构建 websocket 服务(WS)

本示例介绍如何构建 websocket 服务器，以及通过后端流程推送消息给 websocket 客户端。

1. 客户端通过 websocket 服务登录
2. api 服务向消息队列中放入一条消息
3. 消息订阅服务将消息推送给 websocket 客户端

1. 创建服务器

创建包含有 ws,api,mqc 三种服务的服务器

main.go

```
package main

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

type wsserver struct {
        *hydra.MicroApp
}

func main() {
        app := &wsserver{
                hydra.NewApp(
                        hydra.WithPlatName("mall"),
                        hydra.WithSystemName("wsserver"),
                        hydra.WithServerTypes("ws-mqc-api")),
        }
        app.init()
        app.Start()
}
```

> app.init 用于挂载服务配置，注册等处理

config.dev.go

```
// +build !prod

package main

func (ws *wsserver) config() {
        ws.IsDebug = true
        ws.Conf.API.SetMainConf(`{"address":":9090","trace":true}`)
    ws.Conf.WS.SetMainConf(`{"address":":9087","trace":true}`)

}
```

2. 服务注册

```
package main

import (
        "github.com/micro-plat/hydra/component"
        "github.com/micro-plat/hydra/quickstart/demo/wsserver01/services/order"
)

//init 检查应用程序配置文件，并根据配置初始化服务
func (rpc *wsserver) init() {
        rpc.config()
        rpc.handling()

        rpc.Initializing(func(c component.IContainer) error {
                //检查db配置是否正确
                if _, err := c.GetDB(); err != nil {
                        return err
                }

                return nil
        })

    //服务注册
    rpc.WS("/member/login",member.NewLoginHandler)//用户通过ws服务登录
    rpc.API("/msg/send", msg.NewSendHandler) //消息放入消息队列
    rpc.MQC("/msg/push",msg.NewPushHandler)//消息订阅服务发送消息给指定的ws用户
}
```

3. 构建服务

`services/member.go`

1. 检查用户名密码是否正确
2. 保存用户登录信息，并存储用户 websocket 客户端 uuid

```go
package member

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

type LoginHandler struct {
        container component.IContainer
}

func NewLoginHandler(container component.IContainer) (u *LoginHandler) {
        return &LoginHandler{
                container: container,
        }
}
//Handle api接口发送消息，将消息存入消息队列
func (u *LoginHandler) Handle(ctx *context.Context) (r interface{}) {
    userName:=ctx.Request.String("uname")
    pwd:=ctx.Request.String("pwd")
    uuid:=ctx.Request.GetUUID()
    //处理登录逻辑，保存ws客户端uuid,用于后续消息推送
    return "success"
}
```

services/msg_send.go

1. 从数据库或缓存中拉取用户登录信息，包含 websocket 客户端 uuid
2. 构建包含 uuid 的消息，发送到消息队列

```go
package msg

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

type SendHandler struct {
        container component.IContainer
}

func NewSendHandler(container component.IContainer) (u *SendHandler) {
        return &SendHandler{
                container: container,
        }
}
//Handle api接口发送消息，将消息存入消息队列
func (u *SendHandler) Handle(ctx *context.Context) (r interface{}) {
    //从缓存或数据库中拉取用户信息，根据保存的uuid发送消息
    queue:=ctx.GetContain().GetRegularQueue()
    if err:=queue.Push("mall:wsserver:msg",`{"uuid":"890997777","msg":"充值成功"}`);err!=
        return err
    }
    return "success"
}
```

services/msg_push.go

1. 从输入参数中获取 uuid,msg
2. 调用 `context.WSExchange.Notify` 向 websocket 客户端发送消息。也可以调用 `context.WSExchange.Broadcast` 向所有 `websocket` 客户端广播消息

```go
package msg

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

type PushHandler struct {
        container component.IContainer
}

func NewPushHandler(container component.IContainer) (u *PushHandler) {
        return &PushHandler{
                container: container,
        }
}
//Handle api接口发送消息，将消息存入消息队列
func (u *PushHandler) Handle(ctx *context.Context) (r interface{}) {
    uuid:=ctx.Request.String("uuid")
    content:=ctx.Request.String("msg")
    if err := context.WSExchange.Notify(uuid,content); err != nil {
                return err
        }
    return "success"
}
```