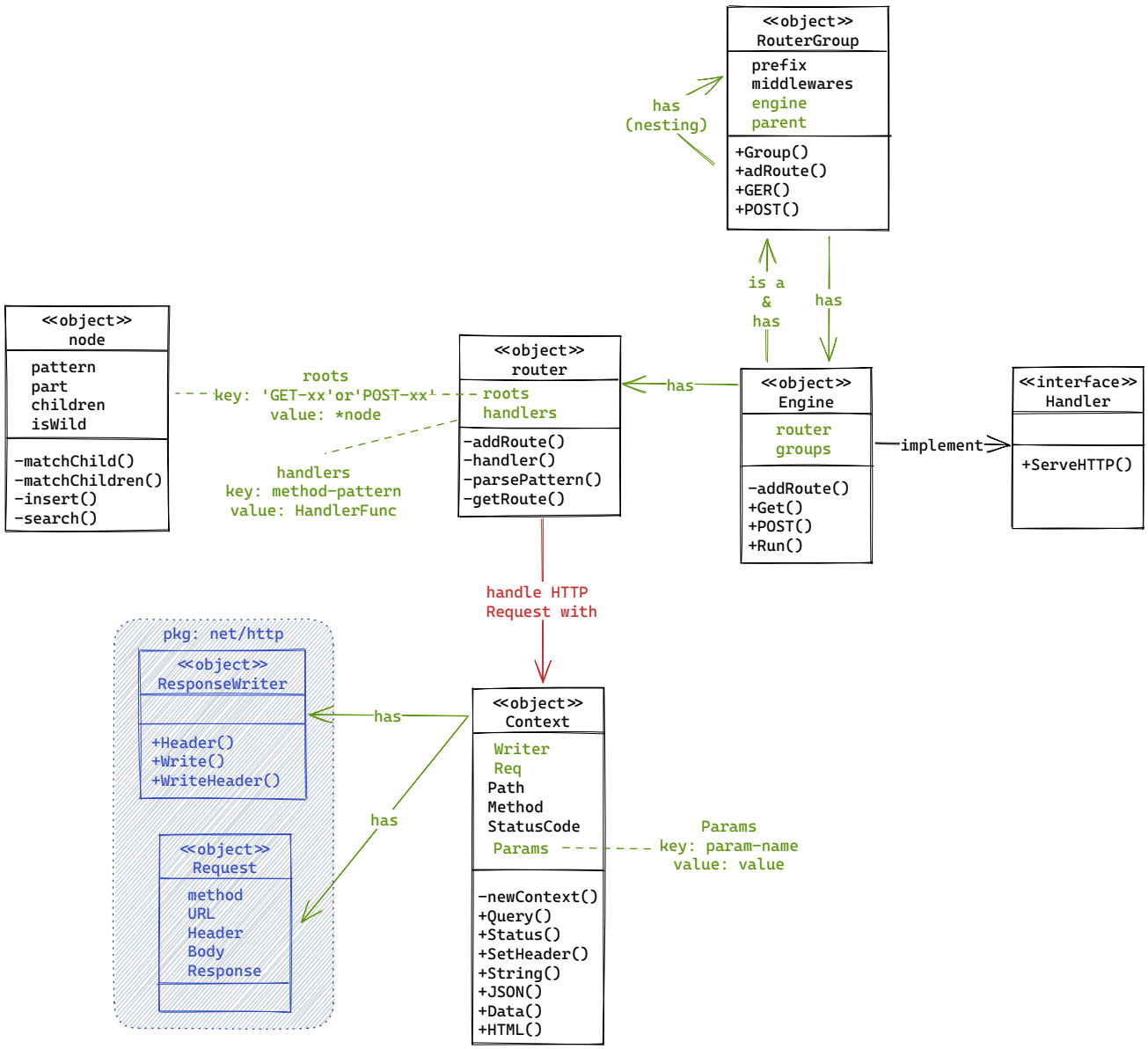
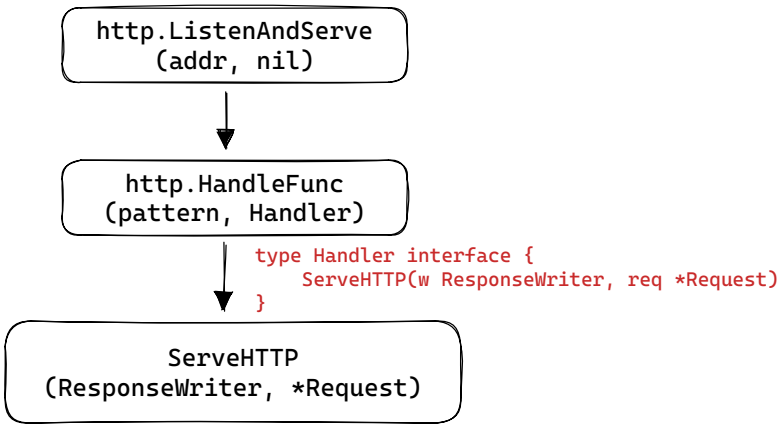
Web 框架 - Gee

