Tigo框架使用文档

Tigo是一个使用Go语言开发的web框架。

安装

```
go get github.com/karldoenitz/Tigo/...
```

示例

Hello Tigo

```
package main

import "github.com/karldoenitz/Tigo/TigoWeb"

// handler
type HelloHandler struct {
    TigoWeb.BaseHandler
}

func (helloHandler *HelloHandler)Get() {
    helloHandler.ResponseAsHtml("<p1 style='color: red'>Hello Tigo!</p1>")
}

// url路由配置
var urls = map[string]interface{}{
    "/hello-tigo": &HelloHandler{},
}
```

```
// 主函数
func main() {
    application := TigoWeb.Application{
        IPAddress:  "127.0.0.1",
        Port:       8888,
        UrlPattern: urls,
    }
    application.Run()
}
```

编译

打开终端，进入代码目录，运行如下命令：

```
go build main.go
```

运行

编译完成后，会有一个可执行文件`main`，运行如下命令：

```
./main
```

终端会有如下显示：

```
INFO: 2018/07/09 15:02:36 Application.go:22: Server run on: 0.0.0.0:8888
```

打开浏览器访问地址`http://127.0.0.1:8888/hello-tigo`，就可以看到Hello Tigo。

Tigo是一款用go开发的web应用框架，基于net/http库实现，目的是用来快速搭建restful服务。

API目录：

        - type GlobalConfig
            - func Init
        - utils
            - func Encrypt
            - func Decrypt
            - func InitGlobalConfig
- logger
        - Demo
        - structure
            - type LogLevel
        - functions
            - func SetLogPath
            - func InitLoggerWithConfigFile
            - func InitLoggerWithObject
            - func InitTrace
            - func InitInfo
            - func InitWarning
            - func InitError
- request
        - type Response
            - ToContentStr
        - functions
            - func Request
            - func Get
            - func Post
            - func Put
            - func Patch
            - func Head
            - func Options
            - func Delete
- binding
        - functions
            - func ParseJsonToInstance
            - func ValidateInstance

Tigo.TigoWeb

TigoWeb是Tigo框架中的核心部分，Handler、URLpattern以及Application三大核心组件包含于此。

type BaseHandler

```
type BaseHandler struct {
    ResponseWriter  http.ResponseWriter
    Request         *http.Request
}
```

`BaseHandler`是一切handler的父结构体，开发者开发的handler必须继承此结构体。

func (*BaseHandler)InitHandler

```
func (baseHandler *BaseHandler)InitHandler(responseWriter http.ResponseWriter,
 request *http.Request)
```

`InitHandler`方法是初始化结构体必须要使用的方法，所有的Handler中的Handle方法必须调用此方法。

func (*BaseHandler)GetJsonValue

```
func (baseHandler *BaseHandler)GetJsonValue(key string) (interface{})
```

`GetJsonValue`方法是根据key获取客户端传递的json对象。客户端发送的请求的Content-Type必须是application/json。

func (*BaseHandler)GetParameter

```
func (baseHandler *BaseHandler)GetParameter(key string) (value *JsonParams)
```

`GetParameter`方法是根据key获取客户端传递的参数值。

- 也可以从`baseHandler.Request`对象中调用`Request`的相关函数获取参数；
- 可以从URL上获取参数值，或是form中；如果http的body是json，也可以从中解析出参数值。

func (*BaseHandler)GetHeader

```
func (baseHandler *BaseHandler)GetHeader(name string) (value string)
```

`GetHeader`方法是根据name获取http的header值。

func (*BaseHandler)SetHeader

```
func (baseHandler *BaseHandler)SetHeader(name string, value string)
```

`SetHeader`方法是根据name设置http的header值。

func (*BaseHandler)GetCtxVal

```
func (baseHandler *BaseHandler)GetCtxVal(key string) interface{}
```

`GetCtxVal`方法是根据key从http上下文中获取值。

func (*BaseHandler)SetCtxVal

```
func (baseHandler *BaseHandler) SetCtxVal(key string, val interface{})
```

`SetCtxVal`方法是根据key设置在http上下文中设置值。

func (*BaseHandler)GetCookie

```
func (baseHandler *BaseHandler)GetCookie(name string) (value string)
```

`GetCookie`方法是根据name获取cookie值。

func (*BaseHandler)SetCookie

```
func (baseHandler *BaseHandler)SetCookie(name string, value string)
```

`SetCookie`方法是根据name设置cookie值。

func (*BaseHandler)GetSecureCookie

```
func (baseHandler *BaseHandler)GetSecureCookie(name string, key ...string) (val
ue string)
```

