Introduction to error handling with Go

Wojtek Frackowski, 10.10.2023

About Me

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

The Error Type

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

```go
import "errors"

func division(x, y int) (int, error) {
    if y == 0 {
        return 0, errors.New("can not divide by zero")
    }

    return x / y, nil
}
```

```go
import "errors"

var ErrDivision = errors.New("division by zero not allowed")

func division(x, y int) (int, error) {
    if y == 0 {
        return 0, ErrDivision
    }

    return x / y, nil
}
```

```go
import "fmt"

func division(x, y int) (int, error) {
    if y == 0 {
        return 0, fmt.Errorf("can not divide %d by zero", x)
    }

    return x / y, nil
}
```

```go
import "fmt"

type CustomDivisionError struct {
    x       int
    y       int
    message string
}

func NewCustomDivisionError(x, y int) *CustomDivisionError {
    return &CustomDivisionError{
        x:       x,
        y:       y,
        message: fmt.Sprintf("can not divide %d by %d", x, y),
    }
}

func (e *CustomDivisionError) Error() string {
    return e.message
}

func (e *CustomDivisionError) GetX() int {
    return e.x
}

func (e *CustomDivisionError) GetY() int {
    return e.y
}
```

Error Wrapping

```go
import (
    "errors"
    "fmt"
    "log"
)

func division(x, y int) (int, error) {
    if y == 0 {
        return 0, fmt.Errorf("can not divide %d by zero", x)
    }

    return x / y, nil
}

func divisionWrapper(x, y int) (int, error) {
    result, err := division(x, y)
    if err != nil {
        // the error operand has to include a %w in order to implement the Unwrap method
        return 0, fmt.Errorf("wrapper: %w", err)
    }

    return result, nil
}

func main() {
    result, err := divisionWrapper(2, 0)
    if err != nil {
        log.Printf("wrapped errror: %s\n", err.Error()) // outputs "wrapped errror: wrapper: can not divide 2 by zero"

        if unwrappedErr := errors.Unwrap(err); unwrappedErr != nil {
            log.Printf("unwrapped error: %s\n", unwrappedErr.Error()) // outputs "unwrapped error: can not divide 2 by zero"
        }
        return
    }

    log.Println(result)
}
```

- Can be returned as nil.
- Typically returned as the last argument in a function.
- Other return variable should be returned as default.
- Message written in lowercase letters without punctuation at the end.

Error Comparison

```go
import (
    "errors"
    "fmt"
    "log"
)

var ErrDivision = errors.New("division by zero not allowed")
var ErrDivisionByOne = errors.New("for some reason division by one is also not allowed")

func division(x, y int) (int, error) {
    if y == 0 {
        return 0, ErrDivision
    } else if y == 1 {
        return 0, ErrDivisionByOne
    }

    return x / y, nil
}

func main() {

    divisionResult, err := division(1, 0)
    if err != nil {
        if errors.Is(err, ErrDivision) { // equivalent to err == ErrDivision
            panic(err)
        }

        log.Fatalf("division failed: %s", err.Error())
    }

    fmt.Printf("result of the division: %d\n", divisionResult)
}
```

```go
import (
    "errors"
    "fmt"
    "log"
)

type CustomDivisionError struct {
    x       int
    y       int
    message string
}

func division(x, y int) (int, error) {
    if y == 0 {
        return 0, NewCustomDivisionError(x, y)
    }

    return x / y, nil
}

func main() {

    divisionResult, err := division(1, 0)
    if err != nil {

        var customErr *CustomDivisionError
        if errors.As(err, &customErr) {
            log.Printf("extracted from error: x: %d y: %d message: %s",
                customErr.GetX(),
                customErr.GetY(),
                customErr.Error())
            return
        }

        log.Fatalf("division failed: %s", err.Error())
    }

    fmt.Printf("result of the division: %d\n", divisionResult)
}
```

Questions?

Thank you for listening :)