



Интерфейсы в Go

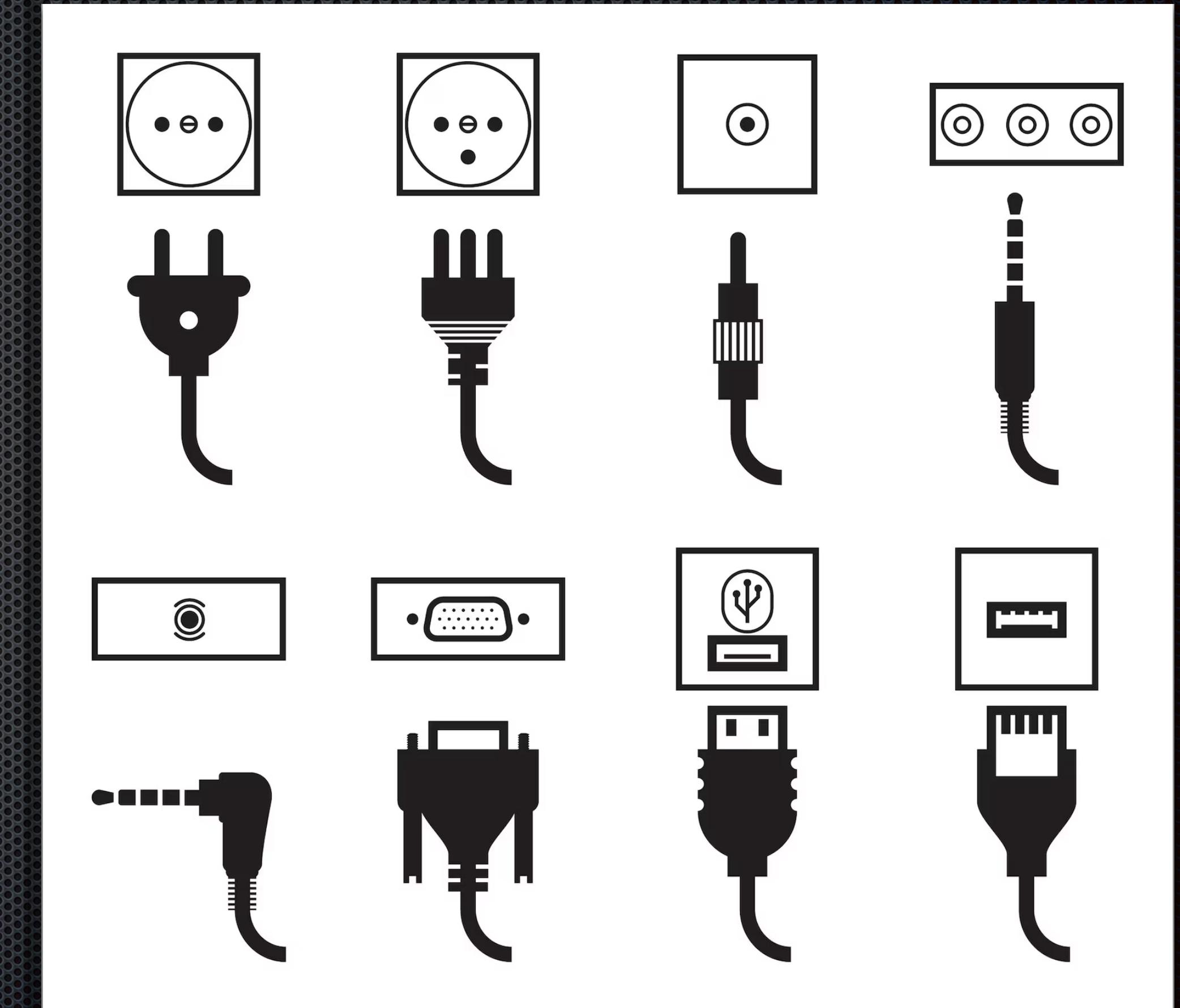
Интерфейсы в общем, и в go в частности

Что такое интерфейсы в общем?

