error handling

Gilvan Ritter
github.com/gritt

Consultant Developer - ThoughtWorks

sintaxe vs conceito

- "how do I catch an error in Go?" *syntax*

  vs

- "error handling in Go" *concept*

características

- erros são retornados no padrão
  - `func foo()(...n, error)`

- erros são identificados com
  - `if err != nil {...}`
  - `if err := foo(); err != nil {...}`

características

- erros são valores
  - `a := "hello world"`
  - `b := errors.New("something went wrong")`
    - `string`
    - `*errors.errorString`

características

- você precisa definir a importância de um erro
  - como o erro vai ser tratado/monitorado/logado/output
  - se o erro vai ser ignorado
  - se a execução deve parar
- você precisa agir sobre o erro quando ele acontece

características

- um erro não contém stack trace
- panic contém stack trace
  - panic é para desastres
  - panic não é muito legível
  - stack trace é para desastres

adicionando contexto

- graceful error handling:
  - **Where**, **What**, **Why**
  - mensagens de erro descritivas que irão ajudar no diagnóstico do erro e feedback
    - dados identificadores não sensíveis
    - operação, path, id, url…

adicionando contexto

- "...é responsabilidade da implementação do erro contextualizar o erro. O erro retornado por `os.Open(...)` imprime: "

  `"open /etc/passwd: permission denied"`

  não apenas

  `"permission denied."`

adicionando contexto

- categorizar erros
  - erros de negócio
  - erros de infraestrutura
  - erros esperados — *"invalid credentials"*
  - erros não esperados — *"database is down"*

criando um erro

where, what, why

```
errors.New(
    "transaction validate: invalid amount"
)
```

formatando erros

```
where, what, why

fmt.Errorf(
    `transaction validate: invalid amount "%d"`,
    number
)
```

criando um tipo de erro

```
type TransactionError struct {
    Operation string          // method name
    Message string            // description
    Transaction Transaction // state
}
```

criando um tipo de erro - *error interface*

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

criando um tipo de erro - *error interface*

```
func (e TransactionError) Error() string {
    return e.Message
}
```

expandindo sua funcionalidade

```
func (e TransactionError) Serialize() (int, string) {
    return bytes, …json.Encode(e.Transaction)…
}
```

expandindo sua funcionalidade

```
type DatabaseError struct {
    Query string    // err description
    Err error       // original error
}
```

identificando um erro por conteúdo >1.13

```
err := t.Validate()
```

```
if errors.Is(err, TransactionError) {...}
```

identificando um erro por tipo >1.13

```
err := t.Validate()
```

```
var e &TransactionError
if errors.As(err, e) {
    res := e.Serialize()
}
```

referências

- GopherCon 2019: Marwan Sulaiman - Handling Go Errors
- https://blog.golang.org/error-handling-and-go
- https://blog.golang.org/go1.13-errors
- https://godoc.org/github.com/pkg/errors
- https://middlemost.com/failure-is-your-domain/