CloudWeGo

探索 Hertz 中间件：用法、生态及实现原理

CloudWeGo 开源团队出品

2023/5/31

目录

CONTENT

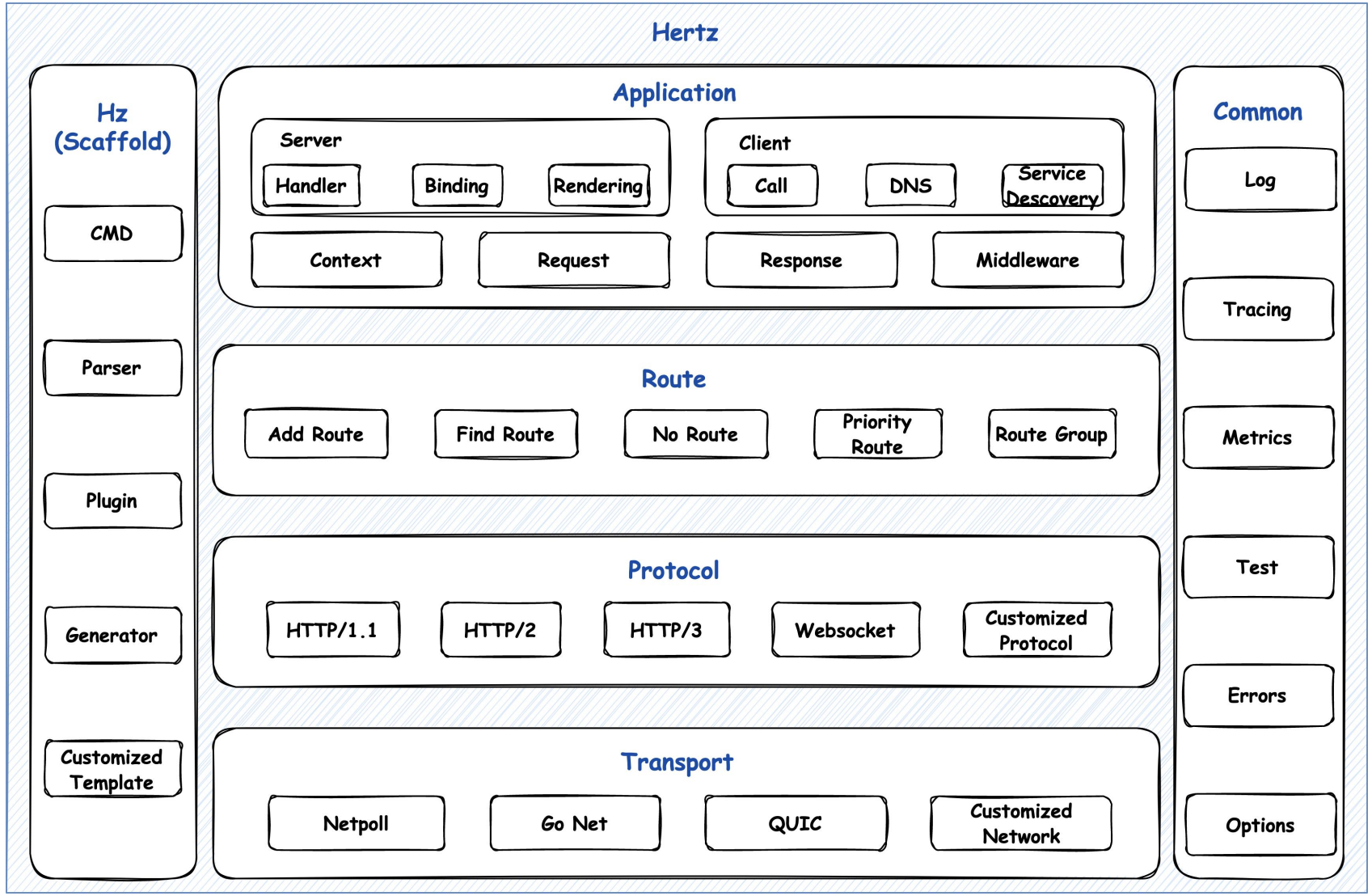
01

Hertz 中间件

