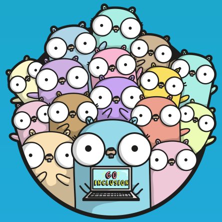
Go 1.13 Release Party, Aug 22 2019

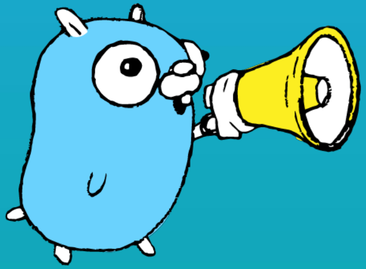
The Short History of Error Proposals in Go 2



권민재

GDG Golang KR

Speaker



목차

The Problems of Go’s Error Handling 01

Go 2 Draft Design and Proposals for Errors 02

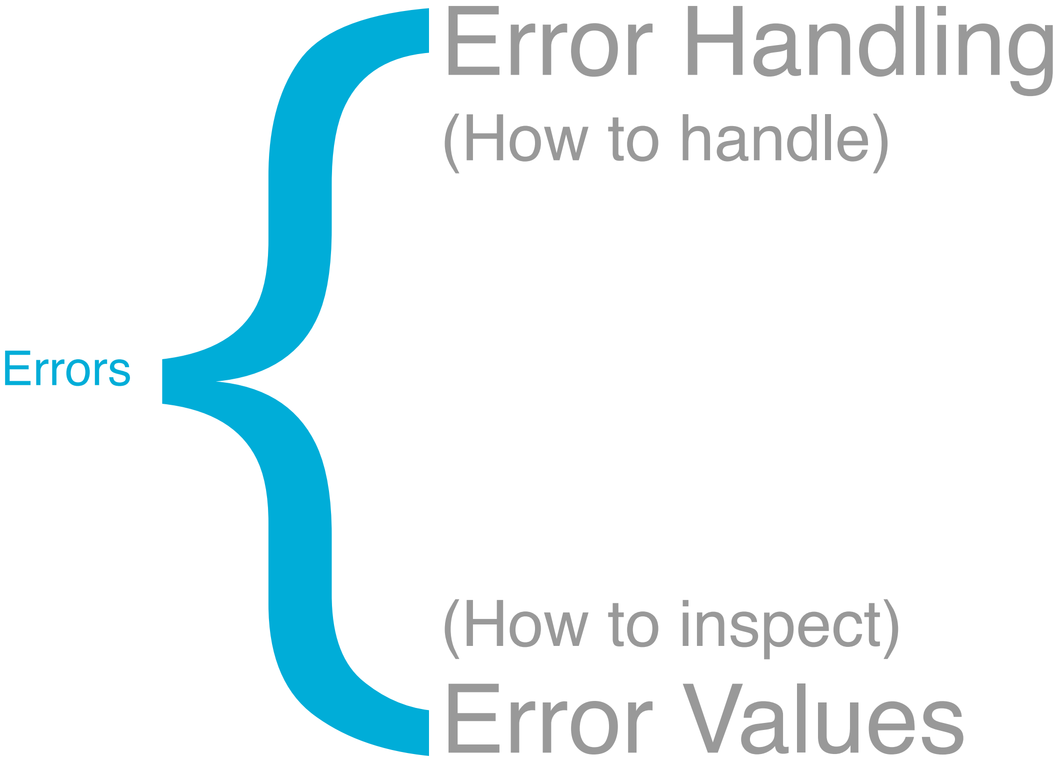
Error Values in Go1.13 (Accepted) 03

Conclusion 04

SECTION 1

The Problems of Go’s Error Handling

How to handle the errors in Go?



Errors

Error Handling

(How to handle)

(How to inspect)

Error Values

How to handle the errors in Go?

2 Ways to handle the errors

- Check and return errors.
- Panic, recover and return errors.

4 Ways to inspect the errors

- Sentinel errors.
- Type assertion or type switch.
- Ad-hoc function.
- Substring search.

Error Handling

Check and return error values

```go
func CopyFile(src, dst string) error {
	r, err := os.Open(src)
	if err != nil {
		return err
	}
	defer r.Close()

	w, err := os.Create(dst)
	if err != nil {
		return err
	}
	defer w.Close()

	if _, err := io.Copy(w, r); err != nil {
		return err
	}
	if err := w.Close(); err != nil {
		return err
	}
}
```

1. **Check** error condition.
2. Return error values.
3. **Check** error is set.
4. Return error values.
5. …
6. Handle the errors.

Error Handling

Check and return error values

```go
func CopyFile(src, dst string) error {
    r, err := os.Open(src)
    if err != nil {
        return fmt.Errorf("copy %s %s: %v", src, dst, err)
    }
    defer r.Close()

    w, err := os.Create(dst)
    if err != nil {
        return fmt.Errorf("copy %s %s: %v", src, dst, err)
    }
    defer w.Close()

    if _, err := io.Copy(w, r); err != nil {
        return fmt.Errorf("copy %s %s: %v", src, dst, err)
    }
    if err := w.Close(); err != nil {
        return fmt.Errorf("copy %s %s: %v", src, dst, err)
    }
}
```

1. **Check** error condition.
2. Return error values.
3. **Check** error is set.
4. Return error values.
5. …
6. Handle the errors.

Error Handling

Panic, recover and return error values

```
func Div(a, b int64) (q int64, err error) {
      defer func() {
            if r := recover(); r != nil {
                  q, err = 0, r.(error)
            }
      }()
      return a / b, nil
}

func Calc(a, b int64) (int64, error) {
      div, err := Div(a, b)
      if err != nil {
            return 0, err
      }
      return div, nil
}
```

1. **Expect** the panics.
2. Prepare the recover.
3. Handle the panics.

Error Values

Sentinel error

```go
var EOF = errors.New(“EOF")

...

func (b *Buffer) ReadFrom(r io.Reader) (n int64, err error) {
    b.lastRead = opInvalid
    for {
        ...

        b.buf = b.buf[:i+m]
        n += int64(m)
        if e == io.EOF {
            return n, nil
        }
        if e != nil {
            return n, e
        }
    }
}
```

Errors are values in Go. So,

We can test for **equality** with **sentinel errors** like `io.EOF`

Error Values

Type assertion

```
type Error interface {
       error
       Code() string
       Message() string
       OrigErr() error
}

...

output, err := s3manage.Upload(svc, input, opts)
if err != nil {
    if awsErr, ok := err.(awserr.Error); ok {
        fmt.Println(awsErr.Code(), awsErr.Message(), awsErr.OrigErr())

        if reqErr, ok := err.(awserr.RequestFailure); ok {
            fmt.Println(reqErr.StatusCode(), reqErr.RequestID())
        }
    } else {
        fmt.Println(err.Error())
    }
}
```

Error interface allows developers to extract the information from errors.

