在Go 2中反思处理错误的方式

Marcel van Lohuizen
Google, Go core team
@mpvl_
github.com/mpvl
mpvl@golang.org

Rethinking Errors for Go 2

关于我

2002年加入Google

- 搜索引擎
- Borg (Kubernetes的灵感来源) 创始成员
- 2011年加入Go团队

About me

At Google since 2002

- Search engine
- Founding member Borg (inspiration for Kubernetes)
- Go team since 2011

```go
r, err := os.Open(“foo.txt”)
if err != nil {
    return fmt.Errorf(“oops: %v”, err)
}
defer r.Close()
```

首先明确语义!

为什么?
合理处理错误很难，往往让人放不慎防

Semantics first!

Why?
error handling can be tricky

```
func writeToGS(c net.Context, bkt, dst string, r io.Reader) error {
    var err error
    w := client.Bucket(bkt).Object(dst).NewWriter(c)
    defer func() { w.CloseWithError(err) }

    if _, err = io.Copy(w, r); err != nil {
        return fmt.Errorf(“oops: %v”, err)
    }
    return nil
}
```

handle panic 失败

```
func writeToGS(c net.Context, bkt, dst string, r io.Reader) error {

    err := errPanicking
    w := client.Bucket(bkt).Object(dst).NewWriter(c)
    defer func() { w.CloseWithError(err) }

    if _, err = io.Copy(w, r); err != nil {
        return fmt.Errorf(“oops: %v”, err)
    }
    return err
}

var errPanicking = errors.New(“panicking”)
```

return error from Close 失败

```
func writeToGS(c net.Context, bkt, dst string, r io.Reader) (err error) {

    w := client.Bucket(bkt).Object(dst).NewWriter(c)
    err = errPanicking
    defer func() {
        if err != nil {
            _ = w.CloseWithError(err)
        } else if err = w.Close(); err != nil {
            err = fmt.Errorf("oh noes: %v", err)
        }
    }

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

Awkward! 尴尬!

```
func writeToGS(c net.Context, bkt, dst string, r io.Reader) (err error) {
    w := client.Bucket(bkt).Object(dst).NewWriter(c)
    err = errPanicking
    defer func() {
        if err != nil {
            _ = w.CloseWithError(err)
        } else if err = w.Close(); err != nil {
            err = fmt.Errorf("oops: %v", err)
        }
    }()

    if _, err = io.Copy(w, r); err != nil {
        return fmt.Errorf("oops: %v", err)
    }
    return nil
}

var errPanicking = errors.New("panicking")
```

如何简化这些?

首先明确语义!

how to approach simplifying this?

semantics first!

Close

error

error

panic

error和panic的相似之处在Go中被忽视了

there is overlap
not recognized in Go

error是可恢复的

panic却不是

(某种意义上)

an error is recoverable

panic is not

(sort of)

github.com/mpvl/errc
github.com/mpvl/errd

将error和panic 指向唯一一个变量

records all errors,
including panics,
in a single place

自动化繁冗的控制流程

automate tedious
control flow

```
func writeToGS(c net.Context, bkt, dst string, r io.Reader) (err error) {
    w := client.Bucket(bkt).Object(dst).NewWriter(c)
    err = errPanicking
    defer func() {
        if err != nil {
           _ = w.CloseWithError(err)
        } else if err = w.Close(); err != nil {
           err = fmt.Errorf(“oops: %v”, err)
        }
    }()

    if _, err = io.Copy(w, r); err != nil {
        return fmt.Errorf(“oops: %v”, err)
    }
    return nil
}

var errPanicking = errors.New(“panicking”)
```

package github.com/mpvl/errc

```
func writeToGS(c net.Context, bkt, dst string, r io.Reader) (err error) {
  e := errc.Catch(&err)
  defer e.Handle()

  w := client.Bucket(bkt).Object(dst).NewWriter(c)
  e.Defer(w.CloseWithError, msg(“oops”))

  _, err = io.Copy(w, r)
  e.Must(err, msg(“oops”))
  return nil
}
```

package github.com/mpvl/errd

```
func writeToGS(c net.Context, bkt, dst string, r io.Reader) error {
  return errd.Run(func(e *errd.E) {
    w := client.Bucket(bkt).Object(dst).NewWriter(c)
    e.Defer(w.CloseWithError, msg(“oops”))

    _, err = io.Copy(w, r)
    e.Must(err, msg(“oops”))
  }
}
```

```
w := client.Bucket(bkt).Object(dst).NewWriter(c)
err = errPanicking
defer func() {
    if err != nil {
       _ = w.CloseWithError(err)
    } else if err = w.Close(); err != nil {
       err = fmt.Errorf(“oops: %v”, err)
    }
}()

if _, err = io.Copy(w, r); err != nil {
    return fmt.Errorf(“oops: %v”, err)
}
return nil
```