Hertz 中间件的概述

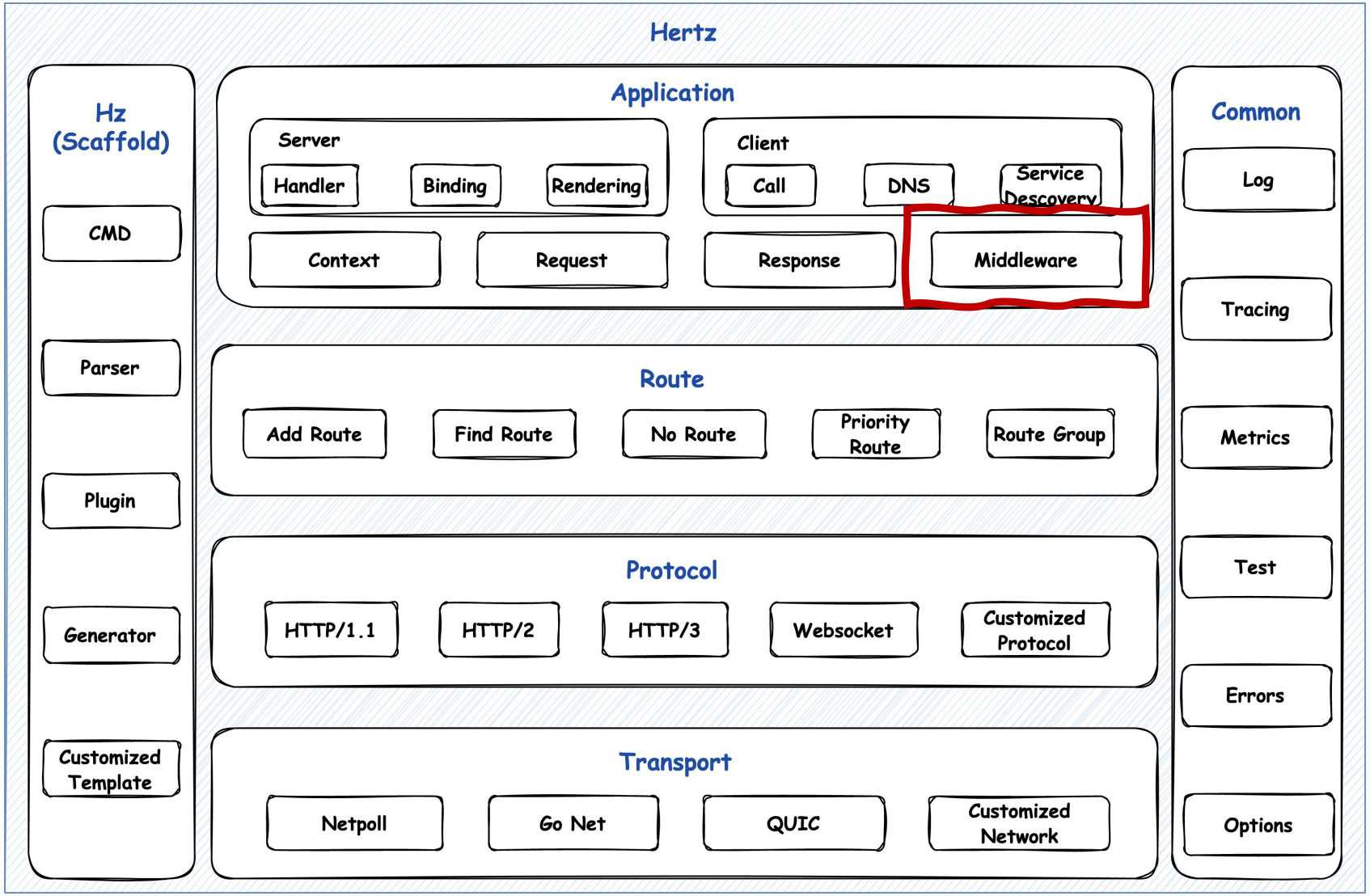
Hertz 介绍

Hertz 是一个 Golang 微服务 HTTP 框架，具有高易用性、高性能、高扩展性等特点。



Hertz 介绍