It can be done by **wrapping** the original errors.

You can inspect the details using **type assertion.**

Error Values

Ad-hoc check

```
type PathError struct {
        Op   string
        Path string
        Err  error
}

...

func isNotExist(err error) bool {
        return checkErrMessageContent(err, "does not exist", "not found”, ...)
}

...

func underlyingError(err error) error {
        switch err := err.(type) {
        case *PathError:
                return err.Err
        case *LinkError:
                return err.Err
        case *SyscallError:
                return err.Err
        }
        return err
}

...

func main() {
        filename := "a-nonexistent-file"
        if _, err := os.Stat(filename); os.IsNotExist(err) {
                fmt.Println("file does not exist")
        }
}
```

Check for a **specific kind** of error, doing limited **unwrapping**.

Mixed of unwrapping (using **type switch**) and **substring search**.

Error Values

Substring search

```
func Query(query string) (string, error) {
	db, err := sql.Open("dsn")
	if err != nil {
		return "", fmt.Errorf("db error: %v", err)
	}
	rows, err := db.Query(query)
	if err != nil {
		return "", fmt.Errorf(“db error: %v", err)
	}
	// Process the rows.
}

func main() {
	res, err := Query("SELECT * FROM users LIMIT 10")
	if err != nil {
		if strings.Contains(err.Error(), "connect") {
			log.Fatalln("could not connect to database")
		}
		if strings.Contains(err.Error(), "query") {
			log.Fatalln("could not complete the query")
		}
	}
}
```

Search the specific **substring** in the error text reported by **error.Error()**.

There may no **worse** way than this approach.

The problems of current error handling

- There could be too many error checking **boilerplate** codes.
- Propagate the error context is **not easy**.
- Panics are **not always handleable** and hard to **expect**.
- Panic handling **!=** Error handling.

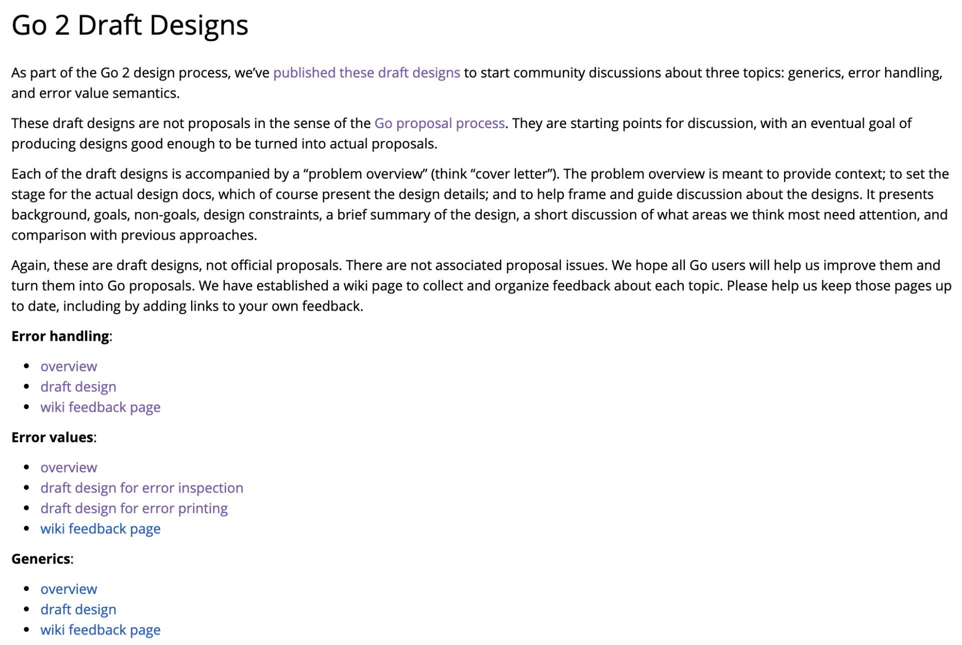
The problems of current error values

- Single sentinel error has **no additional context or information**.
- Ad-hoc checks **lacks generality** and understands only **a very limited number of wrapping** types.
- Substring search is **not programmatic** way for error inspection.
- Go has to **preserve** and **keep** the **error context** for tracing the error stacks.

SECTION 2

Go 2 Draft Design and Proposals for Errors

Go 2 Draft Design



Go 2 Draft Designs

As part of the Go 2 design process, we've published these draft designs to start community discussions about three topics: generics, error handling, and error value semantics.

These draft designs are not proposals in the sense of the Go proposal process. They are starting points for discussion, with an eventual goal of producing designs good enough to be turned into actual proposals.

Each of the draft designs is accompanied by a “problem overview” (think “cover letter”). The problem overview is meant to provide context; to set the stage for the actual design docs, which of course present the design details; and to help frame and guide discussion about the designs. It presents background, goals, non-goals, design constraints, a brief summary of the design, a short discussion of what areas we think most need attention, and comparison with previous approaches.

Again, these are draft designs, not official proposals. There are not associated proposal issues. We hope all Go users will help us improve them and turn them into Go proposals. We have established a wiki page to collect and organize feedback about each topic. Please help us keep those pages up to date, including by adding links to your own feedback.

Error handling:

- overview
- draft design
- wiki feedback page

Error values:

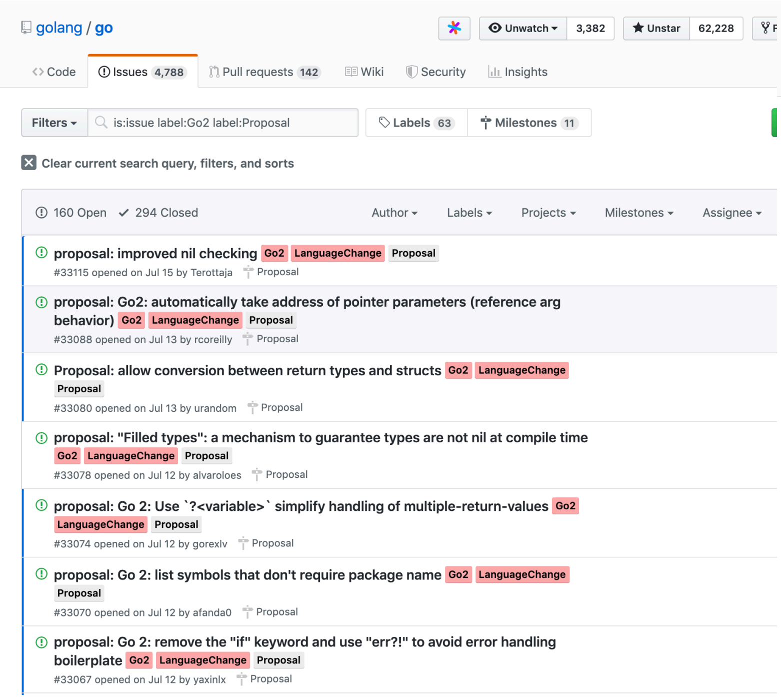
- overview
- draft design for error inspection
- draft design for error printing
- wiki feedback page