`GetSecureCookie`方法是用来获取加密的cookie值，key为解密所需要用到的密钥，key可以不填，也可以在configuration配置文件中配置。

- 当configuration文件和函数参数中都设置了key，则以函数中设置的key为准；
- 当configuration文件和函数参数中都未设置key，则依然会进行解密。

func (*BaseHandler)SetSecureCookie

```
func (baseHandler *BaseHandler)SetSecureCookie(name string, value string, key
 ...string)
```

`SetSecureCookie`方法是用来给客户端设置一个加密cookie，key为加密所需要用到的密钥，key可以不填，也可以在configuration配置文件中配置。

- 当configuration文件和函数参数中都设置了key，则以函数中设置的key为准；
- 当configuration文件和函数参数中都未设置key，则依然会进行加密。

func (*BaseHandler)GetCookieObject

```
func (baseHandler *BaseHandler)GetCookieObject(name ...string) (Cookie, error)
```

`GetCookieObject`方法是用来根据name获取指定的cookie对象，返回值类型为`Cookie`。

func (*BaseHandler)SetCookieObject

```
func (baseHandler *BaseHandler)SetCookieObject(cookie Cookie)
```

`SetCookieObject`方法接收一个`Cookie`对象，为客户端设置cookie。

func (*BaseHandler)ClearCookie

```
func (baseHandler *BaseHandler)ClearCookie(name string)
```

`ClearCookie`方法是根据指定的name清除cookie。

func (*BaseHandler)ClearAllCookie

```
func (baseHandler *BaseHandler)ClearAllCookie()
```

`ClearAllCookie`方法是用来清空所有的cookie的。

func (*BaseHandler)Redirect

```
func (baseHandler *BaseHandler)Redirect(url string, expire ...time.Time)
```

`Redirect`方法是将当前handler所挂载的URL重定向到另一个URL地址，expire为客户端过期时间，如果不填写expire，则会使用客户端默认过期时间。

func (*BaseHandler)RedirectPermanently

```
func (baseHandler *BaseHandler)RedirectPermanently(url string)
```

`RedirectPermanently`方法是将当前handler所挂载的URL永久性重定向到另一个URL地址。

func (baseHandler *BaseHandler)Render

```
func (baseHandler *BaseHandler)Render(data interface{}, templates ...string)
```

`Render`方法是根据数据和html模板渲染网页，并将渲染后的结果以网页形式相应给客户端。

参数解析：

- data：任意类型结构体实例；
- templates：需要渲染的模板文件名称。

注意：

如果在配置文件中没有配置template的路径，则此函数选择模板时将会使用相对路径。

func (*BaseHandler)ResponseAsHtml

```
func (baseHandler *BaseHandler)ResponseAsHtml(result string)
```

`ResponseAsHtml`方法是将传入的字符串以html文本类型响应给客户端。

func (*BaseHandler)ResponseAsText

```
func (baseHandler *BaseHandler)ResponseAsText(result string)
```

`ResponseAsText`方法是将传入的字符串以text文本类型响应给客户端。

func (*BaseHandler)ResponseAsJson

```
func (baseHandler *BaseHandler)ResponseAsJson(response interface{})
```

`ResponseAsJson`方法是将传入的对象转换成json字符串，然后响应给客户端，如果转换失败则会向客户端响应空字符串。

func (*BaseHandler)ToJson

```
func (baseHandler *BaseHandler)ToJson(response interface{}) (result string)
```

`ToJson`方法是将一个对象转换成json字符串。如果转换失败则会返回空字符串。

func (*BaseHandler)DumpHttpRequestMsg

```
func (baseHandler *BaseHandler)DumpHttpRequestMsg(logLevel int) (result string)
```

`DumpHttpRequestMsg`方法是将此次请求的http报文输出到终端或log文件中。

参数logLevel如下：

- 1: 将http报文输出到trace级别日志中 // logger.TraceLevel
- 2: 将http报文输出到info级别日志中 // logger.InfoLevel
- 3: 将http报文输出到warning级别日志中 // logger.WarningLevel
- 4: 将http报文输出到error级别日志中 // logger.ErrorLevel
- others: 将http报文输出到控制台

func (*BaseHandler)CheckJsonBinding

```
func (baseHandler *BaseHandler) CheckJsonBinding(obj interface{}) error
```

`CheckJsonBinding`校验客户端发送的json是否符合要求。

tag如下：

- required: 是否需要校验，true为校验，false为忽略此字段的校验
- default: 设置字段的默认值，只有required设置为true时此tag生效
- regex: 正则表达式匹配，只有required设置为true时此tag生效

