构建API服务器二

本示例介绍跨域设置，RESTful服务编写，jwt设置与登录信息获取,用户权限验证。

接口	功能	说明
/member/login	用户登录	登录成功后，使用jwt返回登录状态
/product	添加，修改，查询，删除	使用RESTful风格实现

知识点:

- 跨域配置
- jwt配置
- 编写RESTful服务
- 登录状态设置，获取
- 权限验证

1. 服务配置

```
package main

func (api *apiserver) config() {
        api.IsDebug = true
        api.Conf.API.SetSubConf("header", `
        {
                "Access-Control-Allow-Origin": "*",
                "Access-Control-Allow-Methods": "GET,POST,PUT,DELETE,PATCH,OPTIONS",
                "Access-Control-Allow-Headers": "X-Requested-With,Content-Type",
                "Access-Control-Allow-Credentials": "true"
        }
            `)
    api.Conf.API.SetSubConf("auth", `
        {
                "jwt": {
                        "exclude": ["/member/login"],
                        "expireAt": 36000,
                        "mode": "HS512",
                        "name": "__jwt__",
                        "secret": "45d25cb71f3bee254c2bc6fc0dc0caf1"
                }
        }
                `)
}
```

> header可设置跨域信息或其它http头信息，`"Access-Control-Allow-Origin": "*"`，系统会修改响应头的 `Access-Control-Allow-Origin` 值为请求域名

> `auth.jwt.exclude` 无需登录验证的服务名称

> `mode` jwt加密方式，支持: `HS256` ， `HS384` ， `HS512` ， `ES256` ， `ES384` ， `ES512` ， `RS256` ， `RS384` ， `RS512`

> `expireAt` 超时时间，超时后返回 `403` ,客户端需处理此状态码，并引导用户重新登录

> 可通过参数 `source` 设置jwt保存到cookie中或header中,
> 值： `header` , `cookie` , `HEADER` , `COOKIE` , `H` ,默认通过cookie保存jwt信息

2. 检查用户登录状态

```go
package main

import (
	"fmt"

	"github.com/micro-plat/hydra/component"

	"github.com/micro-plat/hydra/context"
)

//handling 处理jwt排除页面，保存登录对象，验证用户权限
func (api *apiserver) handling() {
	//每个请求执行前执行
	api.MicroApp.Handling(func(ctx *context.Context) (rt interface{}) {

		//获取jwt
		jwt, err := ctx.Request.GetJWTConfig() //获取jwt配置
		if err != nil {
			return err
		}
		for _, u := range jwt.Exclude { //排除请求
			if u == ctx.Service {
				return nil
			}
		}

		//从jwt中获取用户信息，并转换为mem.LoginState,保存到ctx中
		var m mem.LoginState
		if err = ctx.Request.GetJWT(&m); err != nil {
			return context.NewError(context.ERR_FORBIDDEN, err)
		}
		if err = mem.Save(ctx, &m); err != nil {
			return err
		}


		//检查用户权限
		tags := r.GetTags(ctx.Service)
		menu := xmenu.Get(ctx.GetContainer().(component.IContainer))
		for _, tag := range tags {
			if tag == "*" {
				return nil
			}
			if err = menu.Verify(m.UserID, m.SystemID, tag, ctx.Request.GetM
				return nil
			}
		}
		return context.NewError(context.ERR_NOT_ACCEPTABLE, fmt.Sprintf("没有权限
	})
}
```

