安全认证

本示例介绍常用的几种跨域身份验证解决方案，包括jwt,数字摘要，远程服务

知识点:

- JSON Web Token(JWT) 认证及使用
- 数字摘要(MD5,SHA)验证
- 远程服务验证

一、 JWT认证

配置 `auth` > `jwt` 节点，除排除的页面外，其它页面都会被服务器拦截进行 `jwt` 校验， `jwt` 数字摘要不正确或已过期则会自动返回 `403` ,验证通过的请求才会执行对应的服务。

1. 参数配置

```
package main

func (api *apiserver) config() {
        api.Conf.API.SetAuthes(
                conf.NewAuthes().
                        WithJWT(
                                conf.NewJWT(
                                        "__jwt__", "HS512",
                                        "45d25cb71f3bee254c2bc6fc0dc0caf1",
                                        36000, "/member/login")))
}
```

参数顺次为:

> `name` jwt存储在 `cookie` 或 `header` 中名称
> `mode` jwt加密方式，支持: `HS256` ， `HS384` ， `HS512` ， `ES256` ， `ES384` ， `ES512` ， `RS256` ， `RS384` ， `RS512`
> `secret` jwt加密密钥
> `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
> `auth.jwt.exclude` 无需jwt验证的服务名称

可选参数

> 可通过参数 `.WithHeaderStore()` 指定将jwt信息存储到 `header` 中，未指定存储到 `cookie` 中

2. 设置 `jwt` 值

```
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("password"
        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. 保存登录信息

```
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
func (api *apiserver) handling() {
        //每个请求执行前执行
        api.MicroApp.Handling(func(ctx *context.Context) (rt interface{}) {

                if b, err := ctx.Request.SkipJWTExclude(); b || err != nil {
                    return err
            }

                //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
        }
        return nil
        })
}
```

> 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

二、数字摘要(MD5,SHA)验证

配置 `auth > fixed-secret` 节点可进行固定密钥的数字摘要验证。默认所有请求都会进行验证

1. 参数配置

```
package main

func (api *apiserver) config() {
        api.Conf.API.SetAuthes(
                conf.NewAuthes().
                        WithFixedSecretSign(conf.NewFixedSecretAuth(
                                "45d25cb71f3bee254c2bc6fc0dc0caf1", "md5")))
}
```

或指定服务进行验证

```
func (api *apiserver) config() {
    app.Conf.API.SetAuthes(
            conf.NewAuthes().
                WithFixedSecretSign(conf.NewFixedSecretAuth(
                    "45d25cb71f3bee254c2bc6fc0dc0caf1", "sha256").
                    WithInclude("/order/request")))
}
```

> 未通过 `WithInclude` 指定服务，则所有服务都会参与验证

> 验证方式为：所有传入参数按键值组成单个字符串，并进行ascii排序并拼接为一个字符串，并在最后拼接 `secret` ,并进行指定的模式生成摘要，并转成16进制字符串，与传入的 `sign` 进行比较,相同则通过验证。如请求: `order/request?mid=890098&pid=100&timestamp=20190918144523&sign=98709cb71f3bee234c2bc6fc0dc0caf9` ，摘要生成方式为 `sha256` ，则验证方式为:strings.ToUpper(hex(sha256("mid890098pid100mid890098timestamp20190918144523")))==strings.ToUpper(ctx.Request.GetString("sign"))

> 必须传入参数为 `sign` , `timestamp`

验证代码请参考engine.check.sign.go

三、远程服务验证

远程服务验证指通过本服务器或其它服务器提供 `rpc` 服务进行参数验证和参数解密。

```
func (api *apiserver) config() {
    api.Conf.API.SetAuthes(
        conf.NewAuthes().
        WithRemoteAuth(
            conf.NewRemoteAuth("/request/verify@auth.as")))
}
```

或指定服务进行验证

```
func (api *apiserver) config() {
    api.Conf.API.SetAuthes(
        conf.NewAuthes().
        WithRemoteAuth(
            conf.NewRemoteAuth("/request/verify@auth.as")
            .WithInclude("/order/request")))

}
```

> 未通过 `WithInclude` 指定服务，则所有服务参数所会原样发送到 `/request/verify@auth.as"` 服务进行验证

> 远程验证服务可进行参数解密，解密后的结果保存到 `response` 中，本地服务通过 `ctx.Request.Metadata.Get...` 获取解密结果

验证代码请参考engine.check.sign.go