Generics:

- overview
- draft design
- wiki feedback page

Draft designs are not proposals in the sense of the Go proposal process

Go 2 Proposals



golang / go

Unwatch 3,382 | Unstar 62,228

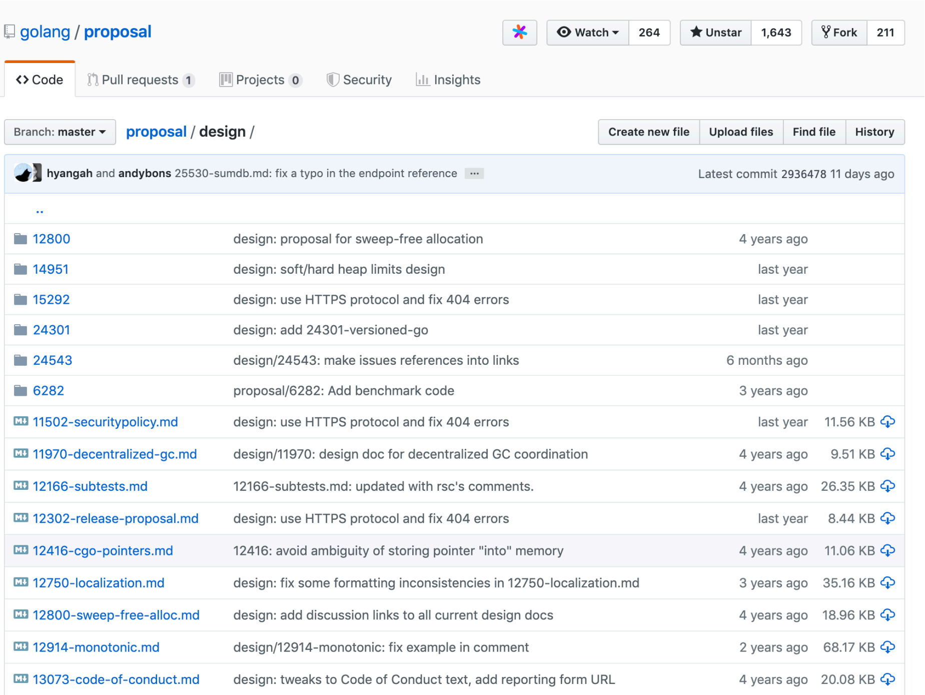
Code | Issues 4,788 | Pull requests 142 | Wiki | Security | Insights

Filters | is:issue label:Go2 label:Proposal | Labels 63 | Milestones 11

Clear current search query, filters, and sorts

160 Open ✓ 294 Closed | Author | Labels | Projects | Milestones | Assignee

- proposal: improved nil checking Go2 LanguageChange Proposal
  #33115 opened on Jul 15 by Terottaja Proposal
- proposal: Go2: automatically take address of pointer parameters (reference arg behavior) Go2 LanguageChange Proposal
  #33088 opened on Jul 13 by rcoreilly Proposal
- Proposal: allow conversion between return types and structs Go2 LanguageChange Proposal
  #33080 opened on Jul 13 by urandom Proposal
- proposal: "Filled types": a mechanism to guarantee types are not nil at compile time Go2 LanguageChange Proposal
  #33078 opened on Jul 12 by alvaroloes Proposal
- proposal: Go 2: Use `?<variable>` simplify handling of multiple-return-values Go2 LanguageChange Proposal
  #33074 opened on Jul 12 by gorexlv Proposal
- proposal: Go 2: list symbols that don't require package name Go2 LanguageChange Proposal
  #33070 opened on Jul 12 by afanda0 Proposal
- proposal: Go 2: remove the "if" keyword and use "err?!" to avoid error handling boilerplate Go2 LanguageChange Proposal
  #33067 opened on Jul 12 by yaxinlx Proposal



golang / proposal

Watch 264 | Unstar 1,643 | Fork 211

Code | Pull requests 1 | Projects 0 | Security | Insights

Branch: master | proposal / design / | Create new file | Upload files | Find file | History

hyangah and andybons 25530-sumdb.md: fix a typo in the endpoint reference … Latest commit 2936478 11 days ago

..			
12800	design: proposal for sweep-free allocation	4 years ago	
14951	design: soft/hard heap limits design	last year	
15292	design: use HTTPS protocol and fix 404 errors	last year	
24301	design: add 24301-versioned-go	last year	
24543	design/24543: make issues references into links	6 months ago	
6282	proposal/6282: Add benchmark code	3 years ago	
11502-securitypolicy.md	design: use HTTPS protocol and fix 404 errors	last year	11.56 KB
11970-decentralized-gc.md	design/11970: design doc for decentralized GC coordination	4 years ago	9.51 KB
12166-subtests.md	12166-subtests.md: updated with rsc's comments.	4 years ago	26.35 KB
12302-release-proposal.md	design: use HTTPS protocol and fix 404 errors	last year	8.44 KB
12416-cgo-pointers.md	12416: avoid ambiguity of storing pointer "into" memory	4 years ago	11.06 KB
12750-localization.md	design: fix some formatting inconsistencies in 12750-localization.md	3 years ago	35.16 KB
12800-sweep-free-alloc.md	design: add discussion links to all current design docs	4 years ago	18.96 KB
12914-monotonic.md	design/12914-monotonic: fix example in comment	2 years ago	68.17 KB
13073-code-of-conduct.md	design: tweaks to Code of Conduct text, add reporting form URL	4 years ago	20.08 KB

Discussed Draft Design and Proposals

Error Handling

- check and handle
- try
- if err != nil (!?)

