check() to turn on/off individual log line based on `user_id`, `account_id`.

> And how do we know which `User`/`Account` is enabled for debug logs?

You can implement something like l.go and logctl.go.

- Add config to `Logger` which contains `UserID` and `AccountID` maps.
- Update `logctl.go` to be able to turn on/off individual `UserID`, `AccountID`.
- `ll.Check()` acts like Logger.Check() or automatically extracts `user_id`, `account_id` from `context.Context`.

Vu Nguyen @ng-vu commented a month ago Master

With `l.Wrap`, we don't need to add log to each error returned:

```
if err != nil {
  ll.Error("Something does not work", l.Error(err), l
  return nil, err
}
```

We can change the pattern to:

```
if err != nil {
  return nil, l.Wrap(err, "Something does not work",
}
```

And let the middleware print all request/response, related information and trace all `return` statements.

Hiep Nguyen @hiepnv commented a month ago Developer

@ng-vu Signature of `Logger.Check()` does not support to pass `user_id`, `account_id` to it?

Hiep Nguyen @hiepnv commented a month ago Developer

@ng-vu The important thing is we need to overwrite behavior of `zap.Logger.Check()`, but it seems we can't do that in `common/l` package.

Vu Nguyen @ng-vu commented a month ago Master

Could you also upgrade `zap` to the latest version? The latest version has so many breaking changes (doesn't compatible with our `l` package).

Hiep Nguyen @hiepnv commented a month ago Developer

Yes, should I upgrade it in new branch?

Edited a month ago by Hiep Nguyen

Hiep Nguyen @hiepnv commented a month ago Developer

Update `zap` package to latest version !836 (merged)

Hiep Nguyen @hiepnv commented a month ago Developer

@ng-vu Will I modify this function for `Add a grpc middleware`?

Vu Nguyen @ng-vu commented a month ago Master

Yes, I think so. Print `Error` trace if we encounter an error.

Hiep Nguyen @hiepnv commented a month ago Developer

@ng-vu grpc logging middleware only print `Info` log and `Error` log, it seems we don't need enabled log for specific user/account feature. If we want to use that feature, we need to pass a context that contains userID or accountID. So we need additional step to print `Debug` logs...

Vu Nguyen @ng-vu commented a month ago Master

I think it should check for the context with `userID` or `accountID`.

If the `userID` / `accountID` has debugging enabled, the

Vu Nguyen @ng-vu commented a month ago Master

> Also, I think append log info to `ctx` is not a good idea. That sounds like `ctx` is a super container, it contains everything.

`ctx` will be discarded at the end of the request. Don't worry.

I suggest an implementation like this:

```
package l // l.go

type keyLog struct{}

type LogTrace {
  Stack []LogItem  // LogItem struct{ message, file, l
}

func NewLogContext(ctx context.Context) (context.Conte
  return context.WithValue(keyLog{}, ctx)
}

func (ll Logger) Debug(ctx, ...) {
  logTrace, ok := ctx.Value(keyLog{}).(*LogTrace)
  if !ok {
     // Unexpected. Print stacktrace.
     return
  }

  logTrace.Stack = append(logTrace.Stack, LogItem{...}
}
```

Hiep Nguyen @hiepnv commented a month ago Developer

I mean, how do we pass `ctx` to `ll.Debug` when we don't have it?

Hiep Nguyen @hiepnv commented a month ago Developer

For example, we could not pass `ctx` to `ledger.models` functions just for logging.

Vu Nguyen @ng-vu commented a month ago Master

We should update `ll.Debug()` to accept `ctx` as the first param.

For `ledger.models`, we can ignore it. It can fallback to stdout log (we'll lose userID information for debugging).

In case of returned error, `l.Wrap()` is more reliable. It doesn't require `ctx`.

Hiep Nguyen @hiepnv commented a month ago Developer

@ng-vu

I modified logging function to receive `ctx` as first argument
I think the main purpose of enable logs for particular user/account is in debug mod, we want to view more debug logs for that user/account. If we ignore at `models` package then I think it will be less efficient. Also other services do not use `grpc` do not have

Problems with Go error handling

- Error() is for human, not machine
- How to inspect the returned error value to decide what should we do?
- Where the error occur?
  How to trace back?
- How to translate error value to response code?
- How to log the values from different functions in a session?

```
result, err := sendNotification(tokens)
if err != nil {
  log.Write(“Error %v, Tokens: %v”, err, tokens)
  return err
}
```

What are people doing?

I downloaded top 100 Go repositories on GitHub to find out.

Pattern of error handling

- `errors.New()`
  `fmt.Errorf()`
- `if err != nil`
- `switch err := err.(type)`
- `switch err.Code`
  `switch errors.Code(err)`

-
  ```
  switch {
    case err.(FooInterface):
  ```
- `err.Stack()`
- `allErrors.Append(err)`
  `allErrors.AggrError()`
- `status.Error`

Which error package people are using?

