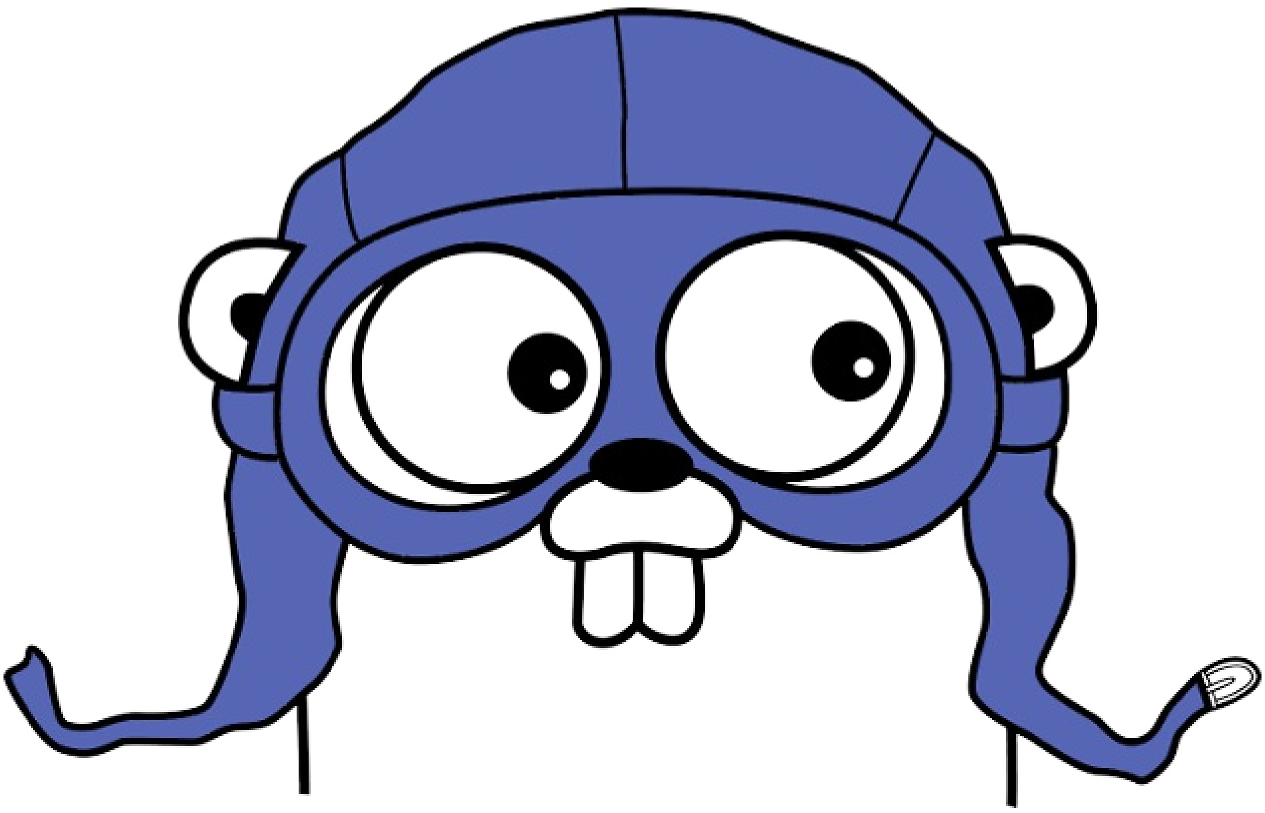
Table of Contents



Go Patterns

一堆Go语言惯例和应用模式，翻译自go patterns，翻译目的主要是为了自己学习go语言。

创建模式

模式	描述	状态
抽象工厂Abstract Factory	一个用于创建相关对象族的接口	✘
构建器Builder	利用简单对象构建一个复杂对象	✔
工厂方法Factory Method	将对象创建工作推迟委派给一个指定的函数	✔
对象池Object Pool	实例化并维护一组相同类型的对象实例	✔
单例Singleton	限制只能实例化一个对象	✔

结构模式

模式	描述	状态
Bridge	Decouples an interface from its implementation so that the two can vary independently	✘
Composite	Encapsulates and provides access to a number of different objects	✘
Decorator	Adds behavior to an object, statically or dynamically	✔
Facade	Uses one type as an API to a number of others	✘
Flyweight	Reuses existing instances of objects with similar/identical state to minimize resource usage	✘
Proxy	Provides a surrogate for an object to control it's actions	✘

行为模式

模式	描述	状态
Chain of Responsibility	Avoids coupling a sender to receiver by giving more than object a chance to handle the request	✘
Command	Bundles a command and arguments to call later	✘
Mediator	Connects objects and acts as a proxy	✘
Memento	Generate an opaque token that can be used to go back to a previous state	✘
Observer	Provide a callback for notification of events/changes to data	✔
Registry	Keep track of all subclasses of a given class	✘
State	Encapsulates varying behavior for the same object based on its internal state	✘
Strategy	Enables an algorithm's behavior to be selected at runtime	✔
Template	Defines a skeleton class which defers some methods to subclasses	✘
Visitor	Separates an algorithm from an object on which it operates	✘

