GESTION DES ERREURS DANS LE LANGAGE DE PROGRAMMATION GO

Présentation

Il est possible avec le langage de programmation go de gérer les erreurs plus précisément Go nous permet de **détecter** ou de **créer une erreur** pour ensuite la manier comme bon nous semble.

La première chose à réaliser pour gérer d'éventuelles erreurs lors de votre compilation, c'est avant tout de les repérer .

Détection d'erreurs

Je vais volontairement sur cet exemple provoquer une erreur lors de l'exécution d'une fonction, et le compilateur va nous prévenir d'une manière ou d'une autre

qu’une erreur a été levée.

```
package main

import (
    "bufio"
    "fmt"
    "os"
    "strconv"
)

func division() {
    scanner := bufio.NewScanner(os.Stdin)
    fmt.Print("Entrez un chiffre : ")
    scanner.Scan()
    nbr, _ := strconv.Atoi(scanner.Text())
    fmt.Println("Résultat :", 1000/nbr)
}

func main() {
    division()
    fmt.Println("Fin")
}
```

Erreur :

```
Entrez un chiffre : nope
panic: runtime error: integer divide by zero
```

Sans aucune surprise nous obtenons une erreur qui nous explique qu'il est impossible de diviser par 0.

Pour le bon déroulement de notre programme, il faut gérer cette erreur en expliquant à l'utilisateur qu'il n'est pas possible de diviser par 0 et qu'il doit rentrer un nombre supérieur à 0.

```
package main

import (
    "bufio"
    "fmt"
    "os"
    "strconv"
```

```
)

func division() {
    for true {
        scanner := bufio.NewScanner(os.Stdin)
        fmt.Print("Entrez un chiffre : ")
        scanner.Scan()
        nbr, _ := strconv.Atoi(scanner.Text())
        if nbr <= 0 {
            fmt.Println("[division par zéro impossible] Votre valeur doit être supéri
        } else {
            fmt.Println("Résultat :", 1000/nbr)
            break
        }
    }

}

func main() {
    division()
    fmt.Println("Fin")
}
```

Résultat :

```
Entrez un chiffre : dsfsdf
[division par zéro impossible] Votre valeur doit être supérieur ou égal à 0
Entrez un chiffre : dssd
[division par zéro impossible] Votre valeur doit être supérieur ou égal à 0
Entrez un chiffre : 78
Résultat : 12
Fin
```

C'est cool car on a résolu le problème de la division par zéro mais par contre lorsque l'utilisateur rentre des caractères on lui affiche toujours le même message comme quoi son entrée doit être supérieur à 0. Il serait plus judicieux de l'informer qu'il doit rentrer un nombre et non des caractères.

Pour afficher cette information on va analyser un peu plus en détail la fonction `Atoi()`.

Déjà une chose est sûr l'erreur est levée lors de la fonction `Atoi()`, voyons voir un peu plus en détail le prototype de cette fonction :

```
func Atoi(s string) (int, error)
```

On peut apprendre grâce au prototype que la fonction `Atoi()` retourne deux types de valeurs :

- Un type `int` qui correspond à la chaîne de caractères `s` converti en entier
- un type `error` qui est l'erreur retourné par la fonction `Atoi()` qu'on peut gérer

Donc pour gérer cette erreur de conversion il faut vérifier la valeur de retour de `error` de la fonction `Atoi()` dans une condition.

```
package main

import (
    "bufio"
    "fmt"
    "os"
    "strconv"
)

func division() {
    for true {
        scanner := bufio.NewScanner(os.Stdin)
        fmt.Print("Entre un chiffre : ")
        scanner.Scan()
        nbr, err := strconv.Atoi(scanner.Text())
        if err != nil { // Gestion de l'erreur de la fonction  Atoi()
            fmt.Println("Vous devenez rentrer un nombre et non une chaîne de caractèr
        } else if nbr <= 0 {
            fmt.Println("[division par zéro impossible] Votre valeur doit être supéri
        } else {
            fmt.Println("Résultat :", 1000/nbr)
            break
        }
    }
}

func main() {
    division()
    fmt.Println("Fin")
}
```

Résultat :

```
Entre un chiffre : -5
[division par zéro impossible] Votre valeur doit être supérieur ou égal à 0
Entre un chiffre : nope
Vous devenez rentrer un nombre et non une chaîne de caractères !
Entre un chiffre : 5
Résultat : 200
Fin
```

Créer une erreur

L'interface error

Il est possible de construire un message d'erreur grâce à la fonction `New()` de la structure `errors`.

```
package main

import (
    "errors"
    "fmt"
    "os"
)

func verificationDivision(nbr float64) (float64, error) {
    if nbr <= 0 {
        return 0, errors.New("Erreur: Il est impossible de diviser par 0 !") //créati
    } else {
        return nbr, nil // on retourne nil si aucune erreur est détectée
    }
}

func main() {
    nbr := 0.0

    nbr, err := verificationDivision(nbr)

    if err != nil {
        panic(err)
    } else {
        fmt.Println("Aucune erreur")
    }
```

```
}
```

Résultat :

```
panic: Erreur: Il est impossible de diviser par 0 !

goroutine 1 [running]:
exit status 2
```

Information

Si votre fonction retourne une interface error alors vous ne pouvez retourner que le type `nil` ou le type `errors`

Ici j'utilise la fonction `panic()` pour **quitter mon programme** avec un **code retour** différent de 0. Un code d'erreur égale à zéro indique que l'exécution de votre programme s'est bien déroulée si il est différent de zéro cela indique un **échec** de votre programme. C'est une valeur qui est retournée lors de la fin d'exécution de votre programme.

Sur Linux il est possible d'afficher le dernier code d'erreur d'un programme/commande en utilisant la commande suivante :

```
echo $?
```

Il est intéressant de retourner le bon code d'erreur lors de l'exécution de votre programme, car c'est possible que cette valeur soit récupérée et exploitée par une autre personne qui utilise votre programme en tant que script pour vérifier si votre programme s'est bien exécuté.

Recommandation

la fonction `errors.New()` ne fait que renvoyer un string, donc il est totalement possible de **créer votre propre système de gestion d'erreur** comme sur l'exemple ci-dessous :

```
package main

import (
 "fmt"
)

func verificationDivision(nbr float64) (float64, bool) {
 if nbr <= 0 {
  return 0, false
 } else {
  return nbr, true
 }
}

func main() {
 nbr := 0.0

 nbr, err := verificationDivision(nbr)

 if err == false {
  panic("Erreur: Il est impossible de diviser par 0 !")
 } else {
  fmt.Println("Aucune erreur")
 }
}
```

Mais il reste préférable de gérer vos erreurs en utilisant la fonction `errors.New()` afin de rendre votre code plus lisible et clair. Cela aide à la compréhension, à la relecture, au contrôle visuel et à la maintenance de votre code.