В ООП - это описание совокупности поведения и состояния

Правила поведения*

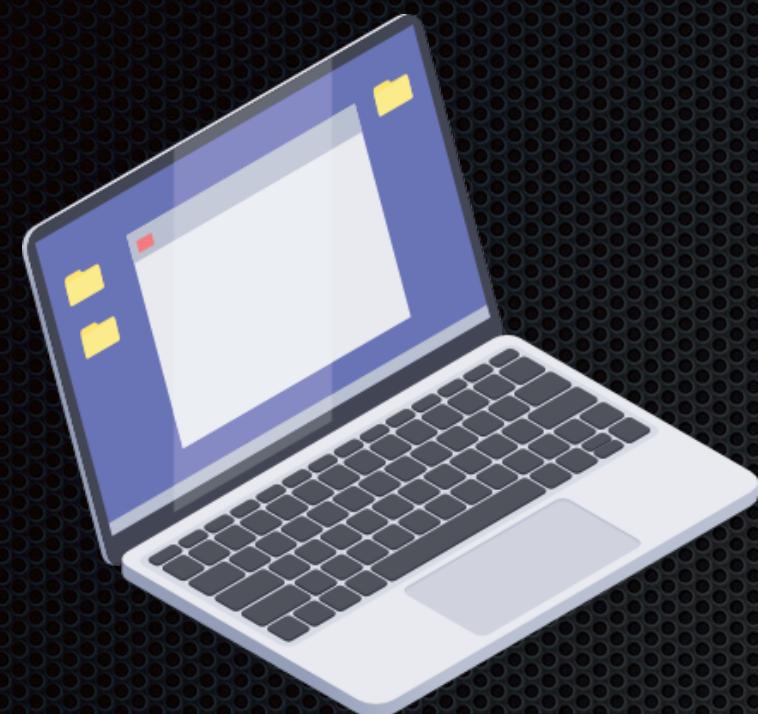
*или состояния и поведения



Структуры в Go

Объектом в Go являются структуры, вместо привычных классов

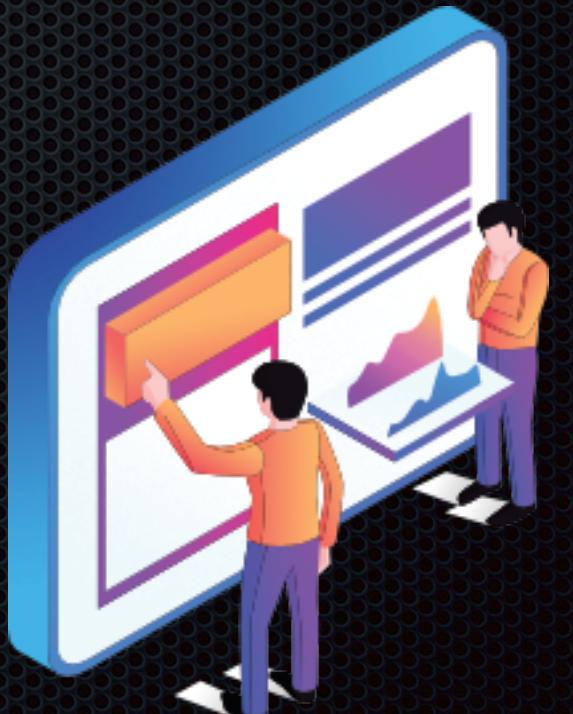
*К примеру, в C++ есть и то, и другое



```
type Product struct {  
    Id          int  
    Name        string  
    Desc        string  
    Price       int  
}
```



```
type Order struct {  
    Id          int  
    Products   []Product  
    Total      int  
    Created    time.Time  
}
```



```
type User struct {  
    Id          int  
    Name        string  
    Orders     []Order  
    Registered time.Time  
}
```

Методы типов

Метод типа (type method) в Go — это функция со специальным аргументом приемником. Такой метод объявляется как обычная функция с дополнительным параметром, который ставится перед именем функции. Данный параметр связывает функцию с типом этого дополнительного параметра. Именно этот параметр называется приемником метода (reciever).



```
type NotifyService struct {  
    Config      *Config  
    CreatedTpl  *Template  
    PaidTpl     *Template  
    CompletedTpl *Template  
}
```

