



# 21天微服务实战营-Day11

华为云DevCloud & ServiceStage服务联合出品



# Day11 CSE实战之使用CSEGoSDK开发微服务

## 大纲

- 开发CSEGoSDK微服务
- 开发CSEGoSDK服务调用者

# 开发CSEGoSDK微服务

准备工作：

- ❑ 创建一个新的GO项目
- ❑ 从华为云上下载CSEGoSDK压缩包，并解压到项目的vendor下



# 开发CSEGoSDK微服务

## ➤ 创建Rest接口

我们创建一个struct，名为XXX,然后创建XXX的Hello方法，该法必须参数为\*restful.Context

```
type Service struct {}  
  
func (*Service) Hello(ctx *restful.Context) {  
    name := ctx.ReadPathParameter( name: "name")  
    ctx.WriteJSON(fmt.Sprintf( format: "hello., %s", name), contentType: "application/json")  
}
```

## ➤ 编写url patterns.

```
// URLPatterns  
func (*Service) URLPatterns() []restful.Route {  
    return []restful.Route {  
        {Method: http.MethodGet, Path: "/provider/v0/hello/ {name}", ResourceFuncName: "Hello"},  
    }  
}
```

# 开发CSEGoSDK微服务

创建一个main.go：

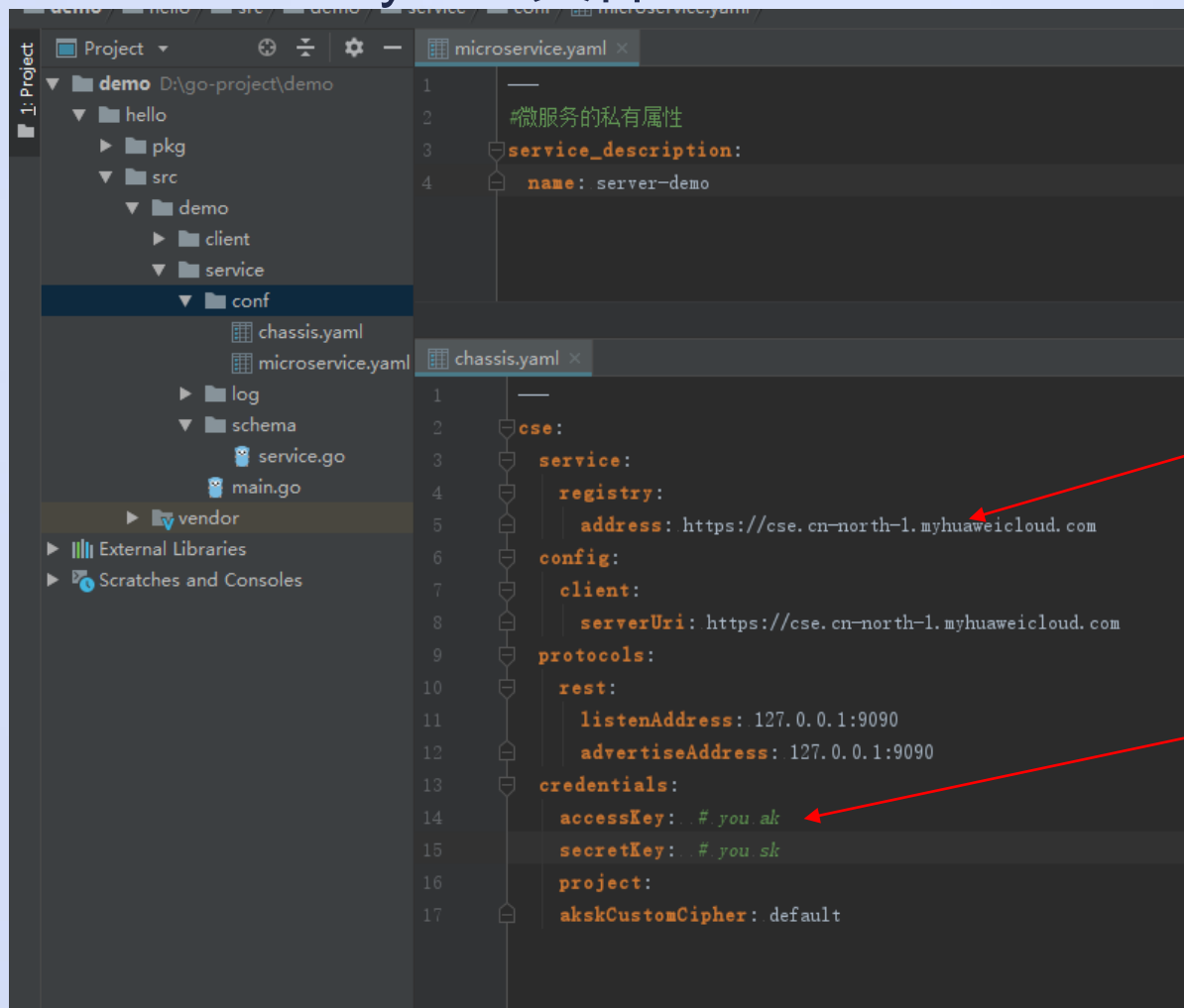
在main.go中我们需要将我们刚刚创建的Service进行注册

```
func main() {  
    chassis.RegisterSchema( serverName: "rest", &schema.Server{}, server.WithSchemaID( schemaID: "server_go"))  
    err := chassis.Init()  
    if err != nil {  
        openlogging.GetLogger().Error(err.Error())  
        return  
    }  
    chassis.Run()  
}
```