Error Values

- Error inspection
  - Unwrap, Is, As
- Error formatting
- Error stack

Go 2 Error Handling

Goals

- **Lightweight** error checking by reducing the boilerplate code
- More **convenient** write to error handling

Go 2 Error Handling

Design

- check and handle
- try
- if err != nil

Go 2 Error Handling

check and handle (draft design) by Marcel van Lohuizen (August 27, 2018)

New keywords ”**check**” and “**handle"**

Similar to “panic" and "recover"

https://go.googlesource.com/proposal/+/master/design/go2draft-error-handling.md

Go 2 Error Handling

check and handle / Background

There have been many proposals over time to improve error handling in Go. For instance, see:

- golang.org/issue/21161: simplify error handling with `|| err` suffix
- golang.org/issue/18721: add “must” operator `#` to check and return error
- golang.org/issue/16225: add functionality to remove repetitive `if err != nil` return
- golang.org/issue/21182: reduce noise in return statements that contain mostly zero values
- golang.org/issue/19727: add vet check for test of wrong `err` variable
- golang.org/issue/19642: define `_` on right-hand side of assignment as zero value
- golang.org/issue/19991: add built-in result type, like Rust, OCaml
- …

Go 2 Error Handling

check and handle

```go
func CopyFile(src, dst string) error {
	r, err := os.Open(src)
	if err != nil {
		return err
	}
	defer r.Close()

	w, err := os.Create(dst)
	if err != nil {
		return err
	}
	defer w.Close()

	if _, err := io.Copy(w, r); err != nil {
		return err
	}
	if err := w.Close(); err != nil {
		return err
	}
}
```

It **does not remove** `dst`
when `io.Copy` or `w.Close` fails.

Go 2 Error Handling

check and handle

```go
func CopyFile(src, dst string) error {
        r, err := os.Open(src)
        if err != nil {
                return err
        }
        defer r.Close()

        w, err := os.Create(dst)
        if err != nil {
                return err
        }
        defer w.Close()

        if _, err := io.Copy(w, r); err != nil {
                return err
        }
        if err := w.Close(); err != nil {
                return err
        }
}
```

```go
func CopyFile(src, dst string) error {
        r, err := os.Open(src)
        if err != nil {
                return err
        }
        defer r.Close()

        w, err := os.Create(dst)
        if err != nil {
                return err
        }

        if _, err := io.Copy(w, r); err != nil {
                w.Close()
                os.Remove(dst)
                return err
        }

        if err := w.Close(); err != nil {
                os.Remove(dst)
                return err
        }
}
```

Go 2 Error Handling

check and handle

```
func CopyFile(src, dst string) error {
        r, err := os.Open(src)
        if err != nil {
                return err
        }
        defer r.Close()

        w, err := os.Create(dst)
        if err != nil {
                return err
        }

        if _, err := io.Copy(w, r); err != nil {
                w.Close()
                os.Remove(dst)
                return err
        }

        if err := w.Close(); err != nil {
                os.Remove(dst)
                return err
        }
}
```

```
func CopyFile(src, dst string) error {
        handle err {
                return err
        }

        r := check os.Open(src)
        defer r.Close()

        w := check os.Create(dst)
        handle err {
                w.Close()
                os.Remove(dst)
        }

        check io.Copy(w, r)
        check w.Close()
        return nil
}
```

Go 2 Error Handling

check and handle / check

```
v1, ..., vN := check <expr>

// is equivalent to

v1, ..., vN, vErr := <expr>
if vErr != nil {
      <error result> = handlerChain(vn)
      return
}

...

func handleChain(err error) error {
      return err
}
```

If error is **not nil**, check call **handlerChain** implicitly.

It is likely to be implemented **differently** inside the Go compiler.

Go 2 Error Handling

check and handle / check

```
func printSum(a, b string) error {
        x, err := strconv.Atoi(a)
        if err != nil {
                return fmt.Errorf("printSum(%q + %q): %v", a, b, err)
        }
        y, err := strconv.Atoi(b)
        if err != nil {
                return fmt.Errorf("printSum(%q + %q): %v", a, b, err)
        }
        fmt.Println("result:", x+y)
        return nil
}
```

Repeated error checking code snippet.

How to fix with check and handle?

Go 2 Error Handling

check and handle / check

```go
func printSum(a, b string) error {
	handle err {
		return fmt.Errorf("printSum(%q + %q): %v", a, b, err)
	}
	x := check strconv.Atoi(a)
	y := check strconv.Atoi(b)
	fmt.Println("result:", x + y)
	return nil
}
```

We could rewrite like this.

For each check, there is an implicit **handler chain function**.

Go 2 Error Handling

check and handle / check

```
func printSum(a, b string) error {
      handle err {
            return fmt.Errorf("printSum(%q + %q): %v", a, b, err)
      }
      fmt.Println("result:", check strconv.Atoi(x) + check
strconv.Atoi(y))
      return nil
}
```

Since a `check` is an **expression**, we could write like this.

Go 2 Error Handling

check and handle / handle

```
Statement   = Declaration | ...  | DeferStmt | HandleStmt .
HandleStmt  = "handle" identifier Block .

...

func main() {
      handle err {
            log.Fatal(err)
      }

      hex := check ioutil.ReadAll(os.Stdin)
      data := check parseHexdump(string(hex))
      os.Stdout.Write(data)
}
```

The `handle` statement defines a block, called a ***handler***, to **handle** an error detected **by** a `check`.

A *handler chain function* takes an argument of type **`error`**.

There is **no** way to resume control in the enclosing function **after `check` detects** an error.

panic in handle? just panic as if it occurred in function.

Go 2 Error Handling

check and handle / handle

```
func process(user string, files chan string) (n int, err error)
{
    handle err { return 0, fmt.Errorf("process: %v", err)  }
// handler A
    for i := 0; i < 3; i++ {
        handle err { err = fmt.Errorf("attempt %d: %v", i,
err) } // handler B
        handle err { err = moreWrapping(err) }
// handler C

        check do(something())  // check 1: handler chain C, B, A
    }
    check do(somethingElse())  // check 2: handler chain A
}
```

return in handle? cause the enclosing function to return.

It **executes all** handlers in lexical **scope** in **reverse order** of declaration **until** one of them executes a **`return`** statement.

If the enclosing function has **result parameters**, it is a **compile-time error** if the **handler** chain for any check is **not guaranteed to execute a `return`** statement.

Any handler **always** executes **before** any **deferred** functions are executed.

Go 2 Error Handling

check and handle / examples

```
type Error struct {
        Func string
        User string
        Path string
        Err  error
}

func (e *Error) Error() string

func ProcessFiles(user string, files chan string) error {
        e := Error{ Func: "ProcessFile", User: user}
        handle err { e.Err = err; return &e } // handler A
        u := check OpenUserInfo(user)         // check 1
        defer u.Close()
        for file := range files {
                handle err { e.Path = file }       // handler B
                check process(check os.Open(file)) // check 2
        }
        ...
}
```

Add context information to the error with handle.

Second handle will be executed **exactly once only** when **second** check fails.

Go 2 Error Handling

check and handle / drawbacks and limits

```
// Compile error!
func Greet(w io.WriteCloser) error {
      defer func() {
            check w.Close()
      }()
      fmt.Fprintf(w, "hello, world\n")
      return nil
}

// This code has an ordering problem.
func Greet(w io.WriteCloser) error {
      defer check w.Close()
      fmt.Fprintf(w, "hello, world\n")
      return nil
}
```

Context-dependent control-flow **jump**. (break, continue, defer, handle…)

It does **not** provide a mechanism for checking errors returned by **deferred** calls.