> mem.Save，mem.Get用于保存和或需用户登录信息，此信息缓存到输入的context中

2. 编写登录接口

```
package member

import (
        "fmt"

        "github.com/micro-plat/hydra/component"
        "github.com/micro-plat/hydra/context"
        "github.com/micro-plat/hydra/quickstart/demo/apiserver11/modules/member")

//LoginHandler 用户登录对象
type LoginHandler struct {
        c      component.IContainer
        m      member.IMember
}

//NewLoginHandler 创建登录对象
func NewLoginHandler(container component.IContainer) (u *LoginHandler) {
        return &LoginHandler{
                c:      container,
                m:      member.NewMember(container),
        }
}


//Handle 用户登录
func (u *LoginHandler) Handle(ctx *context.Context) (r interface{}) {
        ctx.Log.Info("-------用户登录---------")
        ctx.Log.Info("1. 检查输入参数")
        if err := ctx.Request.Check("username", "password"); err != nil {
                return context.NewError(context.ERR_NOT_ACCEPTABLE, err)
        }
        ctx.Log.Info("2. 开始登录")
        member, err := u.m.Login(ctx.Request.GetString("username"),ctx.Request.GetString
        if err != nil {
                return err
        }

        ctx.Log.Info("3. 设置jwt,返回数据")
        ctx.Response.SetJWT(member)
        return member
}
```

> ctx.Response.SetJWT 将输入对象放入 `jwt` ,http response会自动生成jwt参数

3. RESTfull服务

```
package order

import (
    "github.com/micro-plat/hydra/component"
    "github.com/micro-plat/hydra/context"
)

type OrderHandler struct {
    container component.IContainer
}

func NewOrderHandler(container component.IContainer) (u *OrderHandler) {
    return &OrderHandler{
        container: container,
    }
}

//GetHandle 查询
func (u *OrderHandler) GetHandle(ctx *context.Context) (r interface{}) {
 }

 //PostHandle 新增
func (u *OrderHandler) PostHandle(ctx *context.Context) (r interface{}) {
         member:=mem.Get(ctx)
 }

//PutHandle 修改
 func (u *OrderHandler) PutHandle(ctx *context.Context) (r interface{}) {
 }

//DeleteHandle 删除
 func (u *OrderHandler) DeleteHandle(ctx *context.Context) (r interface{}) {
 }
```

> mem.Get可直接获取在handling中通过mem.Save保存的用户信息

4. 服务注册

```
package main

import (
        "github.com/micro-plat/hydra/component"
        "github.com/micro-plat/hydra/quickstart/demo/apiserver12/services/product"
)

//init 检查应用程序配置文件，并根据配置初始化服务
func (api *apiserver) init() {
        api.Initializing(func(c component.IContainer) error {
                //检查db配置是否正确
                if _, err := c.GetDB(); err != nil {
                        return err
                }
                //检查消息队列配置

                //拉取应用程序配置

                return nil
        })

        //服务注册
        api.Micro("/product", order.NewProductHandler)
}
```

> `/product` 服务可使用 `GET`，`POST`，`PUT`，`DELETE` 请求，系统自动执行对应的 `Handle`

5. 保存用户信息

```
package member

import (
        "encoding/json"
        "github.com/micro-plat/hydra/context"
)
//Save 保存member信息
func Save(ctx *context.Context, m *LoginState) error {
        //不允许同一个账户多终端登录
        //检查用户是否已锁定、禁用
        ctx.Meta.Set("__member_info__", m)
        return nil
}

//Get 获取member信息
func Get(ctx *context.Context) *LoginState {
        v, _ := ctx.Meta.Get("__member_info__")
        if v == nil {
                return nil
        }
        return v.(*LoginState)
}
```

> 保存用户状态时可检查用户状态，非正常状态返回错误

总结:

- 通过app.Conf设置 `header` ， `jwt`
- 在app.Handling中处理登录验证、用户状态检查、保存登录信息、处理用户权限
- 调用 `ctx.Response.SetJWT` ,即可设置登录信息到http响应头中
- RESTfull服务只需编写 `GetHandle`，`PostHandle`，`PutHandle`，`DeleteHandle` ,使用注册的服务名发送对应的 `Get`，`POST`，`PUT`，`DELETE` 请求即可
- 注册服务时指定 `tag` ,将 `tag` 、用户信息与权限配置信息匹配，实现权限验证功能