Martini 的工作方式

喻恒春 2014-1-17

> 本文为转载，原文地址：http://my.oschina.net/achun/blog/192912

匿名字段

因为 golang 中没有继承, golang 中用的方法是匿名字段, 很多 golang 开发者称之为复合, 可是我没有发现官方文档中对此进行正规命名, 用继承这个词肯定不合适,容易对初学者造成理解上的错误, 复合这个词很多初学者不一定知道具体含义. 干脆直接写作扩展自.

Injector 基础

Martini 极好的 Go WEB 框架 一笔带过 Injector 的功能: 通过对被调用(Invoke)函数的参数类型匹, 对函数进行调用.

这里重新列举 Injector 在线文档 中Injector的部分定义.

下面的代码是为了方便把匿名接口列举到了一起, 必须注意事实上Injector是有多个匿名接口复合.实际运用也许会有更多变化.

```
type Injector interface{
    // 设置父Injector
    SetParent(parent Injector)
    // Maps val. 以val的反射Type为key,反射Value为值
    Map(val interface{}) TypeMapper
    // Maps val. 以 ifacePtr 的反射Type为key,val的反射Value为值
    // ifacePtr 正如其名必须是个指针
    MapTo(val interface{}, ifacePtr interface{}) TypeMapper
    // 在已经Maps中匹配 t 返回reflect.Value
    Get(t reflect.Type) reflect.Value
    // 调用函数 f,通过其参数类型定义,在Maps中匹配参数对应的反射值
    // 返回: 执行结果, 错误
    Invoke(f interface{}) ([]reflect.Value, error)
    // 匹配已经Maps的值, 赋值给对应的 val 字段.
    // val 必须是一个*struct, 通过其字段定义中的 tag语法 `inject`
    Apply(val interface{}) error
}
// 默认的Injector实现struct的定义
type injector struct {
    values map[reflect.Type]reflect.Value // Map,MapTo的参数val就保存在这里
    parent Injector
}
```

MapTo 的使用

Injector 的功能很简洁, 很容易理解. values 中同一种类型只保存一个. 而现实中一个函数的参数中可能有多个相同的类型.这需要用到 MapTo 来解决.

```
package main

import (
    "fmt"
    "github.com/codegangsta/inject"
)

// 自定义一个空interface{}, 必须是interface{}类型MapTo才能接受
type SpecialString interface{}

// 原定义Foo(s1,s2 string), 改成
func Foo(s1 string, s2 SpecialString) {
    fmt.Println(s1, s2.(string)) // type assertion
}

func main() {
    ij := inject.New()
    ij.Map("a")
    // 注意第二个参数的固定写法
    ij.MapTo("b", (*SpecialString)(nil))
    ij.Invoke(Foo)
}
```

看上去为了解决这个问题,要多写一些代码. 这不是问题, 多写的这几行代码给你带来的方便更多.

事实上 MapTo 还有其他的应用方法. Martini.go 中的 createContext 方法展示了用法. 这里举例一个比较明显的例子

```
package main
import (
    "fmt"
    "github.com/codegangsta/inject"
)
type Foo interface {
    Foo()
}
type Bar struct{}
func (bar *Bar) Foo() {
}
func Handler(foo Foo) {
    fmt.Println(foo)
}
func main() {
    v := &Bar{}
    ij := inject.New()
    // ij.Map(v) // 错误的用法
    ij.MapTo(v, (*Foo)(nil))
    fmt.Println(ij.Invoke(Handler))
}
```

如果用 Map 会产生 Value not found for type main.Foo, 因为 reflect 中的 Call 方法强制类型匹配

Martini 基础

Martini 在线文档

列举 Martini 中的部分 type

- Route 接口, 一条具体的路由, 由Router 的RESTful方法生成具体对象
- Router 接口, http.Request 路由器,RESTful风格, 在其Handle方法中进行路由匹配
- Context 接口, 扩展自 Injector, 请求上下文, Martini.ServeHTTP 方法生成具体对象
- Routes 接口, MapTo Context.
- Martini 结构, 扩展自 Injector, 是个顶级WEB应用, 衔接上面的各种接口, 完成需求. Action方法设定最后一个handle函数
- ClassicMartini, 扩展自*Martini和Router. 经典在于把 Router.Handle 设置为 Martini 的 action
- Params map[string]string, 保存路由定义中的 name/value, MapTo Context
- ResponseWriter 接口, 扩展自 http.ResponseWriter,http.Flusher,

如何工作

对应上述列举,我们来解释

martini.New()

```
func New() *Martini {
    m := &Martini{inject.New(), []Handler{}, func() {}, log.New(os.Stdout, "[martini] "
    m.Map(m.logger) // Map 了默认的log.Logger
    m.Map(defaultReturnHandler()) // Map 了默认的ReturnHandler 函数对象
    return m
}
```

martini.Classic()

```
func Classic() *ClassicMartini {
    r := NewRouter()
    m := New()
    // 内置的 handlers
    m.Use(Logger())          // 日志, 事实上配合了Context.Next(),
    m.Use(Recovery())        // 精彩的 panic 捕获, 其实也配合Context.Next(),和Logger形成嵌套
    m.Use(Static("public")) // 静态文件
    m.Action(r.Handle)       // 关键, Router.Handle
    return &ClassicMartini{m, r}
}
func Logger() Handler {
    return func(res http.ResponseWriter, req *http.Request, c Context, log *log.Logger)
        start := time.Now()
        log.Printf("Started %s %s", req.Method, req.URL.Path)
        rw := res.(ResponseWriter)
        c.Next()
        log.Printf("Completed %v %s in %v\n", rw.Status(), http.StatusText(rw.Status()),
    }
}
func Recovery() Handler {
```

```
    return func(res http.ResponseWriter, c Context, logger *log.Logger) {
        defer func() {
            if err := recover(); err != nil {
                res.WriteHeader(http.StatusInternalServerError)
                logger.Printf("PANIC: %s\n%s", err, debug.Stack())
            }
        }()
        c.Next()
    }
}
```

Logger 和 Recovery 的代码由于都用了 c.Next(), 所以形成了嵌套调用.

