构建远程调用服务(RPC)

本示例介绍如何构建RPC服务器，以及RPC调用方法。

RPC服务器的构建方法与API服务器的构建相同，只需修改服务器类型为 `rpc` 即可，其它完全相同。

1. 创建服务器

main.go

```
package main

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

type rpcserver struct {
        *hydra.MicroApp
}

func main() {
        app := &rpcserver{
                hydra.NewApp(
                        hydra.WithPlatName("mall"),
                        hydra.WithSystemName("rpcserver"),
                        hydra.WithServerTypes("rpc")),
        }
        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 (rpc *rpcserver) config() {
        rpc.IsDebug = true
        rpc.Conf.RPC.SetMainConf(`{"address":":9090","trace":true}`)
        rpc.Conf.Plat.SetVarConf("db", "db", `{
                        "provider":"mysql",
                        "connString":"mrss:123456@tcp(192.168.0.36)/mrss?charset=utf8",
                        "maxOpen":20,
                        "maxIdle":10,
                        "lifeTime":600
        }`)

}
```

2. 初始化检查与服务注册

```
package main

import (
        "github.com/micro-plat/hydra/component"
        "github.com/micro-plat/hydra/quickstart/demo/rpcserver01/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 *rpcserver) 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.Micro("/order", order.NewOrderHandler)
}
```

> `rpc.Micro` 注册为 `api`，`rpc` 类型，也可使用 `rpc.RPC("/order", order.NewOrderHandler)` 只注册为 `rpc` 服务

3. 请求预处理，验证签名

```
package main

import (
        "fmt"

        "github.com/micro-plat/hydra/context"
        "github.com/micro-plat/hydra/quickstart/demo/rpcserver11/modules/merchant"
)

func (rpc *rpcserver) handling() {
        rpc.MicroApp.Handling(func(ctx *context.Context) (rt interface{}) {
                if err := ctx.Request.Check("merchant_id"); err != nil {
                        return err
                }
                key, err := merchant.GetKey(ctx,ctx.Request.GetInt(merchant_id))
                if err != nil {
                        return err
                }
                if !ctx.Request.CheckSign(key) {
                        return fmt.Errorf(908, "商户签名错误")
                }
                return nil
        })
}
```

3. 构建服务

servers/order.go

```
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,
        }
}
//QueryHandle 充值结果查询
func (u *OrderHandler) QueryHandle(ctx *context.Context) (r interface{}) {
        ctx.Log.Info("--------------充值结果查询---------------")
        return "SUCCESS"
}
```

说明:

- 设置Response输出类型为 `json`，`xml`，`plain` 等，返回给调用方的数据不会转换为相应的格式，只会将此信息保存到返回参数的 `HeaderMap` 中。可直接获取 `HeaderMap` 的值设置到 `api` 服务的 `Response` 中输出到 `http` 响应流

4. 安装并启动RPC服务器

安装配置信息:

```
~/work/bin$ rpcserver01 registry -r zk://192.168.0.109 -c yl
        -> 创建注册中心配置数据?如存在则不安装(1),如果存在则覆盖(2),删除所有配置并重建(3),退出(n|r
                创建配置: /mall_debug/rpcserver/rpc/yl/conf
                创建配置: /mall_debug/var/db/db
```

运行服务:

```
~/work/bin$ rpcserver01 run -r zk://192.168.0.109 -c yl
[2019/07/02 11:44:12.274275][i][49f56b398]Connected to 192.168.0.109:2181
[2019/07/02 11:44:12.289725][i][49f56b398]Authenticated: id=246395503264334090, timeout=
[2019/07/02 11:44:12.289766][i][49f56b398]Re-submitting `0` credentials after reconnect
[2019/07/02 11:44:12.321996][i][49f56b398]初始化 /mall_debug/rpcserver/rpc/yl
[2019/07/02 11:44:12.336878][i][4f24e906a]开始启动[RPC]服务...
[2019/07/02 11:44:12.337520][d][4f24e906a][未启用 jwt设置]
[2019/07/02 11:44:12.337530][d][4f24e906a][未启用 header设置]
[2019/07/02 11:44:12.337535][d][4f24e906a][未启用 metric设置]
[2019/07/02 11:44:12.337538][d][4f24e906a][未启用 host设置]
[2019/07/02 11:44:12.962662][i][4f24e906a]服务启动成功(RPC,tcp://192.168.4.121:9090,1)
```

5. 查看注册中心

使用 `ZooInspector` 连接到zookeeper

节点 `/[platName]/services/rpc/[systeName]/[serviceName]/providers/` 显示了当前服务的所有提供者

当前服务注册路径为:

mall_debug

-------services

-----------rpc

--------------order

----------------query

--------------------providers

----------------------192.168.4.121

6. 远程调用

1. 内置实例

使用 `ctx` 中提供的 `RPC` 对象调用

```go
package order

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

type QueryHandler struct {
   container component.IContainer
}

func NewQueryHandler(container component.IContainer) (u *QueryHandler) {
   return &QueryHandler{container: container}
}
func (u *QueryHandler) Handle(ctx *context.Context) (r interface{}) {
   //构建输入参数
   header:=map[string]string{}
   input:=map[string]interface{}{
       "order_no":ctx.GetString("order_no"),
   }

   //发起远程调用
   status, result, params, err := ctx.RPC.Request("/order/query@apiserver.mall", "GET",
   if err!=nil{
       return err
   }

   //处理响应
   ctx.Response.SetHeaders(params) //将RPC服务设置的头，设置到当前的response中
   ctx.Response.MustContent(status, result)//根据状态和内容强制设置输出
   return

}
```

说明:

- RPC调用的服务名全称为: `[serverName]@[systemName].[platName]` ,调用系统与RPC系统的平台名相同时，可简写为： `[serverName]@[systemName]` ,调用系统与RPC系统的平台名与系统名相同时可直接写作: `[serverName]`
- 使用 `ctx.RPC` 请求的优势是会自动将当前请求的 `session id` 传给 `rpc` 服务，即：rpc服务打印的 `session id` 与调用方的 `session id` 相同，方便查找日志

2. 独立实例

一般服务使用方与提供方未共用同一个注册中心时，需单独创建 `rpc` 实例

```
//初始化RPC调用服务
invoker:=rpc.NewInvoker(platName,sysName,registryAddr,opts)

//发起远程调用
status, result, params, err := ctx.RPC.Request("/order/query", "GET", nil, nil, true
```

> 通过 opts 可指定 日志组件 ， 负载均衡器 ， ssl证书 等

> 负载均衡器支持轮询与本地优先