  示例：

```
type TestParamCheckHandler struct {
    TigoWeb.BaseHandler
}

func (t *TestParamCheckHandler)Post() {
    params := struct{
        Username string `json:"username" required:"true" regex:"^[0-9a-zA-Z_]
{1,}$"`
        Password string `json:"password" required:"true"`
        Age      int    `json:"age" required:"true" default:"18"`
    }{}
    if err := t.CheckJsonBinding(&params); err != nil {
        t.ResponseAsJson(struct{
            Msg string
        }{err.Error()})
        return
    }
    // 校验通过后的具体逻辑
}
```

Post的json：

```
{
    "username": "wo&ni",
    "password": "tihs si wrodpssa"
} // 此json校验后返回"username regex can not match"
{
    "username": "wo_ni",
} // 此json校验后会返回"password is required"
// 以上两个json都没有填写age，但不会报错，age会被设置为默认值18
```

其他规则可参考`Tigo.binding.ValidateInstance`

type UrlPattern

```
type UrlPattern struct {
    UrlMapping map[string] interface{Handle(http.ResponseWriter, *http.Reques
t)}
}
```

URL路由设置，使用这个结构体在应用中配置URL与对应的handler。

func (urlPattern *UrlPattern)AppendUrlPattern

```
func (urlPattern *UrlPattern)AppendUrlPattern(uri string, v interface{Handle(ht
tp.ResponseWriter, *http.Request)})
```

此方法是向指定URL上挂载一个Handler。

func (urlPattern *UrlPattern)Init

```
func (urlPattern *UrlPattern)Init()
```

初始化URL映射。

type Application

```
type Application struct {
    IPAddress  string      // IP地址
    Port       int         // 端口
    UrlPattern UrlPattern  // url路由配置
    ConfigPath string      // 全局配置
}
```

Application结构体是启动http服务的入口。

- IPAddress：服务绑定的IP地址，可以在configuration中配置，若在configuration中配置了则以configuration中配置的为主；
- Port：端口，可以在configuration中配置，若在configuration中配置了则以configuration中配置的为主；
- UrlPattern：路由配置；
- ConfigPath：配置文件的路径。

func (application *Application)Run

```
func (application *Application)Run()
```

此方法用来启动http服务，如果在configuration中配置了https的密钥和证书，服务则会以https方式启动。

type Cookie

```
type Cookie struct {
    Name        string
    Value       string

    IsSecurity  bool      // 是否对cookie值进行加密
    SecurityKey string    // 加密cookie用到的key

    Path        string    // 可选
    Domain      string    // 可选
    Expires     time.Time // 可选
```

```
    RawExpires  string    // 只有在读取Cookie时有效

    // MaxAge=0 表示未指定“Max-Age”属性
    // MaxAge<0 表示现在删除cookie，相当于'Max-Age：0'
    // MaxAge>0 表示Max-Age属性存在并以秒为单位给出
    MaxAge      int
    Secure      bool
    HttpOnly    bool
    Raw         string
    Unparsed  []string    // 原始文本中未解析的属性值
}
```

cookie结构体，用此结构体进行cookie处理。

func (cookie *Cookie)GetCookieEncodeValue

```
func (cookie *Cookie)GetCookieEncodeValue()(result string)
```

使用此方法获取cookie的加密值。

func (cookie *Cookie)GetCookieDecodeValue

```
func (cookie *Cookie)GetCookieDecodeValue()(result string)
```

使用此方法获取cookie的解密值。

func (cookie *Cookie)ToHttpCookie

```
func (cookie *Cookie)ToHttpCookie()(http.Cookie)
```

使用此方法将Cookie对象转换为http.Cookie对象。

func (cookie *Cookie)ConvertFromHttpCookie

```
func (cookie *Cookie)ConvertFromHttpCookie(httpCookie http.Cookie)
```

使用此方法将http.Cookie对象转换为Cookie对象。

func (cookie *Cookie)SetSecurityKey

```
func (cookie *Cookie)SetSecurityKey(key string)
```

使用此方法为cookie对象设置加密key。

type BaseResponse

```
type BaseResponse struct {

}
```

继承此结构体的对象可以通过`func (baseHandler *BaseHandler)ResponseAsJson(response Response)`方法，序列化为json字符串传递给客户端。

func (baseResponse *BaseResponse)Print

```
func (baseResponse *BaseResponse)Print()
```

使用此方法可以打印当前Response对象到控制台中。

func (baseResponse *BaseResponse)ToJson

```
func (baseResponse *BaseResponse)ToJson() (string)
```