Hertz 是一个 Golang 微服务 HTTP 框架，具有高易用性、高性能、高扩展性等特点。



什么是中间件？

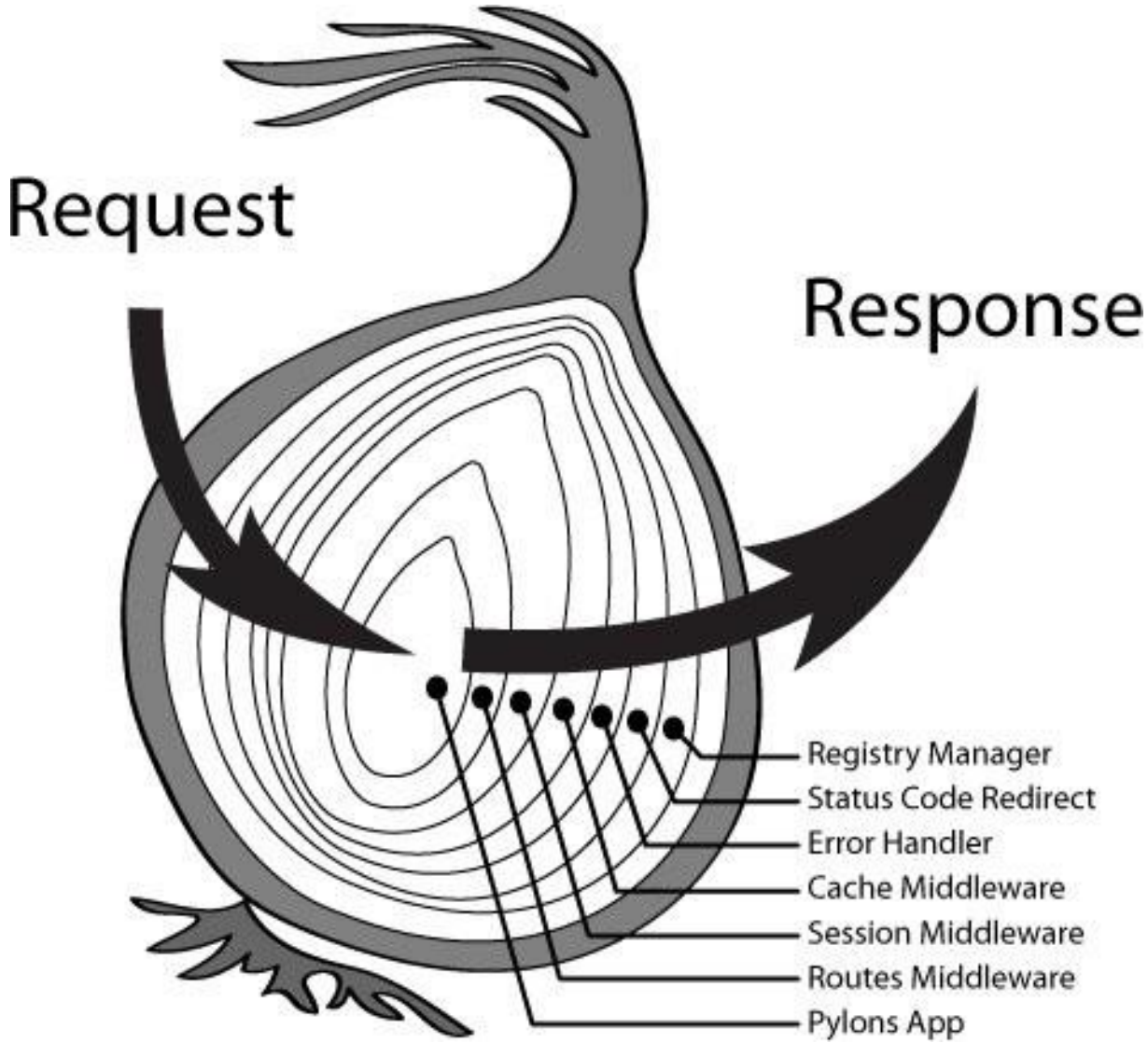
- 中间件设计模式：功能分解，提高可维护性和可扩展性。
- 洋葱模型：中间件处理机制，逐层处理请求和响应，自定义拓展