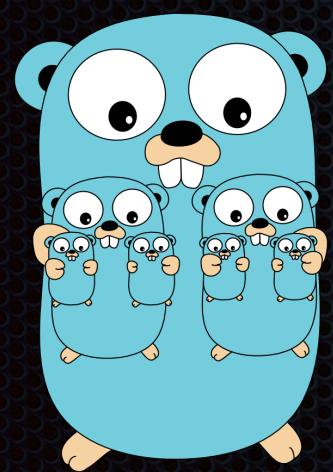
```
func (ns *NotifyService) OrderCreated(order *Order) error {  
    ...  
}  
  
func (ns *NotifyService) OrderPaid(order *Order) (int, error) {  
    ...  
}  
  
func (ns *NotifyService) OrderCompleted(order *Order) error {  
    ...  
}
```

Внедрение зависимости

Dependency injection (внедрение зависимости) — компонование сущностей, таким образом, что одна сущность (зависимость) становится частью состояния другой (родительской сущности). Родительская сущность затем использует зависимость при необходимости.



```
type OrderService struct {
    log      *Logger
    notifyService *NotifyService
}
```



```
func NewOrderService(log *Logger, notifyService *NotifyService) {
    return &orderService{
        log: log,
        notifyService: notifyService
    }
}

log := NewLogger()
notifyService := NewNotifyService(log)
orderService := NewOrderService(log, notifyService)
```

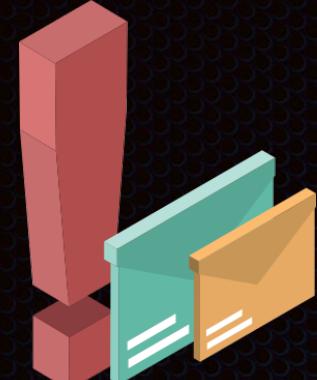
Интерфейсы в Go

Интерфейсный тип в Go определяет поведение других типов путем предоставления списка методов, которые необходимо реализовать. Чтобы тип соответствовал интерфейсу, в нем необходимо реализовать все методы, предусмотренные этим интерфейсом.



```
type Notifier interface {
    OrderCreated(order *Order) error
    OrderPaid(order *Order) (int, error)
    OrderCompleted(order *Order) error
}
```

Использование интерфейса Go



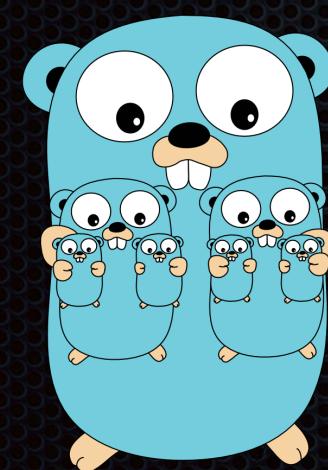
```
type Notifier interface {
    OrderCreated(order *Order) error
    OrderPaid(order *Order) int, error
    OrderCompleted(order *Order) error
}
```



```
type MailNotifyService struct {
    Config      *Config
    CreatedTpl  *Template
    PaidTpl     *Template
    CompletedTpl *Template
}
```



```
type TelegramNotifyService struct {
    TgBotApi      *TgBotApi
    CreatedMsg   string
    PaidMsg      string
    CompletedMsg string
}
```



```
var notifyService Notifier
log := NewLogger(...)

if cfg.notifier == NOTIFIER_TELEGRAM {
    notifyService := NewTelegramNotifyService(log)
} else {
    notifyService := NewMailNotifyService(log)
}
orderService := NewOrderService(log, notifyService)
```

Сегрегация и композиция интерфейсов

The bigger the interface, the weaker the abstraction
- Rob Pike



```
type OrderCreatedNotifier interface {
    OrderCreated(order *Order) error
}

type OrderPaidNotifier interface {
    OrderPaid(order *Order) int, error
}

type OrderCompletedNotifier interface {
    OrderCompleted(order *Order) error
}

type Notifier interface {
    OrderCreatedNotifier
    OrderPaidNotifier
    OrderCompletedNotifier
}
```