使用此方法将当前Response对象序列化为json字符串。

type GlobalConfig

```
type GlobalConfig struct {
    IP       string          `json:"ip"`       // IP地址
    Port     int             `json:"port"`     // 端口
    Cert     string          `json:"cert"`     // https证书路径
```

```
    CertKey   string           `json:"cert_key"`   // https密钥路径
    Cookie    string           `json:"cookie"`     // cookie加密解密的密钥
    Log       logger.LogLevel  `json:"log"`        // log相关属性配置
}
```

func (globalConfig *GlobalConfig)Init

```
func (globalConfig *GlobalConfig)Init(configPath string)
```

根据配置文件初始化全局配置对象。

解析：

- IP：配置服务地址
- Port：配置服务绑定的端口
- Cert：https证书地址
- CertKey：https密钥
- Cookie：cookie加密解密使用的密钥
- Log：log相关属性的配置，Tigo.logger.LogLevel结构体的实例

配置文件configuration.json示例如下：

```
{
    "ip": "0.0.0.0",
    "port": 8888,
    "cert": "/home/work/certfile.ext"
    "cert_key": "/home/work/certkeyfile.ext",
    "log_path": "/home/work/log/server_run.log",
    "cookie": "thisiscookiekey"
}
```

type ReqParams

```
type JsonParams struct {
    Value interface{}
}
```

`BaseHandler.GetParameter`返回此类型的实例，`Value`为`interface`类型，可以按照具体需求转换为自己需要的类型。

- `ToBool`：将Value转换为bool类型
- `ToFloat64`：将Value转换为float64类型
- `ToInt64`：将Value转换为int64类型
- `ToString`：将Value转换为string类型
- `To`：将Value转换为自定义类型

utils

加密方法

```
func Encrypt(src[]byte, key []byte) string
```

使用此方法对字符数组进行aes加密。

解密方法

```
func Decrypt(src[]byte, key []byte) ([]byte)
```

使用此方法对已加密的字符数组进行aes解密。

初始化全局变量方法

```
func InitGlobalConfig(configPath string)
```

使用此方法初始化全局变量。

Tigo.logger

使用此模块打印log。

Demo

在Tigo框架中使用log模块，只要按照如下示例编写代码即可：

```
// 在Tigo框架中使用logger模块
package main

import (
    "net/http"
    "github.com/karldoenitz/Tigo/TigoWeb"
    "github.com/karldoenitz/Tigo/logger"
)

type HelloHandler struct {
    TigoWeb.BaseHandler
}

func (helloHandler *HelloHandler)Get() {
    logger.Info.Printf("info data: %s", "test") // 此处打印log
    helloHandler.ResponseAsHtml("<p1 style='color: red'>Hello Tigo!</p1>")
}

var urls = map[string]interface{}{
    "/hello-tigo": &HelloHandler{},
}

func main() {
    application := TigoWeb.Application{
        UrlPattern: urls,
        ConfigPath: "./configuration.json",  // 此处配置文件，如果不适用配置文件，可以在代码中初始化LogLevel对象，使用该对象进行logger模块初始化。
    }
    application.Run()
}
```

`configuration.json`文件内容如下：

```
{
  "cookie": "TencentCode",
  "ip": "0.0.0.0",
```

```
  "port": 8080,
  "log": {
    "trace": "stdout",  // trace的内容只在终端输出，不在文件内保留
    "info": "/Users/karllee/Desktop/run-info.log",  // info的内容存在run-info.log
文件中
    "warning": "/Users/karllee/Desktop/run.log",  // warning与error的日志存在同一
个文件内
    "error": "/Users/karllee/Desktop/run.log",
    "time_roll": "H*2"  // 表示每两个小时切分一次日志
  }
}
```

以上为在Tigo框架中使用logger模块，如果想在第三方代码中使用logger模块，而不是在Tigo中，则可以参考`func (globalConfig *GlobalConfig)Init(configPath string)`方法，使用LogLevel或是配置文件初始化logger模块。

Structure

log模块所包含的结构体。

type LogLevel

```
// log分级结构体
//   - Trace    跟踪
//   - Info     信息
//   - Warning  预警
//   - Error    错误
//   - TimeRoll 日志切分时长
// discard: 丢弃，stdout: 终端输出，文件路径表示log具体输出的位置
type LogLevel struct {
    Trace    string   `json:"trace"`
    Info     string   `json:"info"`
    Warning  string   `json:"warning"`
    Error    string   `json:"error"`
    TimeRoll string   `json:"time_roll"`
}
```