#golang #学习



Day 1：Engine



```
type HandlerFunc func(w ResponseWriter, req *Request)
```

- 任何实现了 ServeHTTP 接口的实例，所有的 HTTP 请求就都会交给该实例来处理；
  - 构造 Engine 来捕获 HTTP 请求；
- ResponseWriter：用于构造针对该 HTTP 请求的响应；
- Request：包含了 HTTP 请求的所有信息，如请求地址 URL、请求头 Header、请求体 Body 等信息；

Day 2：Context


```
type HandlerFunc func(c *Context)
```

1. 对 HTTP 请求及响应进行封装，简化借口的调用；
   - 消息头 Header
     - 消息类型 ContentType
     - 状态码 StatusCode
   - 消息体 Body
2. 将扩展性和复杂性留在 Context 内部，对外简化借口；
   - 路由的处理函数，以及要实现的中间件，参数都统一使用 Context 实现；
3. 封装路由
   - 将路由从 Engine 中抽离出来，方便后续的提供动态路由支持；

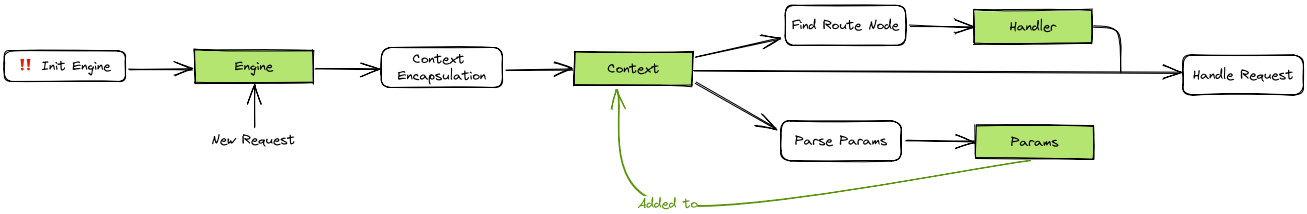
Day 3：TrieTree Router

- 提高路由的解析速度
- 只有叶子结点 `pattern` 才不为空，否则均为空；
- 支持路径参数 `:` 和通配符 `*`；

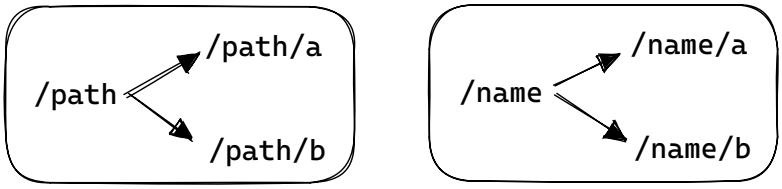
```
        /assets/*filepath: 可以匹配任意 /assets 开头的地址, filepath 表示参数名
        e.g., /assets/css/index.css → {filepath: "css/index.css"}

        /hello/:name: 可以匹配任意 /hello 开头的地址, name 表示参数名
        e.g., /hello/dasein → {name: "dasein"}
```