Checking error returns from deferred calls is **not easy.**

Go 2 Error Handling

try (proposal) by Robert Griesemer (Jun 5, 2019)

New keyword (or built-in func) **”try”**

It could be included in Go 1.14 later, but..

https://github.com/golang/proposal/blob/master/design/32437-try-builtin.md

Go 2 Error Handling

try

```
func try(expr) (T1, T2, ... Tn)

...

x1, x2, ... xn = try(f())

...

t1, ... tn, te := f()  // t1, ... tn, te are local (invisible)
temporaries
if te != nil {
        err = te     // assign te to the error result parameter
        return       // return from enclosing function
}
x1, ... xn = t1, ... tn  // assignment only if there was no error
```

Invoking `try` with a function call `f()` as in (pseudo-code) **turns** into the in-lined code

Go 2 Error Handling

try

```go
f, err := os.Open(filename)
if err != nil {
        return ..., err  // zero values for other results, if any
}

...

f := try(os.Open(filename))
```

Above code can be simplified to below code with try.

How to wrap the errors?

Go 2 Error Handling

try

```go
func CopyFile(src, dst string) (err error) {
        defer func() {
                if err != nil {
                        err = fmt.Errorf("copy %s %s: %v", src, dst, err)
                }
        }()

        r := try(os.Open(src))
        defer r.Close()

        w := try(os.Create(dst))
        defer func() {
                w.Close()
                if err != nil {
                        os.Remove(dst) // only if a “try” fails
                }
        }()

        try(io.Copy(w, r))
        try(w.Close())
        return nil
}
```

Wrap the named error in the **deferred** function.

Go 2 Error Handling

try

```go
func HandleErrorf(err *error, format string, args ...interface{}) {
	if *err != nil {
		*err = fmt.Errorf(format+": %v", append(args, *err)...)
	}
}

func CopyFile(src, dst string) (err error) {
	defer fmt.HandleErrorf(&err, "copy %s %s", src, dst)

	r := try(os.Open(src))
	defer r.Close()

	w := try(os.Create(dst))
	defer func() {
		w.Close()
		if err != nil {
			os.Remove(dst) // only if a “try" fails
		}
	}()

	try(io.Copy(w, r))
	try(w.Close())
	return nil
}
```

Or use the **helper** function and **deferred** function.

Go 2 Error Handling

try / summary

- There is **no** interference with the rest of the language.
- Because it is syntactic sugar, `try` is **easily explained** in more basic terms of the language.
- The design does not require new syntax.
- The design is fully **backwards-compatible**.

Go 2 Error Handling

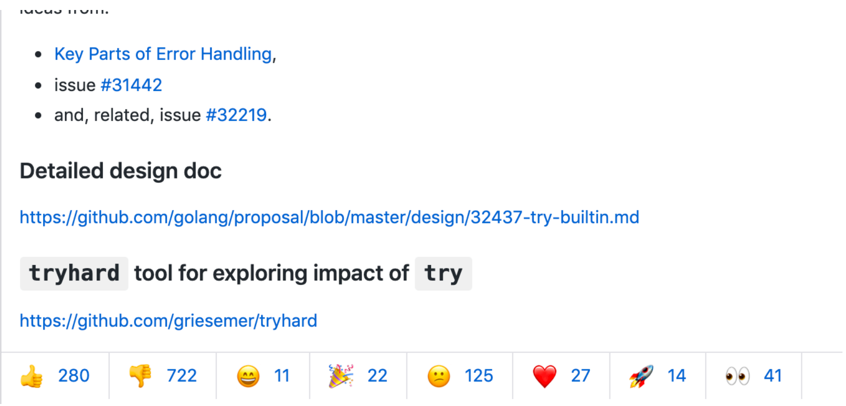
try

But this proposal was **declined** at Jul 17, 2019

Why?

Go 2 Error Handling

try / decline



ideas from:

- Key Parts of Error Handling,
- issue #31442
- and, related, issue #32219.

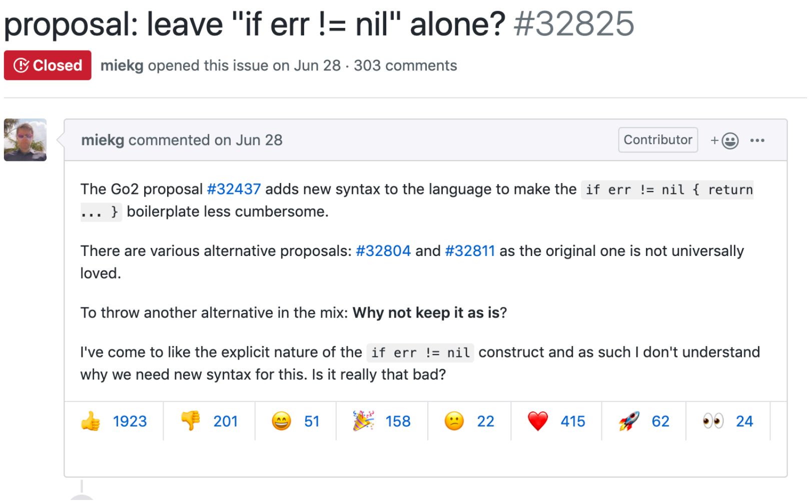
Detailed design doc

https://github.com/golang/proposal/blob/master/design/32437-try-builtin.md

`tryhard` tool for exploring impact of `try`

https://github.com/griesemer/tryhard

👍 280 👎 722 😄 11 🎉 22 😕 125 ❤️ 27 🚀 14 👀 41



proposal: leave "if err != nil" alone? #32825

miekg opened this issue on Jun 28 · 303 comments

miekg commented on Jun 28

The Go2 proposal #32437 adds new syntax to the language to make the `if err != nil { return ... }` boilerplate less cumbersome.

There are various alternative proposals: #32804 and #32811 as the original one is not universally loved.

To throw another alternative in the mix: **Why not keep it as is**?

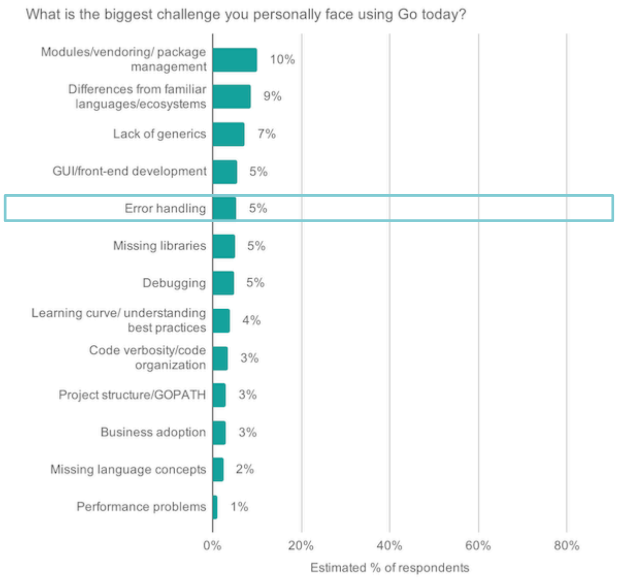
I've come to like the explicit nature of the `if err != nil` construct and as such I don't understand why we need new syntax for this. Is it really that bad?