```
e := errc.Catch(&err)
defer e.Handle()

w := client.Bucket(bkt).Object(dst).NewWriter(c)
e.Defer(w.CloseWithError, msg(“oops”))

_, err = io.Copy(w, r)
e.Must(err, msg(“oops”))
return nil
```

Err Dare

演示

github.com/mpvl/errdare

如何在Go中做到这些?

How to translate this to
the Go language?

其它问题

繁冗的流程控制
添加上下文的重复代码
重复代码

Other issues

Complex and tedious control flow
Repetition of wrappers
Repetition

```go
func writeToGS(c net.Context, bkt, dst string, r io.Reader) (err error) {
    w := client.Bucket(bkt).Object(dst).NewWriter(c)
    err = errPanicking
    defer func() {
        if err != nil {
           _ = w.CloseWithError(err)
        } else if err = w.Close(); err != nil {
           err = fmt.Errorf(“oops: %v”, err)
        }
    }()

    if _, err = io.Copy(w, r); err != nil {
        return fmt.Errorf(“oops: %v”, err)
    }
    return nil
}

var errPanicking = errors.New(“panicking”)
```

Go 2 初稿 Go 2 Draft

```
func writeToGS(c context, bkt, dst string, r io.Reader) error {
    handle err { return errors.Wrap(err) }

    w := client.Bucket(bkt).Object(dst).NewWriter(c)
    defer err { try w.CloseWithError(err) }

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

`defer err { … }`

- err被设定为PanicError（如果有panic发生）或者函数返回的错误
- 或者只传递函数返回的错误，而另外添加一个内置函数用以检查panic状态

  - err is set to a PanicError, if there is a panic, or the returned error value otherwise
  - alternatively, pass returned error only and have a builtin to peek panic state

`try` `<expr>`

- 去掉最后一个返回值
- "try os.Open(...)"的类型是*File
- <expr>发生错误时
- 保存错误并引用错误处理函数
- 在defer语句中，若已经存在错误，则其不会被覆盖
- 返回函数

- Strips last evaluated value
  - Type of “try os.Open(…)” is *File
- On error,
  - record the error and call the handler chain.
  - Within defers a previous error is not overwritten
  - Returns from the function

`handle err { … }`

- handle定义一个在try发现错误时执行的语句块
- 以内联方式执行从而保留行号信息
- 变量(err)只在语句块中可见

- handle defines a block to be executed when a try detects an error
- executed in place to preserve line number info
- variable name (err) only visible in block

- 每个语句块可以拥有它自己的错误处理函数
- 自内而外的执行直到执行return语句
- 无return语句时，默认的处理函数将返回错误和零值

- each block may have its own handler
- inside-out execution halts when one returns
- implicit handler that returns error and zero values

```
func foo() error {
  msg := “foo”
  handle err { return wrap(err, msg) }

  {
    handle err { msg = “bar” }
    …
  }
}
```

```
// Computes the eigenvalue factorization of a Hermitian matrix.
```

```go
func EigHerm(a Const) (*Mat, []float64, error) {
    if err := errNonPosDims(a); err != nil {
        return nil, nil, err
    }
    if err := errNonSquare(a); err != nil {
        return nil, nil, err
    }
    if err := errNonHerm(a); err != nil {
        return nil, nil, err
    }
    return eigHerm(cloneMat(a), DefaultTri)
}
```

```go
func EigHerm(a Const) (*Mat, []float64, error) {
    try errNonPosDims(a)
    try errNonSquare(a)
    try errNonHerm(a)
    return eigHerm(cloneMat(a), DefaultTri)
}
```

```
func cpToGS(c net.Context, r io.Reader) error {
  obj := client.Bucket("b").Object("d")
  w := obj.NewWriter(c)
  err := errPanicking
  defer func() {
    if err != nil {
      _ = w.CloseWithError(err)
    } else if err1 = w.Close(); err1 != nil {
      err = err1
    }
  }()

  _, err = io.Copy(w, r)
  return err
}

var errPanicking = errors.New("panicking")
```

```
func cpToGS(c net.Context, r io.Reader) error {
  obj := client.Bucket("b").Object("d")
  w := obj.NewWriter(c)
  defer err { try w.CloseWithError(err) }

  _, err := io.Copy(w, r)
  return err
}
```

如何改正 CloseWithErr?