- Error package and number of repos / top 100 repos

“errors”	85
“github.com/pkg/errors”	17
“github.com/go-errors/errors”	2
“github.com/juju/errors”	1
Custom packages	25

Pattern of error handling

- `errors.New()`
  `fmt.Errorf()`
- `if err != nil`
- `switch err := err.(type)`
- `switch err.Code`
  `switch errors.Code(err)`
-
  ```
  switch {
    case err.(FooInterface):
  ```
- `err.Stack()`
- `allErrors.Append(err)`
  `allErrors.AggrError()`
- `status.Error`

github.com/pkg/errors

- Imported by 17 / top 100 repos

```
_, err := ioutil.ReadAll(r)
if err != nil {
        return errors.Wrap(err, "read failed")
}

// output
read failed: open not_found.txt: no such file or
directory

// trace
read failed
main.loadFile
        /Users/i/go/src/sample/errors.go:13
main.main
        /Users/i/go/src/sample/errors.go:23
```

apex/apex	kubernetes/kubernetes
cloudson/gitql	kubernetes/minikube
cockroachdb/cockroach	moby/moby
containous/traefik	ncw/rclone
elastic/beats	simeji/jid
git-lfs/git-lfs	weaveworks/weave
golang/dep	xtaci/kcptun
hashicorp/consul	yudai/gotty
hashicorp/vault	

github.com/go-errors/errors

- Imported by 2 / top 100 repos

hashicorp/vault

zyedidia/micro

```
_, err := ioutil.ReadAll(r)
if err != nil {
        return errors.Wrap(err, 0)
}

// output
open not_found.txt: no such file or directory

// stack
*os.PathError open not_found.txt: no such file or directory
/Users/i/go/src/sample/errors.go:13 (0x109be06)
        loadFile: return errors.Wrap(err, 0)
/Users/i/go/src/sample/errors.go:27 (0x109be76)
        main: err := loadFile()
```

github.com/juju/errors

- Imported by 1 / top 100 repos

pingcap/tidb

```
_, err := ioutil.ReadAll(r)
if err != nil {
        return errors.Annotate(err, "read failed")
}

// output
read failed: open not_found.txt: no such file or directory

// trace
open not_found.txt: no such file or directory
sample/etc/errors.go:13: read failed
```

Custom error packages

- 25 / top 100 repos use custom packages
- Wrap error context
- Aggregate multiple errors
- Custom status code
- Interface error

Middleware for handling error (1)

- Translate error value to response code
- Write logs

```
func(ctx context.Context, req interface{}, info *grpc.UnaryServerInfo, handler grpc.UnaryHandler) (resp interface{}, err error)
{
    defer func() {
      e := recover()
      if e != nil {
        logger.Error("Panic (Recovered)", l.Error(err), l.Stack("stacktrace"))
        err = grpc.Errorf(codes.Internal, "Internal Error (%v)", e)
      }
      if err == nil {
        logger.Info(info.FullMethod, l.Interface("\n→", req), l.Interface("\n⇐", resp))
        return
      }
      logger.Error(info.FullMethod, l.Interface("\n→", req), l.String("\n⇐ERROR", err.Error()))
    }()
    resp, err = handler(ctx, req)
    err = translateError(err)
  }
```

Middleware for handling error (2)

- Translate error value to response code
- Write logs

```go
func translateError(err error) int {
    switch err.(type) {
    case NotFound:
        return 404
    // ...
    }

    switch codes.Code(err) {
    case CodeNotFound:
        return 404
    // ...
    }

    return InternalError
}
```

moby/moby

- Custom package: “github.com/docker/docker/api/errdefs”
- Rely on error interface and GRPC error code

```
func GetHTTPErrorStatusCode(err error) int


type ErrNotFound interface {
    NotFound()
}


switch {
    case errdefs.IsNotFound(err):
        statusCode = http.StatusNotFound
    // ...
}
```

Aggregate multiple errors

- Track errors from multiple source
- Translate to single error to return / respond

```go
type AllErrorAggregator func(errors []error) error


func (aer *AllErrorRecorder) AggrError(aggr AllErrorAggregator) error {
    aer.mu.Lock()
    defer aer.mu.Unlock()
    if len(aer.Errors) == 0 {
        return nil
    }
    return aggr(aer.Errors)
}
```

Patterns of error handling

- Wrap error context
- Aggregate multiple errors
- Custom status code
- Interface error
- Middleware for translating error code

Conclusion

Pattern for error handling

Top error handling methods

- errors.New() for simple case
- errors.Wrap() for tracing error and writing logs
- Custom error package for translating error