👍 1923 👎 201 😄 51 🎉 158 😕 22 ❤️ 415 🚀 62 👀 24

Proposal: leave "if err != nil" alone?

Go 2 Error Handling

try / decline



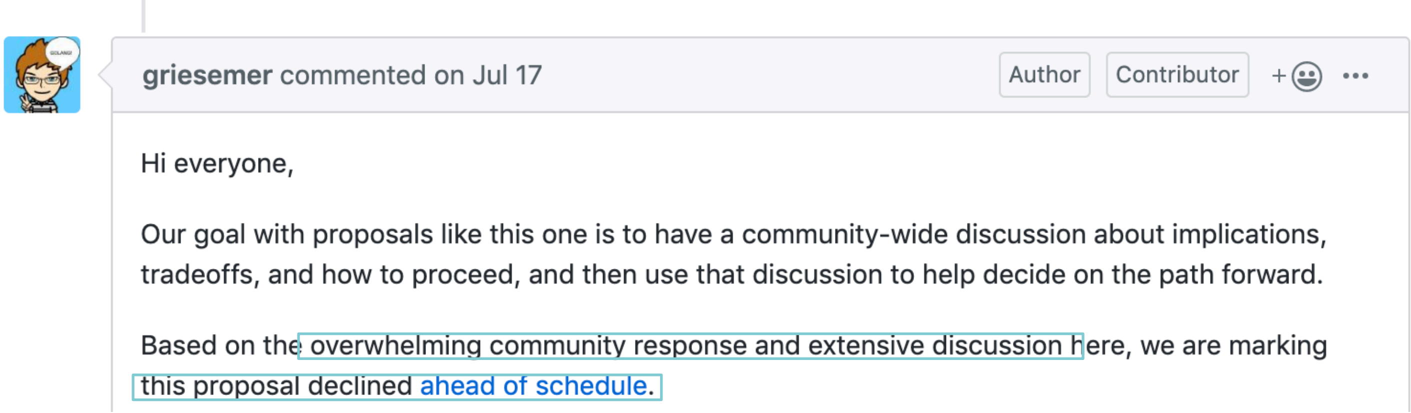
It introduces **two ways** to do the **same thing** as it relates to the simple case when an error will **only be propagated back up** to the caller

This new mechanic is going to cause severe **inconsistencies** in code bases

Open Letter To The Go Team About Try

Go 2 Error Handling

try / decline



griesemer commented on Jul 17

Hi everyone,

Our goal with proposals like this one is to have a community-wide discussion about implications, tradeoffs, and how to proceed, and then use that discussion to help decide on the path forward.

Based on the overwhelming community response and extensive discussion here, we are marking this proposal declined ahead of schedule.

Go 2 Error Values

Goals

- Make error inspection by programs **easier** and **less error-prone.**
- Programs can treat errors from different packages **uniformly**.
- Make it possible to print errors with **additional detail**, in a **standard** form.

Go 2 Error Handling

Design / by Jonathan Amsterdam and 3 others (Jan 25, 2019)

- Wrapping
  - Unwrap, Is, As
- Formatting
- Stack Frames

https://github.com/golang/go/issues/29934

Go 2 Error Values

Error Inspection

```
write users database: call myserver.Method: \
    dial myserver:3333: open /etc/resolv.conf: permission denied

...

write users database
call myserver.Method
dial myserver:3333
open /etc/resolv.conf
permission denied
```

Here is a complex example.

How to **make** and **inspect** this chained errors?

We should **wrap** the each errors for different operations.

Go 2 Error Handling

Error Inspection

1. A `WriteError`, which provides `"write users database: "` and wraps
2. an `RPCError`, which provides `"call myserver.Method: "` and wraps
3. a `net.OpError`, which provides `"dial myserver:3333: "` and wraps
4. an `os.PathError`, which provides `"open /etc/resolv.conf: "` and wraps
5. `syscall.EPERM`, which provides `"permission denied"`

1. Is it an **RPCError**?
2. Is it a **net.OpError**?
3. Does it satisfy the **net.Error interface**?
4. Is it an **os.PathError**?
5. Is it a **permission error**?

Go 2 Error Values

Error Inspection / Unwrapping

```go
// Wrapping using fmt.Errorf
if err != nil {
	return fmt.Errorf("write users database: %v", err)
}

// Wrapping with new type
if err != nil {
	return &WriteError{Database: "users", Err: err}
}
```

Easy to equality checks? Easy to type assertion?

How to unwrap? How to check specific error type?

Go 2 Error Values

Error Inspection / Wrapper interface

```go
// A Wrapper is an error implementation
// wrapping context around another error.
type Wrapper interface {
	// Unwrap returns the next error in the error chain.
	// If there is no next error, Unwrap returns nil.
	Unwrap() error
}
```

Program can inspect the **chain of wrapped** errors by using a type assertion to check for the **`Unwrap`**

How to apply it?

Go 2 Error Values

Error Inspection / Unwrap function

```go
// Unwrap returns the result of calling the Unwrap method on
err, if err implements Unwrap.
// Otherwise, Unwrap returns nil.
func Unwrap(err error) error
```

The `Unwrap` function is a **convenience** for calling the `Unwrap` method if one exists.

There is **no need** to write Wrapper interface **explicitly**.