在main.go中使用chassis.RegisterSchema方法进行注册，使用chassis.Init()进行初始化，使用chassis.Run()进行启动服务，CSEGoSDK支持rest协议和grpc协议，开发者可以自由选择

# 开发CSEGoSDK微服务

在main.go同级目录下创建conf目录，并在conf目录下创建下 chassis.yaml 和 microservice.yaml 文件



servercenter地址，该地址必须填写

配置ak/sk，ak/sk为必填项，如不配置将无法连接到华为云

# 开发CSEGoSDK微服务

运行main.go,可以在ServiceStage的微服务控制台上可以开到server-demo 服务



微服务控制台  
Cloud Service Engine

仪表盘

服务目录

服务治理

全局配置

事务看板

服务目录

您可以从应用、微服务和实例的维度查看微服务详细信息。如果微服务较多，您可以通过搜索功能查找目标微服务。[如何维护微服务？](#)

应用列表 微服务列表 实例列表

创建微服务

删除

全部应用(1)

微服务名称

<input type="checkbox"/> 微服务名称	所属应用	版本数	实例数	创建时间	操作
<input type="checkbox"/> server-demo	default	1	1	2019/02/25 09:08:16 GMT+08:00	<a href="#">删除</a>

注意：如果直接使用IDE启动，需要配置环境变量  
CHASSIS\_HOME=/path/to/demo/service/

# 你已成功开发出CSEGoSDK服务

至此，你已经使用CSEGoSDK成功开发出了微服务。调用  
<http://127.0.0.1:9090/provider/v0/hello/Bod>，得到server以下的回复

The screenshot displays a REST client interface with the following components:

- Request Bar:** Method: GET, URL: `http://127.0.0.1:9090/provider/v0/hello/Bod`
- Request Tabs:** Params, Authorization, Headers (1), Body, Pre-request Script, Tests. The 'Headers' tab is active.
- Request Headers Table:**

KEY	VALUE
Key	Value
- Response Tabs:** Body, Cookies (1), Headers (3), Test Results. The 'Body' tab is active.
- Response Format:** Pretty, Raw, Preview. Format: JSON.
- Response Content:**

```
1  "hello , Bod"
```



# 开发微服务调用者

开发一个consumer服务来调用provider服务，定义一个REST接口类接收外部请求并调用provider服务：


```
14
15     type Client struct {
16     }
17
18     func (*Client) Hello(ctx *restful.Context) {
19         //可以使用 cse:// 和 http://作为前缀
20         //req, err := rest.NewRequest(http.MethodGet, fmt.Sprintf("http://server-demo/provider/v0/hello/%s", ctx.ReadPathParameter("name")), nil)
21         req, err := rest.NewRequest(http.MethodGet, fmt.Sprintf("cse://server-demo/provider/v0/hello/%s", ctx.ReadPathParameter("name")), body: nil)
22         if err != nil {
23             ctx.WriteError(http.StatusInternalServerError, err)
24             return
25         }
26         resp, err := core.NewRestInvoker().ContextDo(context.TODO(), req)
27         if err != nil {
28             ctx.WriteError(http.StatusInternalServerError, err)
29             return
30         }
31         ctx.Write(httputil.ReadBody(resp))
32     }
```