执行顺序：

RM->SCR->EH->CM->SM->RM->PA->

“业务逻辑”->

PA->RM->SM->CM->EH->SCR->RM



Hertz 中间件的作用

1. **预处理请求**：中间件可以在请求到达业务逻辑之前对请求进行预处理，例如解析请求参数、验证请求头、权限等。

2. **后处理响应**：中间件可以在请求处理完成后对响应进行后处理，例如添加响应头、修改响应数据、记录日志等。

3. **实现逻辑复用**：中间件可以将某些通用逻辑封装起来，以达到复用的目的。例如，认证中间件可以用于多个业务逻辑，避免在每个业务逻辑中都写认证逻辑。

4. **实现功能扩展**：中间件可以用于实现框架本身不具备的功能，例如跨域资源共享、请求解压缩、性能分析等。

Hertz 中间件的使用

全局中间件

- 注册在 Hertz 引擎
- 对所有请求均生效

```go
package main

import (
    "time"

    "github.com/cloudwego/hertz/pkg/app/server"
    "github.com/cloudwego/hertz/pkg/app/middlewares/server/recovery"
    "github.com/hertz-contrib/logger/accesslog"
)

func main() {
    h := server.New()
    h.Use(recovery.Recovery())
    h.Use(accesslog.New())
    h.GET("/ping", func(c context.Context, ctx *app.RequestContext) {
        ctx.JSON(consts.StatusOK, utils.H{"ping": "pong"})
    })
    h.Spin()
}
```

Hertz 中间件的使用

组中间件

- 注册在路由组上
- 对该路由组的所有请求进行处理。

```go
package main

import (
    "time"

    "github.com/cloudwego/hertz/pkg/app/server"
    "github.com/cloudwego/hertz/pkg/app/middlewares/server/basic_auth"
)

func main() {
    h := server.New()
    v1 := h.Group("/v1")
    v1.Use(basic_auth.BasicAuth(map[string]string{
        "username1": "password1",
        "username2": "password2",
        }))

    {
        v1.GET("/ping", func(c context.Context, ctx *app.RequestContext) {
            ctx.JSON(consts.StatusOK, utils.H{"ping": "pong"})
        })
    }
    h.Spin()
}
```

Hertz 中间件的使用

Client 中间件

- 注册在 hertz client 上
- 该 client 发的请求均生效

```go
package main

import (
    "context"

    "github.com/cloudwego/hertz/pkg/app/client"
    "github.com/cloudwego/hertz/pkg/protocol"
)

func main() {
    client, err := client.NewClient()
    if err != nil {
        return
    }
    mw1 := func(next client.Endpoint) client.Endpoint {
        return func(ctx context.Context, req *protocol.Request, resp *protocol.Response) (err error) {
            req.SetRequestURI("http://google.com")
            err = next(ctx, req, resp)
            if err != nil {
                hlog.Errorf("some error happens: %s", err.Error())
                return
            }
            // some logic after executing
            return
        }
    }
    client.Use()
    req := &protocol.Request{}
    res := &protocol.Response{}
    req.SetRequestURI("http://127.0.0.1:8080/middleware")
    err = client.Do(context.Background(), req, res)
    if err != nil {
        return
    }
}
```