同步模式

模式	描述	状态
Condition Variable	Provides a mechanism for threads to temporarily give up access in order to wait for some condition	✘
Lock/Mutex	Enforces mutual exclusion limit on a resource to gain exclusive access	✘
Monitor	Combination of mutex and condition variable patterns	✘
Read-Write Lock	Allows parallel read access, but only exclusive access on write operations to a resource	✘
Semaphore	Allows controlling access to a common resource	✔

并发模式

模式	描述	状态
N-Barrier	Prevents a process from proceeding until all N processes reach to the barrier	✘
Bounded Parallelism	Completes large number of independent tasks with resource limits	✔
Broadcast	Transfers a message to all recipients simultaneously	✘
Coroutines	Subroutines that allow suspending and resuming execution at certain locations	✘
Generators	Yields a sequence of values one at a time	✔
Reactor	Demultiplexes service requests delivered concurrently to a service handler and dispatches them syncronously to the associated request handlers	✘
Parallelism	Completes large number of independent tasks	✔
Producer Consumer	Separates tasks from task executions	✘

消息模式

模式	描述	状态
Fan-In	Funnels tasks to a work sink (e.g. server)	✔
Fan-Out	Distributes tasks among workers (e.g. producer)	✔
Futures & Promises	Acts as a place-holder of a result that is initially unknown for synchronization purposes	✘
Publish/Subscribe	Passes information to a collection of recipients who subscribed to a topic	✔
Push & Pull	Distributes messages to multiple workers, arranged in a pipeline	✘

稳定性模式

模式	描述	状态
Bulkheads	Enforces a principle of failure containment (i.e. prevents cascading failures)	✘
Circuit-Breaker	Stops the flow of the requests when requests are likely to fail	✔
Deadline	Allows clients to stop waiting for a response once the probability of response becomes low (e.g. after waiting 10 seconds for a page refresh)	✘
Fail-Fast	Checks the availability of required resources at the start of a request and fails if the requirements are not satisfied	✘
Handshaking	Asks a component if it can take any more load, if it can't, the request is declined	✘
Steady-State	For every service that accumulates a resource, some other service must recycle that resource	✘

探测模式

模式	描述	状态
Timing Functions	Wraps a function and logs the execution	✔

惯例

模式	描述	状态
Functional Options	Allows creating clean APIs with sane defaults and idiomatic overrides	✔

反模式

模式	描述	状态
Cascading Failures	Afailure in a system of interconnected parts in which the failure of a part causes a domino effect	✘

构建者Builder模式

构建者模式将复杂对象的构建和表示相互分离，以使相同的构造流程可以创建不同的表示。

在Go语言中，通常用一个配置结构可达到相同的效果，但是传递配置结构易造成构建器方法中包含大量的 `if cfg.Field != nil` 检查。

实现

```go
package car

type Speed float64

const (
    MPH Speed = 1
    KPH       = 1.60934
)

type Color string

const (
    BlueColor  Color = "blue"
    GreenColor       = "green"
    RedColor         = "red"
)

type Wheels string

const (
    SportsWheels Wheels = "sports"
    SteelWheels         = "steel"
)

type Builder interface {
    Color(Color) Builder
    Wheels(Wheels) Builder
    TopSpeed(Speed) Builder
    Build() Interface
}

type Interface interface {
    Drive() error
    Stop() error
}
```

Usage

```go
assembly := car.NewBuilder().Color(car.RedColor)

familyCar := assembly.Wheels(car.SportsWheels).TopSpeed(50 * car.MPH).Build()
familyCar.Drive()

sportsCar := assembly.Wheels(car.SteelWheels).TopSpeed(150 * car.MPH).Build()
sportsCar.Drive()
```

工厂方法模式

工厂方法设计模式，可以在无需指定对象的确切类型的情况下，创建对象。

实现

该样例实现展示如何提供一个不同后端的数据存储，例如内存方式、磁盘存储方式。

类型

```go
package data

import "io"

type Store interface {
    Open(string) (io.ReadWriteCloser, error)
}
```

Different Implementations

```go
package data

type StorageType int

const (
    DiskStorage StorageType = 1 << iota
    TempStorage
    MemoryStorage
)

func NewStore(t StorageType) Store {
    switch t {
    case MemoryStorage:
        return newMemoryStorage( /*...*/ )
    case DiskStorage:
        return newDiskStorage( /*...*/ )
    default:
        return newTempStorage( /*...*/ )
    }
}
```

使用

利用工厂方法，用户可以指定他们想要的存储类型。

```go
s, _ := data.NewStore(data.MemoryStorage)
f, _ := s.Open("file")

n, _ := f.Write([]byte("data"))
defer f.Close()
```

贡献指导

请确保你的拉请求遵循如下指导：

- 针对每个建议做一个独立的拉请求(pull request)
- 选择对应的模式章节做完善或添加
- 确保增加后列表保持词法顺

提交消息指导

- 消息应该采用祈使句，用小写。
- 请尽量在提交消息体中包含解释。
- 使用形式 `<模式-章节>/<模式-名称>: <消息>` (例如 `创建/单例: 重构单例构造函数`)

模式模板

每个模式应当用一个markdown文件，包含尽可能简单且重点的实现，使用和解释，确保读者不需要花费大力气读大量的代码才能理解。

请使用如下模板添加新模式：

```
# <模式-名称>
<模式描述>

## 实现

## 使用

// 可选
## 经验法则
```