Go 2 Error Values

Error Inspection / Is function

```go
// Is reports whether any error in err's chain matches target.
//
// An error is considered to match a target if it is equal to
that target or if
// it implements a method Is(error) bool such that Is(target)
returns true.
func Is(err, target error) bool

// instead of err == io.ErrUnexpectedEOF
if errors.Is(err, io.ErrUnexpectedEOF) { ... }
```

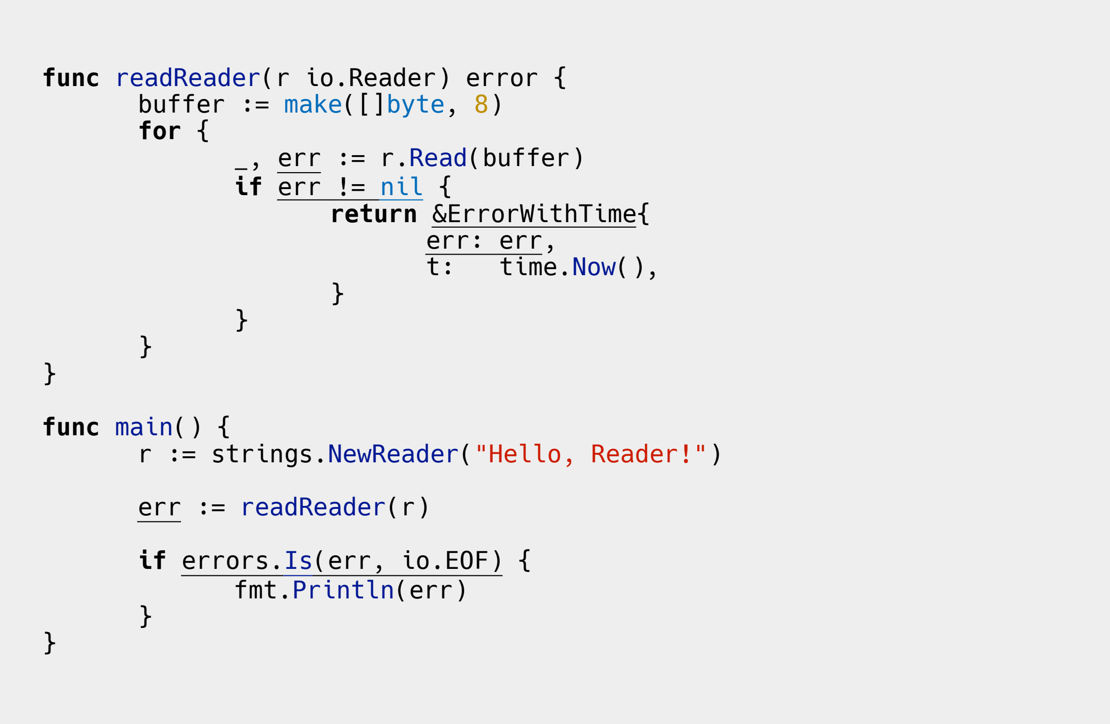
“Is” uses **equality** check for **sentinel** errors.

It can check the **chained** errors thanks to **Unwrap()**.

Go 2 Error Values

Error Inspection / Is function

example from here



```go
func readReader(r io.Reader) error {
        buffer := make([]byte, 8)
        for {
                _, err := r.Read(buffer)
                if err != nil {
                        return &ErrorWithTime{
                                err: err,
                                t:   time.Now(),
                        }
                }
        }
}

func main() {
        r := strings.NewReader("Hello, Reader!")

        err := readReader(r)

        if errors.Is(err, io.EOF) {
                fmt.Println(err)
        }
}
```

Check if the error is "io.EOF " using **errors.Is**.

Go 2 Error Values

Error Inspection / As function

```go
// As finds the first error in err's chain that matches the
type to which target
// points, and if so, sets the target to its value and returns
true. An error
// matches a type if it is assignable to the target type, or if
it has a method
// As(interface{}) bool such that As(target) returns true. As
will panic if target
// is not a non-nil pointer to a type which implements error or
is of interface type.
//
// The As method should set the target to its value and return
true if err
// matches the type to which target points.
func As(err error, target interface{}) bool

// instead of err, ok == err.(OtherError)
if errors.As(err, OtherError) { ... }
```

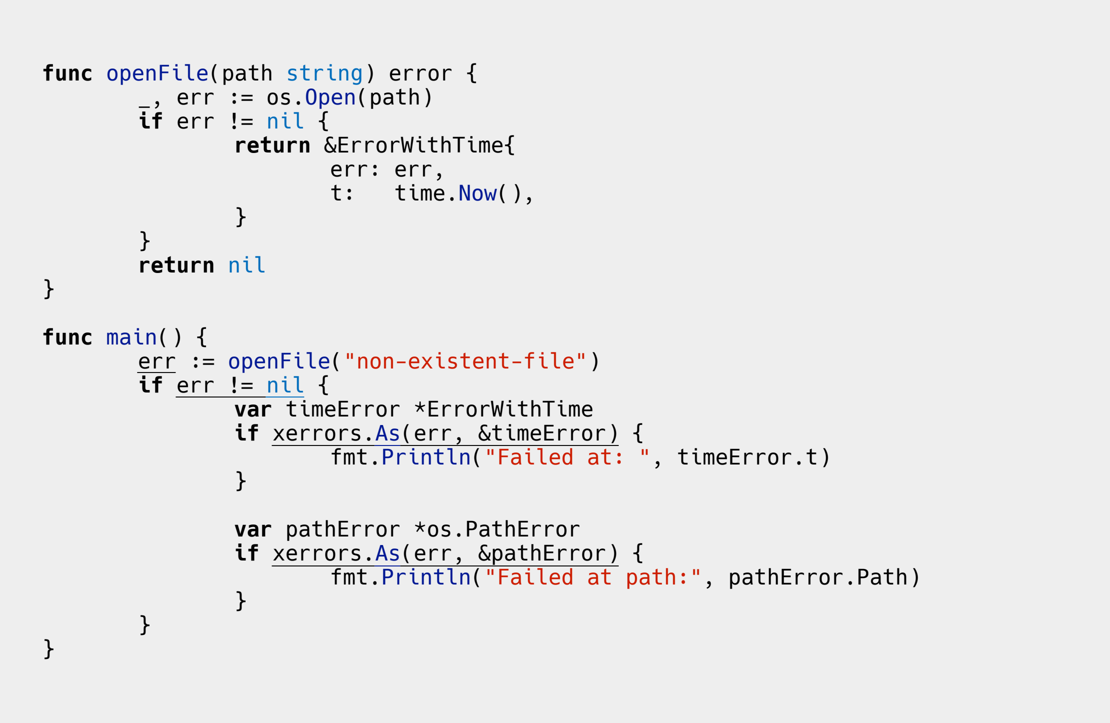
“As” uses **type assertion** or **type switch**.

It can check the **chained** errors thanks to **Unwrap()**.

Go 2 Error Values

Error Inspection / As function

example from here



```
func openFile(path string) error {
        _, err := os.Open(path)
        if err != nil {
                return &ErrorWithTime{
                        err: err,
                        t:   time.Now(),
                }
        }
        return nil
}

func main() {
        err := openFile("non-existent-file")
        if err != nil {
                var timeError *ErrorWithTime
                if xerrors.As(err, &timeError) {
                        fmt.Println("Failed at: ", timeError.t)
                }

                var pathError *os.PathError
                if xerrors.As(err, &pathError) {
                        fmt.Println("Failed at path:", pathError.Path)
                }
        }
}
```

Check if the error is type of “ErrorWithTime“ or “os.PathError" using **errorsAs**.