通过core.NewRestInvoker()，创建新的restInvoker(CSEGoSD所有请求均抽象为 invocation),通过restInvoker下的ContextDo方法来对服务发起请求

# 开发微服务调用者

创建main.go:

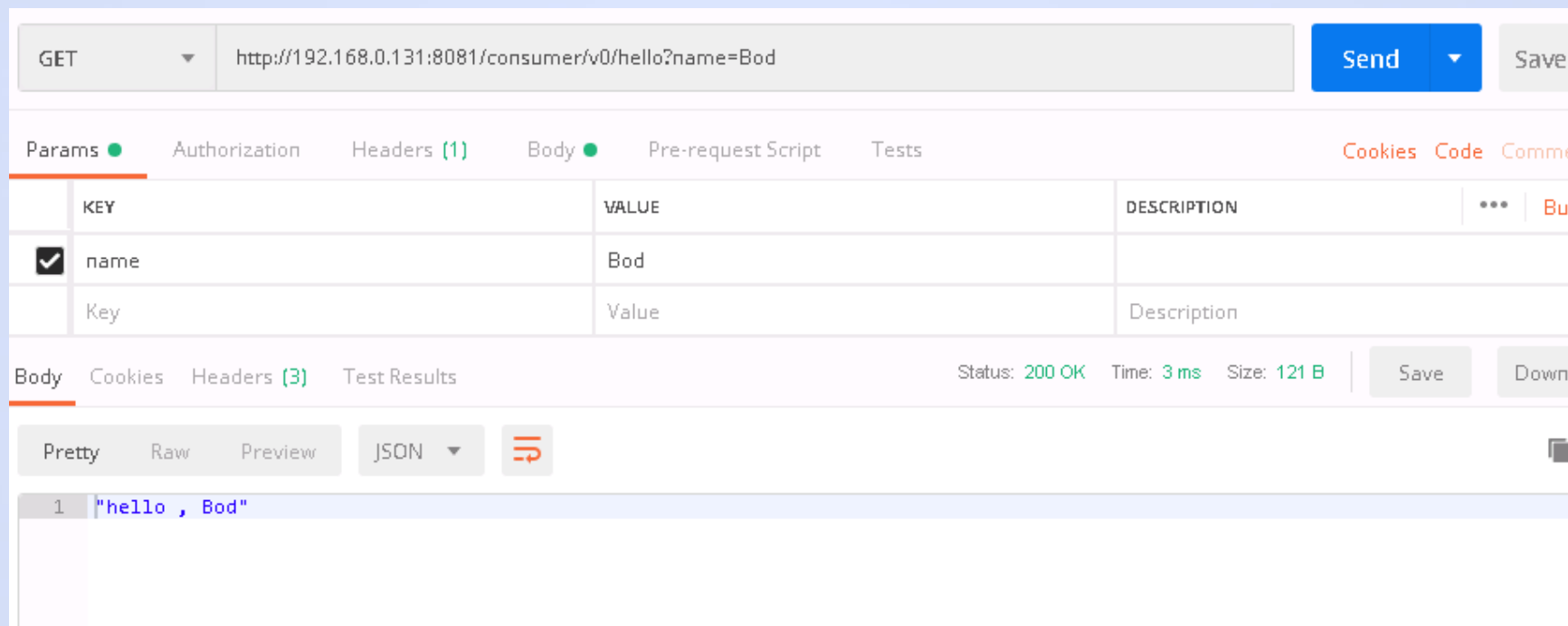
main.go和provider端的main.go基本一致，修改以下两个箭头所指地方即可



```
2 func main() {  
3     chassis.RegisterSchema( serverName: "rest", &schema.Client {}, server.WithSchemaID( schemaID: "client-demo"))  
4     if err := chassis.Init(); err != nil {  
5         openlogging.GetLogger().Error("Init failed." + err.Error())  
6         return  
7     }  
8     chassis.Run()  
9 }
```

# 开发微服务调用者

配置文件chassis.yaml和microservice.yaml文件基本一直，我们只要修改chassis.yaml中的监听地址，监听地址我们改为：127.0.0.1:8080,以及microservice.yaml中的服务名,服务名修改为client-demo。启动consumer端服务，同样在serverstage看到该服务。调用consumer服务



# Thank You