集中错误于一处

放弃有问题的API

内置函数panicking()

How to fix CloseWithErr?

collect errors in one place

deprecate faulty API

builtin function panicking()

如何为错误添加上下文?

What about adding
context to errors?

这种形式的函数将变为一行内联函数

```
func wrapper(err error, args …interface{}) error {
    if err == nil {
        return nil
    }
    return errors.E(err, args…)
}
```

Functions of this form get inlined

```
p, err := s.capture(ctx, in.GetReservation())
if err != nil {
    return nil, errors.Wrap(err, "capturing
proposal")
}
r := p.reservation
if util.HasFoo(r.GetFoo()) {
    r.State.Blacklist = blacklist(r)
    if err := validate.RequestFoo(r); err != nil {
        return nil, errors.Wrap(err, "with foo")
    }
    return &request{}, nil
}
if err := validate.RequestBar(r); err != nil {
    return nil, errors.Wrap(err, “with bar")
}
return &request{}, nil
```

```
p, err := s.capture(ctx, in.GetReservation())
try errors.Wrap(err, "capturing proposal")

r := p.reservation
if util.HasFoo(r.GetFoo()) {
    r.State.Blacklist = blacklist(r)
    err := validate.RequestFoo(r)
    try errors.Wrap(err, "with foo”)

    return &request{}, nil
}

err := validate.RequestBar(r)
try errors.Wrap(err, “with bar")

return &request{}, nil
```

```
p, err := s.capture(ctx, in.GetReservation())
if err != nil {
    return nil, errors.Wrap(err, "capturing proposal")
}
r := p.reservation
if util.HasFoo(r.GetFoo()) {
    r.State.Blacklist = blacklist(r)
    if err := validate.RequestFoo(r); err != nil {
        return nil, errors.Wrap(err, "with foo")
    }
    return &request{}, nil
}
if err := validate.RequestBar(r); err != nil {
    return nil, errors.Wrap(err, “with bar")
}
return &request{}, nil
```

```
p, err := s.capture(ctx, in.GetReservation())
try e(err, "capturing proposal")

r := p.reservation
if util.HasFoo(r.GetFoo()) {
    r.State.Blacklist = blacklist(r)
    try e(validate.RequestFoo(r), "with foo")
    return &request{}, nil
}
try e(validate.RequestBar(r), “with bar")

return &request{}, nil
```

```go
p, err := s.capture(ctx, in.GetReservation())
if err != nil {
    return nil, errors.Wrap(err, “capturing …”)
}
r := p.reservation
if util.HasFoo(r.GetFoo()) {
    r.State.Blacklist = blacklist(r)
    if err := validate.RequestFoo(r); err != nil {
        return nil, errors.Wrap(err, "with foo")
    }
    return &request{}, nil
}
if err := validate.RequestBar(r); err != nil {
    return nil, errors.Wrap(err, “with bar")
}
return &request{}, nil
```

```go
handle err { errors.Wrap(err, “create request"}

p := try s.capture(ctx, in.GetReservation())
r := p.reservation
if util.HasFoo(r.GetFoo()) {
    r.State.Blacklist = blacklist(r)
    try validate.RequestFoo(r)
    return &request{}, nil
}
try validate.RequestBar(r)

return &request{}, nil
```

Redundancy in error messages 错误信息中的冗余

```
read failed: failed to read file
```

```
func convert(filename string) (err error) {
  if r, err = os.Open(filename); err != nil {
    return fmt.Errorf("open failed: %v", err)
  }
  b := make([]byte, 1000)
  if _, err := r.Read(b); err != nil {
    return fmt.Errorf("read failed: %v, err)
  }
}
```

A single handler avoids redundancy

and line information is preserved!

唯一的处理函数避免了冗余，

同时保留了行号信息！

```
convert: failed to read file
```

```
func convert(filename string) error {
  handle err { errors.Wrap(err, “convert”) }

  r := try os.Open(filename)
  b := make([]byte, 1000)
  try r.Read(b)
}
```

错误包

错误将仍是普通的值.

不同的场景需要不同的实现方式。

Error Packages

Errors will remain values.

Different scenarios

require different implementations.

但是...

定义原语来帮助你使用自己的错误类型

- 栈信息
- 附加属性
- Cause() / Underlying()
- 互通性

But...

define primitives to roll your own error type

- Frame or Stack information
- Attributes
- Cause() and Underlying()
- Interoperability

谢谢

欢迎尝试并反馈意见!

Try it out!

Feedback Welcome!

Marcel van Lohuizen, Go core Team

github.com/mpvl/errdare

github.com/mpvl/errc

github.com/mpvl/errd

@mpvl_