Go 2 Error Values

Error Inspection / discussion

- **Don’t export** debug-purpose fields of the errors that implement the **Unwrap** method to **allow** users to **inspect** from **outside** your package.
- **Don’t implement the Unwrap** method if you want to allow users to inspect only your errors, but **not any wrapped** errors.
- These approaches **don’t support multiple check** at once. You should implement it yourself.
- Optional “Is" and “As” **methods overriding** for the errors for **default checks** in "errors.Is" and “errors.As".

Go 2 Error Values

Error formatting

```
write users database: call myserver.Method: \
    dial myserver:3333: open /etc/resolv.conf: permission denied
```

Not structured, inconvenient to read.

We need a **standard**, well-**formatted** error printing like stack traces.

Go 2 Error Values

Error formatting

```
write users database:
    more detail here
    mypkg/db.Open
        /path/to/database.go:111
  - call myserver.Method:
    google.golang.org/grpc.Invoke
        /path/to/grpc.go:222
  - dial myserver:3333:
    net.Dial
        /path/to/net/dial.go:333
  - open /etc/resolv.conf:
    os.Open
        /path/to/os/open.go:444
  - permission denied
```

How to achieve it in Go 2?

Go 2 Error Values

Error formatting / Formatter

```
type Formatter interface {
        error

        // FormatError prints the receiver's first error and returns the next
error to
        // be formatted, if any.
        FormatError(p Printer) (next error)
}

type Printer interface {
        // Print appends args to the message output.
        Print(args ...interface{})

        // Printf writes a formatted string.
        Printf(format string, args ...interface{})

        // Detail reports whether error detail is requested.
        // After the first call to Detail, all text written to the Printer
        // is formatted as additional detail, or ignored when
        // detail has not been requested.
        // If Detail returns false, the caller can avoid printing the detail at
all.
        Detail() bool
}
```

With **FormatError** method, we can print **chained** error messages.

The `Printer` interface is designed to allow localization.

Go 2 Error Values

Error formatting / Formatter

```
func (e *WriteError) FormatError(p errors.Printer) (next error)
{
	p.Printf("write %s database", e.Database)
	if p.Detail() {
		p.Printf("more detail here")
	}
	return e.Unwrap()
}

...

fmt.Printf("%+v", err)
```

Use **"%+v"** to print the error in the detailed, multi-line format.

Go 2 Error Values

Error formatting / Formatter

```
// Inlined error message
write users database: call myserver.Method: \
    dial myserver:3333: open /etc/resolv.conf: permission denied

// Formatted error message
write users database:
    more detail here
  - call myserver.Method:
  - dial myserver:3333:
  - open /etc/resolv.conf:
  - permission denied
```

Top error message
+ Detail error message
+ Chained error messages

Go 2 Error Values

Error formatting / Formatter

```
// Inlined error message
write users database: call myserver.Method: \
    dial myserver:3333: open /etc/resolv.conf: permission denied

// Formatted error message
write users database:
    more detail here
    mypkg/db.Open
        /path/to/database.go:111
  - call myserver.Method:
    google.golang.org/grpc.Invoke
        /path/to/grpc.go:222
  - dial myserver:3333:
    net.Dial
        /path/to/net/dial.go:333
  - open /etc/resolv.conf:
    os.Open
        /path/to/os/open.go:444
  - permission denied
```

When **p.Detail** returns true.

Go 2 Error Values

Error formatting / Opaque

```go
// Opaque returns an error with the same error formatting as err
// but that does not match err and cannot be unwrapped.
func Opaque(err error) error
```

The `Opaque` function **hides** a **wrapped** error from programmatic inspection.

Same as current fmt.Errorf.

Go 2 Error Values

Error formatting / fmt.Errorf

```go
e := fmt.Errorf("some text %w", err) // err is SomeError
if xerrors.Is(e, SomeError) {

}

e := fmt.Errorf("some text %w”, err) // err is OtherError
if xerrors.As(e, OtherError) {

}
```

New behavior of `fmt.Errorf`

If the last argument is an We can easily create an **unwrappable** the error with existing **fmt.Errorf**.

Could be a **standard**(?) error wrapping way.

Go 2 Error Values

Error formatting / fmt.Errorf

```go
// Pseudo implementation.
func Errorf(format string, a ...interface{}) error {
        err, wrap := lastError(format, a)
        format = formatPlusW(format)
        if err == nil {
                return &noWrapError{fmt.Sprintf(format, a...), nil, Caller(1)}
        }

        // TODO: this is not entirely correct. The error value could be
        // printed elsewhere in format if it mixes numbered with unnumbered
        // substitutions. With relatively small changes to doPrintf we can
        // have it optionally ignore extra arguments and pass the argument
        // list in its entirety.
        msg := fmt.Sprintf(format[:len(format)-len(": %s")], a[:len(a)-1]...)
        frame := Frame{}
        if internal.EnableTrace {
                frame = Caller(1)
        }
        if wrap {
                return &wrapError{msg, err, frame}
        }
        return &noWrapError{msg, err, frame}
}
```

error `err` and the **format string ends with** `: %s`, `: %v`, **or** `: %w`, then the returned error will implement `FormatError` to return `err`

Go 2 Error Values

Error stack / Stack Frames

```
type Frame struct {
      // unexported fields
}

func Caller(skip int) Frame

// Format prints the stack as error detail.
// It should be called from an error's FormatError
implementation,
// before printing any other error detail.
func (f Frame) Format(p Printer)
```

The `Frame` type holds **location information**: the function **name**, **file** and **line** of a single stack frame.

Go 2 Error Values

Error stack / Stack Frames

```
func (e *WriteError) FormatError(p errors.Printer) (next error)
{
	p.Printf("write %s database", e.Database)
	if p.Detail() {
		e.Frame.Format(p)
	}
	return e.Unwrap()
}
```

With Frame, it will be displayed when the error is formatted with **additional detail**.

SECTION 3

Error Values in Go1.13 (Accepted)

Error Values in Go 1.13

What’s new?

- **errors.Unwrap**
- **errors.Is** and **errors.As**
- **%w** format verb
- ~~Wrapper interface~~
- ~~Opaque~~
- ~~Formatting and location removed~~

Error Values in Go 1.13

What’s new?

https://github.com/golang/go/issues/29934#issuecomment-489682919

https://go.googlesource.com/go/+/refs/tags/go1.13rc1/src/errors/wrap.go

https://go.googlesource.com/go/+/refs/tags/go1.13rc1/src/fmt/errors.go

https://go.googlesource.com/go/+/refs/tags/go1.13rc1/src/fmt/print.go

SECTION 4

Conclusion

Conclusion

There may no silver bullet for error handling

Conclusion

Thank you!

Any Questions?