初始化此结构体，将此结构体作为参数传入`InitLoggerWithObject`中，初始化logger模块。

TimeRoll：

- D：表示按天切分日志，例如："D*6"则表示每6天切分一次日志
- H：表示按小时切分日志，例如："H*6"则表示每6小时切分一次日志
- M：表示按分钟切分日志，例如："M*6"则表示每6分钟切分一次日志
- S：表示按秒切分日志，例如："S*6"则表示每6秒切分一次日志

logger模块内置方法

func SetLogPath

设置log文件的路径

```
func SetLogPath(logPath string)
```

示例：

```
import "github.com/karldoenitz/Tigo/logger"

logger.Info.Printf("It is a test...")
logger.Warning.Printf("warning!")
logger.Error.Printf("ERROR!!!")
```

注意：使用此方法会使原先的log配置失效。

func InitLoggerWithConfigFile

```
func InitLoggerWithConfigFile(filePath string)
```

根据配置文件初始化logger模块。

func InitLoggerWithObject

```
func InitLoggerWithObject(logLevel LogLevel)
```

根据LogLevel实例初始化logger模块。

func InitTrace

```
func InitTrace(level string)
```

初始化Trace实例。

参数解释：

- discard：不处理；
- stdout： 终端输出，不打印到文件；
- 文件具体路径：存储log的文件的路径。

func InitInfo

```
func InitInfo(level string)
```

初始化Info实例。

参数解释：

- discard：不处理；
- stdout： 终端输出，不打印到文件；
- 文件具体路径：存储log的文件的路径。

func InitWarning

```
func InitWarning(level string)
```

初始化Warning实例。

参数解释：

- discard：不处理；
- stdout： 终端输出，不打印到文件；
- 文件具体路径：存储log的文件的路径。

func InitError

```
func InitError(level string)
```

初始化Error实例。

参数解释：

- discard：不处理；
- stdout： 终端输出，不打印到文件；
- 文件具体路径：存储log的文件的路径。

Tigo.request

request模块是Tigo框架中用来进行http request请求的模块，可使用此模块内的方法对目标连接发送http请求。

type Response

```
type Response struct {
    *http.Response
    Content []byte
}
```

HTTP请求返回的对象。

func (response *Response)ToContentStr

```
func (response *Response)ToContentStr() string
```

将Response对象的Content转换为string类型。

request模块内置方法

func Request

```
func Request(method string, requestUrl string, postParams map[string]interface
{}, headers ...map[string]string) (*Response, error)
```

向一个连接发送请求。

func Get

```
func Get(requestUrl string, headers ...map[string]string) (*Response, error)
```

向一个连接发送Get请求。

func Post

```
func Post(requestUrl string, postParams map[string]interface{}, headers ...map[
string]string) (*Response, error)
```

向一个连接发送Post请求。

func Put

```
func Put(requestUrl string, postParams map[string]interface{}, headers ...map[s
tring]string) (*Response, error)
```

向一个连接发送Put请求。

func Patch

```
func Patch(requestUrl string, postParams map[string]interface{}, headers ...map
[string]string) (*Response, error)
```

向一个连接发送Patch请求。

func Head

```
func Head(requestUrl string, headers ...map[string]string) (*Response, error)
```

向一个连接发送Head请求。

func Options

```
func Options(requestUrl string, headers ...map[string]string) (*Response, erro
r)
```

向一个连接发送Options请求。

func Delete

```
func Delete(requestUrl string, headers ...map[string]string) (*Response, error)
```

向一个连接发送Delete请求。

Tigo.binding

binding模块是Tigo框架中用来校验结构体实例是否符合规范工具包。

binding模块内置方法

func ParseJsonToInstance

```
func ParseJsonToInstance(jsonBytes []byte, obj interface{}) error
```

将json的byte数组转化成对象，并根据tag进行校验。

func ValidateInstance

```
func ValidateInstance(obj interface{}) error
```

根据tag对结构体实例进行校验。

```
type Company struct {
    Name string `json:"name" required:"false"`
    Addr string `json:"name" required:"false"`
}
```

```
type Boss struct {
    Name    string  `json:"name" required:"true"`
    Age     int     `json:"age" required:"true" default:"18"`
    Company Company `json:"company" required:"true"`
}
/*以上这种方式OK👌*/

type Stuff struct {
    Name    string   `json:"name" required:"true"`
    Age     int      `json:"age" required:"true" default:"18"`
    Company *Company `json:"company" required:"true"`  // OK
}
/*以上这种方式OK👌*/

type Others struct {
    Name    string   `json:"name" required:"true"`
    Age     *int     `json:"age" required:"true" default:"18"`  // Not Support
    Company Company  `json:"company" required:"true"`
}
/*以上这种方式OK👌*/
```