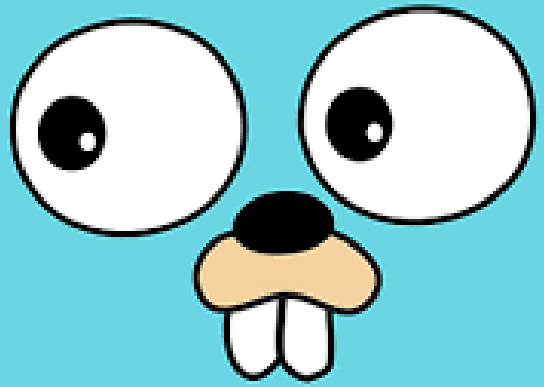

Reflections

Discussion Points

- Understanding reflection in Go
- Using reflect package

Understanding reflection in Go

What is Reflection?

Reflection is the ability of a program to inspect and manipulate its own types and values at runtime.

In Go:
Provided by the reflect package
Enables runtime type inspection
Allows dynamic behavior when types are unknown

Why it exists
Work with unknown types
Build reusable libraries/frameworks
Runtime metadata processing

Understanding reflection in Go

Core Concepts?

Reflection is based on two key concepts:

1 Type
Describes what something is.
reflect.TypeOf(x)
Examples: int, struct, slice, map

2 Value
Represents the actual runtime data.
reflect.ValueOf(x)

```go
package main

import (
    "fmt"
    "reflect"
)

func main() {
    x := 42

    t := reflect.TypeOf(x)
    v := reflect.ValueOf(x)

    fmt.Println( a...: "Type:", t)
    fmt.Println( a...: "Value:", v)
}
```

Using reflect package

Most Practical Use Case?

- Inspecting Struct Fields
- Reading Struct Values
- Modifying Values
- Struct Tags (Most Practical Use Case)

Using reflect package

Inspecting & Reading Struct Fields?

reflect.TypeOf(x)

1. NumField()
2. Field()

```go
type User struct {  1 usage
    Name string
    Age  int
}

func main() {
    u := User{ Name: "Ali",  Age: 30}

    t := reflect.TypeOf(u)

    for i := 0; i < t.NumField(); i++ {
        field := t.Field(i)
        fmt.Println(field.Name, field.Type)
    }
}
```

Using reflect package

Modifying Values ?

Iterate over reflected elements

1. FieldByName()
2. Set

```go
type User struct {  1 usage
    Name string
    Age  int
}

func main() {

    u := User{ Name: "Ali",  Age: 30}

    v := reflect.ValueOf(&u).Elem()
    v.FieldByName( name: "Age").SetInt( x: 31)

    fmt.Println(u.Age)

}
```

Using reflect package

Struct Tags?

Iterate over reflected elements

1. FieldByName()
2. field.Tag.Get()

```go
type User struct {  1 usage
    Name string `json:"name_json"`
    Age  int
}

func main() {

    u := User{ Name: "Ali",  Age: 30}

    t := reflect.TypeOf(u)

    field, _ := t.FieldByName( name: "Name")
    fmt.Println(field.Tag.Get( key: "json"))

}
```

Using reflect package

Real-World usage?

Reflection is heavily used in:

- JSON serialization (encoding/json)
- ORM libraries (e.g., GORM)
- Dependency Injection systems
- Validation libraries
- Config loaders
- Middleware frameworks

Using reflect package

Reflection Drawbacks?

Reflection is powerful but comes with tradeoffs:

- ❌ Runtime performance overhead
- ❌ Reduced type safety
- ❌ Harder debugging
- ❌ Panic risk
- ❌ More complex code

If you can avoid reflection, prefer simpler alternatives.

Using reflect package

Best Practices?

✓ Use reflection mostly in infrastructure layers
✓ Cache reflection results when possible
✓ Prefer interfaces or generics first

Using reflect package

Reflections vs Generics?

Reflection	Generics
Runtime	Compile-time
Slower	Faster
Dynamic	Type-safe
Complex	Cleaner

Thank you.
Questions?

“Learn from yesterday, live for today, hope for tomorrow. The important thing is not to stop questioning.”

- Albert Einstein