Интерфейсы с использованием библиотек

```
package main

import (
    "go.uber.org/zap"
    "go.uber.org/zap/zapcore"
)

type ServiceLogger interface {
    Debug(args ...interface{})
    Info(args ...interface{})
    Warn(args ...interface{})
    Error(fields ...interface{})
}

func main() {
    var log ServiceLogger
    log = NewLogger()

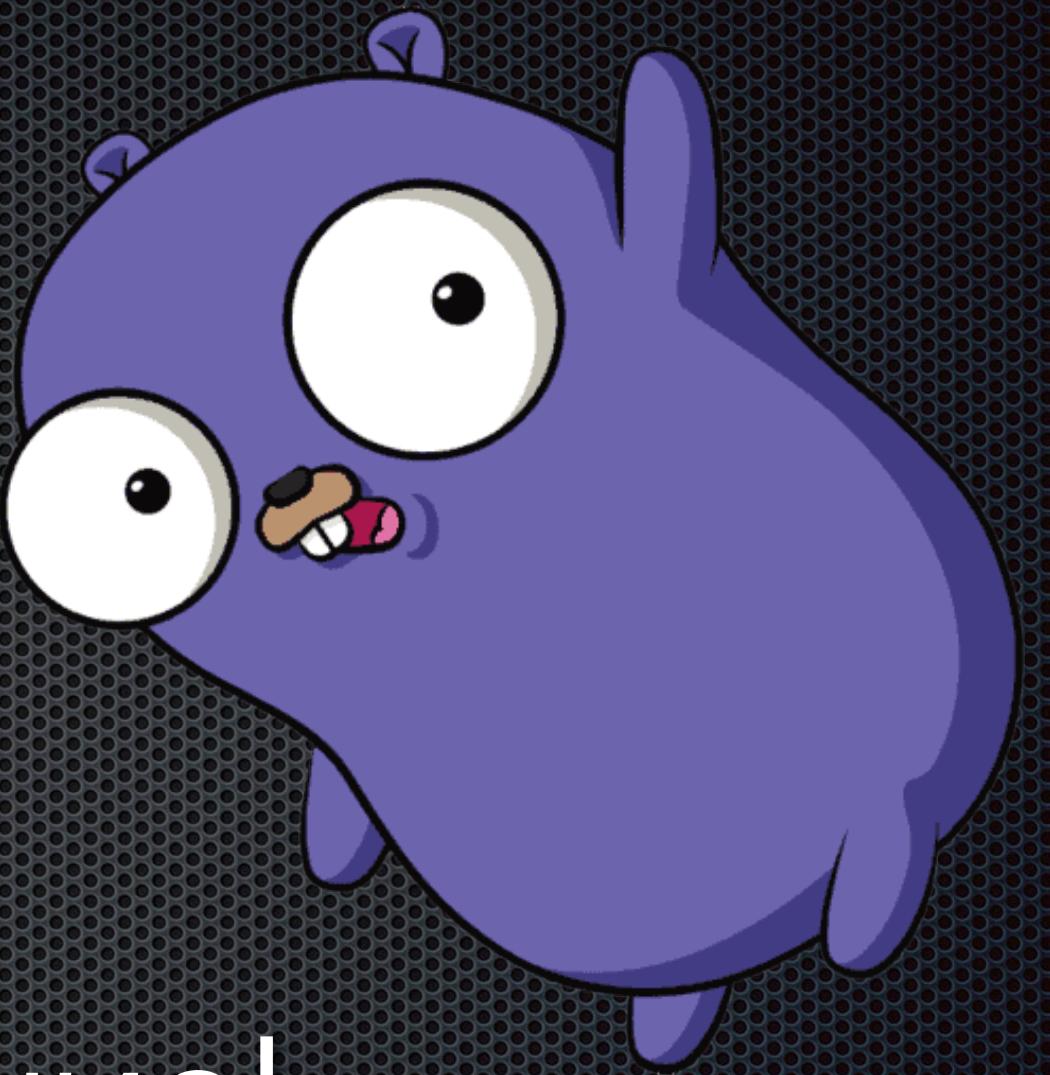
    // Example logs
    log.Debug("This is a debug message.")
    log.Info("This is an info message.")
    log.Warn("This is a warning message.")
    log.Error("This is an error message.")
}
```

```
func NewLogger() *zap.SugaredLogger {
    config := zap.Config{
        Level:           zap.NewAtomicLevelAt(zapcore.DebugLevel),
        Development:    true,
        Encoding:       "json",
        EncoderConfig:  zap.NewProductionEncoderConfig(),
        OutputPaths:    []string{"stdout"},
        ErrorOutputPaths: []string{"stderr"},
    }

    logger, _ := config.Build()
    defer logger.Sync()

    sugar := logger.Sugar()

    return sugar
}
```



Всем спасибо за внимание!

Github: <https://github.com/capsulated/>
Site: <https://logiq.one/>