02

Hertz 拓展的中间件

Hertz 提供了哪些拓展的中间件

Hertz 自定义一个 server 中间件

1. **中间件函数签名**：app.HandlerFunc

2. **前处理逻辑**：c.Next() 之前的处理逻辑，
业务handler之前的处理逻辑在这里进行

3. **后处理逻辑**：c.Next() 之后的处理逻辑，
业务handler处理之后的处理逻辑在这里进行

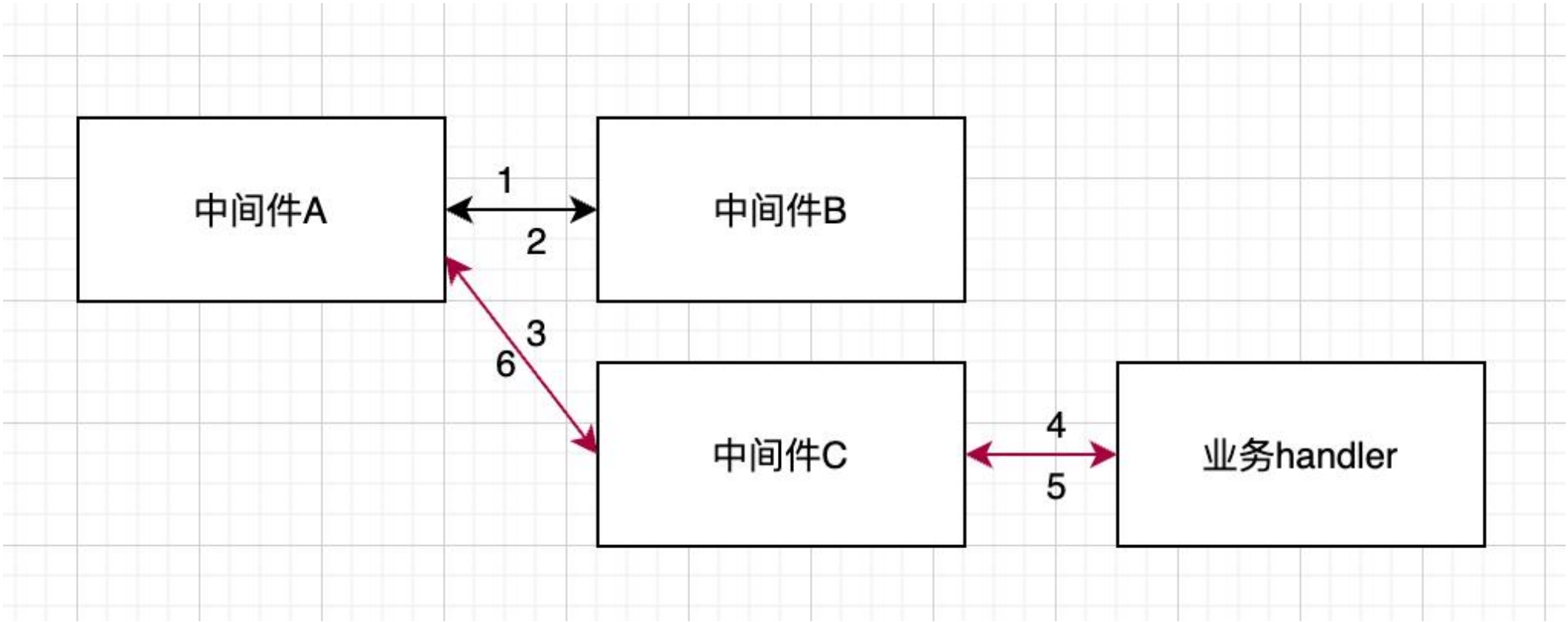
4. **c.Next():** 调用下一个中间件或者业务handler

```go
func MyMiddleware() app.HandlerFunc {
    return func(ctx context.Context, c *app.RequestContext) {
        // 在请求处理之前执行的代码
        fmt.Println("Before request")

        // 快速终止中间件调用
        // c.Abort()

        // 调用下一个中间件或处理函数
        c.Next()

        // 在请求处理之后执行的代码
        fmt.Println("After request")
    }
}
```

Hertz 自定义一个 server 中间件

可以省略 Next() 吗？

1. 可以
2. 省略后，该中间件的逻辑全部为“前处理逻辑”，无法在业务 handler 后对响应进一步处理
3. 省略后，当前中间件的调用与 handler 链的函数调用栈分离



Hertz 自定义一个 server 中间件

如何终止中间件的调用？

1. Abort(): 直接终止后续中间件的调用
2. AbortWithStatus():终止后续中间件的调用，并设置响应“状态码”
3. AbortWithMsg:终止后续中间件的调用，设置响应“状态码”，并将错误信息设置都响应 body

```go
func MyMiddleware1() app.HandlerFunc {
    return func(ctx context.Context, c *app.RequestContext) {
        c.Abort()
        c.AbortWithStatus(300)
        c.AbortWithMsg("some error", 200)
    }
}
```

Hertz Server 中间件的拓展

目前，在社区同学的贡献下，Hertz 已经拥有了丰富的中间件生态，其实现放到了 hertz-contrib 下

类型	中间件	简述
认证/授权	Casbin 中间件	基于Casbin的授权中间件，用于实现RBAC、ABAC、ACL等多种访问控制模型，可以通过配置文件进行动态的权限管理
	JWT 中间件	用于验证JWT令牌，并从中提取用户信息，以便进行授权和身份验证
	KeyAuth 中间件	基于密钥的认证，用于验证请求中的密钥，并根据密钥来授权访问
	Paseto 中间件	用于处理 Paseto 令牌，可以验证Paseto令牌，并从中提取用户信息，以便进行授权和身份验证
	BasicAuth 中间件	用于HTTP基本身份验证，可以用于保护应用程序的敏感资源
安全	Secure 中间件	用于设置HTTP响应头，提高应用的安全性，包括X-XSS-Protection、X-Frame-Options、X-Content-Type-Options等
	CSRF 中间件	用于防止跨站请求伪造攻击（CSRF），可以通过生成和验证token来保护请求的安全性。
	CORS 中间件	用于处理跨域资源共享，防止跨站请求伪造攻击（CSRF）等安全问题
	Limiter 中间件	用于限制请求的频率，可以防止恶意攻击和DDoS攻击等
性能	Etag 中间件	用于生成和验证ETag，可以实现缓存验证和减少网络传输的数据量。
	HttpCache 中间件	用于设置HTTP缓存策略，可以减少网络传输的数据量和提高应用的性能。
	Cache 中间件	用于缓存HTTP响应，可以提高应用的性能