Martini.Action 设置最后的 handler, 多数情况下是 Router.Handle. 本篇暂时不讨论变化.

martini.NewRouter()

```
func NewRouter() Router {
    return &router{notFounds: []Handler{http.NotFound}}
}
```

主要设置了默认的 notFounds handler.可以通过 Router.NotFound 进行设置

Martini.ServeHTTP 方法

```
func (m *Martini) ServeHTTP(res http.ResponseWriter, req *http.Request) {
    m.createContext(res, req).run()
}
func (m *Martini) createContext(res http.ResponseWriter, req *http.Request) *context {
    // 动态创建了 *context  对象, action 变成最后一个handlers
```

```
    c := &context{inject.New(), append(m.handlers, m.action), NewResponseWriter(res), 0
    c.SetParent(m) // 设置Injector的parent
    c.MapTo(c, (*Context)(nil)) // MapTo Context
    c.MapTo(c.rw, (*http.ResponseWriter)(nil)) // MapTo ResponseWriter
    c.Map(req) // Map http.Request
    return c
}
```

context.run()

```
func (c *context) run() {
    for c.index < len(c.handlers) {
        _, err := c.Invoke(c.handlers[c.index]) // Invoke了所有的 handlers.
        if err != nil {
            panic(err)
        }
        c.index += 1 // 很有用的计数器, 配合 Next 方法会产生一些其他用法

        if c.Written() { // break for 条件
            return
        }
    }
}
```

关于 Context.Next() 的技巧,参考 martini.Recovery().

如果你不使用 ClassicMartini, 那么你需要自己通过 Martini.Use/Martini.Action 控制 handlers.

Router.Handle

```
func (r *router) Handle(res http.ResponseWriter, req *http.Request, context Context) {
    for _, route := range r.routes {
        ok, vals := route.Match(req.Method, req.URL.Path)
        if ok { // 路由匹配成功
            params := Params(vals)
            context.Map(params)
            r := routes{}
            context.MapTo(r, (*Routes)(nil)) // 为支持 Routes.URLFor 做准备
            _, err := context.Invoke(route.Handle) // route.Handle 内部 Invoke 了用户定义的
            if err != nil {
                panic(err)
            }
            return
        }
    }

    // no routes exist, 404 // 路由匹配失败
    c := &routeContext{context, 0, r.notFounds} // 设置 handlers 为 notFounds
    context.MapTo(c, (*Context)(nil))
    c.run() // 内部 Invoke notFounds
}
```

Route 和 Routes 暴露出的接口只有URLWith和URLFor, Context. URLFor 比较有趣, 提供了更多变化的可能, 有时间单独介绍.

Martini 没有对 Route 对象进行Map/MapTo. 到不是 Martini 忘记了做了. 而是 Router 的RESTful 方法返回的就是 Route, 如果需要 Map , 应该由应用来完成.

总结

Martini 提供了 martini.Classic() 来支持常规的应用场景. 如果你有自己特殊的需求, 那你需要自己控制 handlers, 把Martini对象和Router对象联系起来.注意以下几点:

Action 方法的作用, 通常应该是 Router.Handle
martini.Recovery() 捕获 panic 的技巧
记得保证 handles 执行的时候要预先把参数用 Context 的 Map/MapTo 准备好

并发安全问题

对于WEB开发,Handler,Router多数都是固定, 一般不会在运行期动态改变, Context 是在具体的请求中动态生成的, 通常也不必考虑并发问题. Martini 依赖的 Injector 用了map, Injector 对map的操作没有考虑并发安全. 因此并发时与 Injector 相关 map 单纯的读并发问题应该是安全的. 也就是说 Injector 是非并发安全的, 为了保证并发 map 安全, martini 应用在 server 运行期不要使用 Martini 对象进行 Map/MapTo/Get 这样的操作.

martini-contrib 中有一些足够好的package可以作为非常好的例子, 比如 模板渲染 render, sessions,binding, strip的代码中有类似子路由的用法. 这里就不再copy代码了.

如果你担心默认Router正则的效率或者有复杂的需求, 那类似子路由的用法你可以参考.

Martini的核心Injector简直是为WEB场景量身打造的. WEB场景中确实存在一个类型的控制变量只有一份的特点.

关于我们

Golang 中国是由 Go 语言中文爱好者组成的技术社区，致力于 Go 语言在中国的发展与传播。

如果您希望分享您的优秀博文，并借名发表，可以通过下方邮箱联系我们。

联系邮箱：joe2010xtmf#163.com

关注官方微博

名人博客推荐

订阅博客更新

发布招聘

如果您的团队需要 Go 语言方面的人才，可以通过我们这个平台发布招聘信息，因为这里汇聚了大量精英。

人才招聘页面

文章列表

最近发表文章：

Go 语言中的方法，接口和嵌入类型
Gopm 快速入门
Go 语言的国际化支持(资源文件翻译)
Read Go - Split Stack
golang: 类型转换和类型断言
golang: 详解 interface 和 nil
Go Slice 机制解析
go build 命令是如何工作的？
go 时间格式风格详解
Martini 的工作方式

浏览所有文章