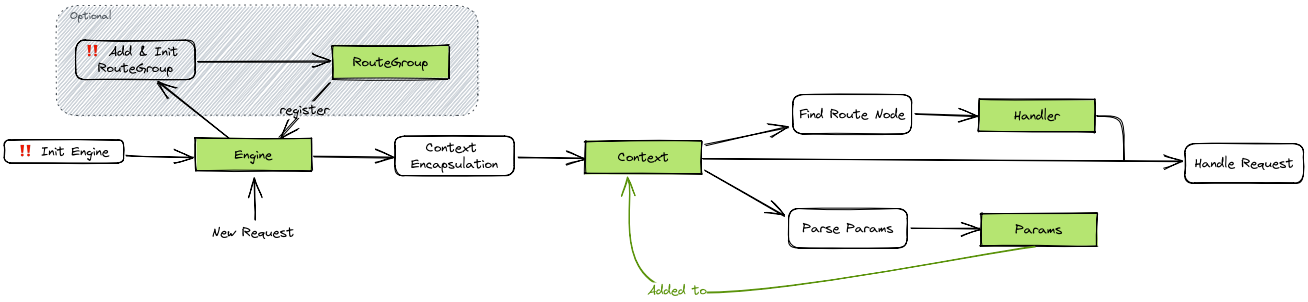
- 使用 Go Convey 进行了单元测试



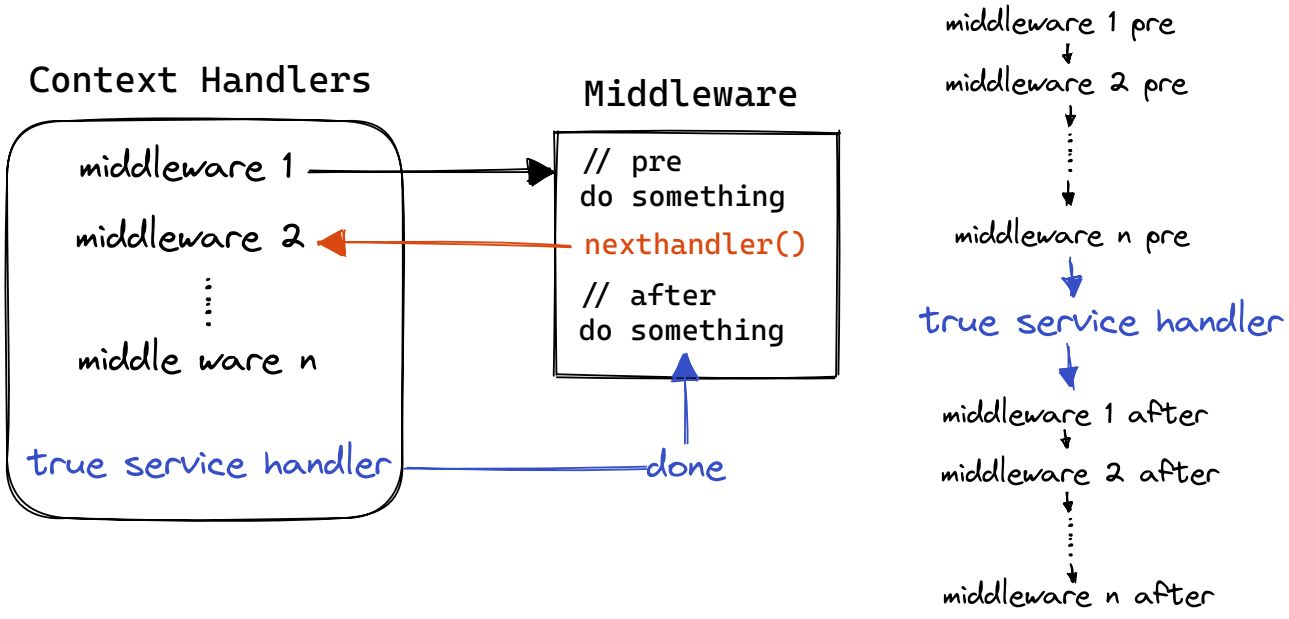
Day 4：Group



- 对路由进行分组，具有相同一级前缀的路由在同一组；
- 每个 Group 都存有指向 Engine 的指针；
- Engine 拥有 Group 的所有方法，且统一管理所有的 Group；



Day 5：MiddleWare



- 中间件：非业务的技术类组件，允许用户自己定义功能嵌入到框架中；
  - 在哪里嵌入中间件?

    - 中间件的输入是什么?
- 日志中间件：
    - 嵌入点：框架接收到请求并初始化 Context 对象之后；
    - 输入：Context；

Day 6：Template

- 支持服务端渲染，支持 JS、CSS 等静态文件；
- 将静态文件放在 `/usr/web` 目录下，那么 `filepath` 的值即为该目录下的文件的相对地址；
- 使用 `html/template` 进行 HTML 渲染；

Day 7：Panic

- 实现错误处理机制
    - panic：会终止当前正在执行的程序；
    - defer：panic 推出前会会执行代码中的 defer 任务后再终止程序；注意 defer 的执行顺序是**逆序** 的，即最后定义的 defer 任务会被最先执行；
    - recover：**recover 只在 defer 任务中生效**，用于捕获 panic 避免程序被终止；
- 将错误处理作为 中间件 来实现；