Hertz Server 中间件的拓展

类型	中间件	简述
HTTP 通用能力	RequestID 中间件	用于生成和管理请求ID，可以方便地跟踪请求的流程。
	SSE 中间件	用于实现服务器推送事件（Server-Sent Events），可以实现实时通信。
	Gzip 中间件	用于压缩/解压缩HTTP Body，减少网络传输的数据量。
	Sessions 中间件	用于管理用户会话，可以实现用户登录状态的维护。
功能拓展	OpenTelemetry 中间件	用于分布式追踪和指标收集，可以跟踪请求的链路和性能，并收集应用的指标
	OpenTracing 中间件	用于分布式追踪，可以跟踪请求的链路和性能。
	OpenSergo 中间件	用于接入OpenSergo项目，提供服务治理在内的多种能力。
	I18n 中间件	用于国际化支持，可以根据用户的语言设置返回不同的响应内容。
	Sentry 中间件	用于集成Sentry错误监控平台，可以实现应用的错误监控和告警。
	Recovery 中间件	用于在发生panic时恢复应用程序，避免应用程序崩溃。

03 Hertz 中间件的实现原理

Hertz server 中间件的实现原理

Hertz server 中间件的实现原理

1. 顺序注册

2. 从外到内，从内到外

```go
func MyMiddleware() app.HandlerFunc {
    return func(ctx context.Context, c *app.RequestContext) {
        // 在请求处理之前执行的代码
        fmt.Println("Before request")

        // 快速终止中间件调用
        // c.Abort()

        // 调用下一个中间件或处理函数
        c.Next()

        // 在请求处理之后执行的代码
        fmt.Println("After request")
    }
}
```

Hertz server 中间件的实现原理

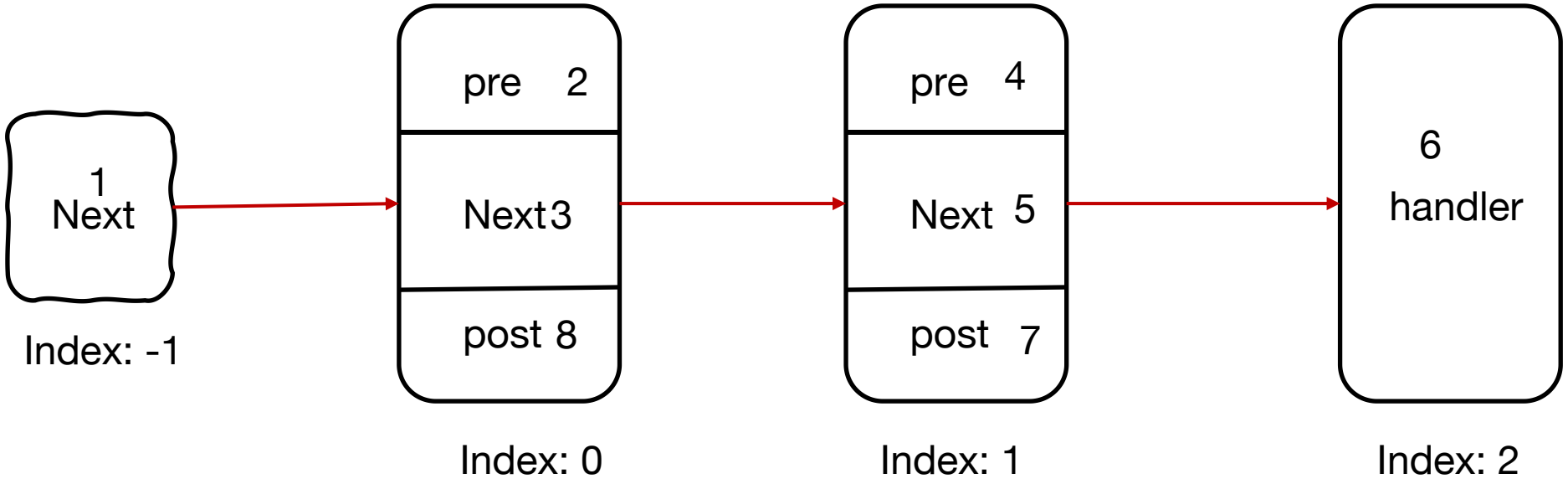
1. 框架发起第一次 Next() 调用
2. 中间件先执行前置逻辑
3. 中间件发起 Next() 调用
4. 中间件执行后置逻辑
5. 中间件退出

```go
// Next should be used only inside middleware.
// It executes the pending handlers in the chain inside the calling handler.
func (ctx *RequestContext) Next(c context.Context) {
    ctx.index++
    for ctx.index < int8(len(ctx.handlers)) {
        ctx.handlers[ctx.index](c, ctx)
        ctx.index++
    }
}
```

```go
type RequestContext struct {
    ...
    handlers    HandlersChain
    index       int8
    ...
}
```

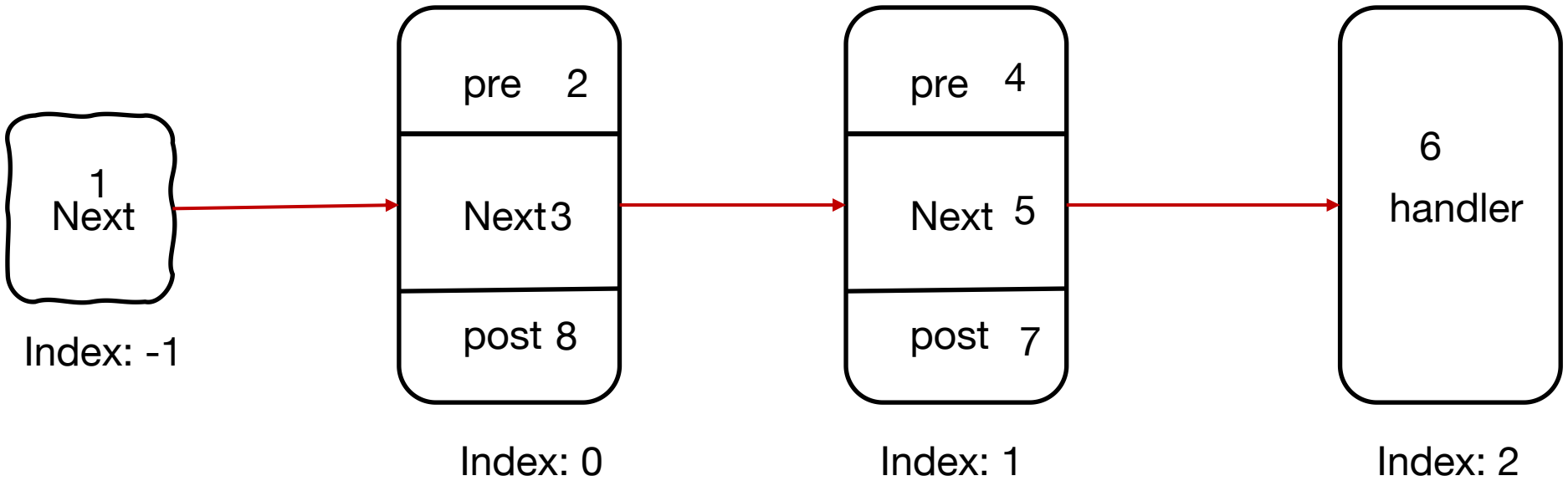
```go
type HandlerFunc func(c context.Context, ctx *RequestContext)

// HandlersChain defines a HandlerFunc array.
type HandlersChain []HandlerFunc
```

Hertz server 中间件的实现原理



步骤	index
1	-1
2	0
3	1
4	1
5	2
6	2
7	3
8	4

Hertz server 中间件的实现原理



步骤	index
1	-1
2	0
3	1
4	1
5	2
6	2
7	3
8	4

问题：如果中间件函数里没有调用 “Next()”，那么其调用逻辑是什么样的？ 请大家分析一下

04 总结

总结

1. Hertz 中间件的作用以及洋葱模型
2. 如何定义并使用 Hertz 的中间件
3. Hertz 拓展的中间件的分类和介绍
4. Hertz 中间件的实现原理，Next() 如何串联起中间件执行链路

CloudWeGo

THANKS

CloudWeGo