接口文档

一、API Layer模块

本模块基于Hertz的HTTP Server；接受HTTP POST请求（JSON格式）。

1.1 供接口

1.1.1 通过路由提供泛化客户端

- **方法名：**

  `GateWayMethod`

- **代码实现：**

```go
// GateWayMethod .
// @router /agw [POST]
func GateWayMethod(ctx context.Context, c *app.RequestContext) {
	var err error
	var req layer.LayReq
	err = c.BindAndValidate(&req)
	if err != nil {
		c.String(consts.StatusBadRequest, err.Error())
		return
	}

	fmt.Println(req.ServiceName)
	fmt.Println(req.ServiceMethod)

	//这里调用routing层的RoutingDistribute方法
	cli := routingHandler.RoutingDistribute(req.ServiceName)

	reqBytes, err := c.Body()
	if err != nil {
		c.String(consts.StatusBadRequest, err.Error())
		return
	}

	resp, err := cli.GenericCall(ctx, req.ServiceMethod, string(reqBytes))
	if err != nil {
		c.String(consts.StatusBadRequest, err.Error())
		return
```

```
28          }
29
30          c.JSON(consts.StatusOK, resp)
31  }
```

- **参数：**

字段	说明	类型	备注	是否必填
ctx	内容	context.Context		是

- **返回值：**

字段	说明	类型	备注	是否必填
c	请求内容指针	*app.RequestContext		是

1.2 需接口

模块	接口方法
Routing模块	`func (ptr *defaultRouteHandler) RoutingDistribute(serviceName string) (client genericclient.Client)`

二、Routing模块

本模块决定请求应当发往哪个RPC服务。

2.1 供接口

2.1.1 通过路由提供泛化客户端

- **方法名：**

  `RoutingDistribute`

- **代码实现：**

```
1  func (ptr *defaultRouteHandler) RoutingDistribute(serviceName string) (client ge
2          client = ptr.cliProvider.NewGenericClient(serviceName)
3          return
```

```
4 }
```

- **参数：**

字段	说明	类型	备注	是否必填
serviceName	服务名称	string		是

- **返回值：**

字段	说明	类型	备注	是否必填
client	客户端	genericclient.Client		是

2.2 需接口

模块	接口方法
Kitex Client Provider模块	`func (ptr *DefaultKitexClientProvider) NewGenericClient(serviceName string, idlContent string) (client genericclient.Client)`

三、Kitex Client Provider模块

本模块根据路由结果获取对应泛化调用客户端；缓存提高性能；从注册中心获取目标服务实例。

3.1 供接口

3.1.1 返回泛化客户端

- **方法名：**

  `NewGenericClient`

- **代码实现：**

```
1 func (ptr *defaultKitexClientProvider) NewGenericClient(serviceName string) (cli
2         // 如果缓存中有,直接返回
3         if cacheData := ptr.cache.get(serviceName); cacheData.client != nil {
4                 return *cacheData.client
5         }
6
```

```
        // 如果缓存中没有,从idlProvider中获取IDL内容,并且生成client
        idlContent := ptr.idlProvider.FindIDLByServiceName(serviceName)
        p, err := generic.NewThriftContentProvider(idlContent, map[string]string
        if err != nil {
                panic(err)
        }
        g, err := generic.JSONThriftGeneric(p)
        if err != nil {
                panic(err)
        }
        client, err = genericclient.NewClient(serviceName, g, cli.WithResolver(p
        if err != nil {
                panic(err)
        }

        ptr.cache.set(serviceName, &cacheData{client: &client, provider: p})
        return
}
```

- **参数**:

字段	说明	类型	备注	是否必填
serviceName	服务名称	string		是

- **返回值**:

字段	说明	类型	备注	是否必填
client	客户端	genericclient.Client		是

3.2 需接口

模块	接口方法
IDL Provider模块	`func (ptr *defaultIdlProvider) FindIDLByServiceName(serviceName string) (idlContent string)`
IDL Management Platform模块	`func NewManageServiceClient(hostUrl string, ops ...Option) (Client, error)`

四、IDL Provider模块

本模块提供构造泛化调用客户端需要的 IDL 文件；缓存提高性能。

4.1 供接口

4.1.1 通过服务名返回对应IDL

- **方法名：**

  `FindIDLByServiceName`

- **代码实现：**

```
func (ptr *defaultIdlProvider) FindIDLByServiceName(serviceName string) (idlCont
        req := idlmanage.NameBasedReq{Name: serviceName}
        _, resp, _ := ptr.idlManagement.DownloadByName(context.Background(), &re
        if resp.StatusCode() != 200 { // 如果IDL管理服务返回状态码不是200,则返回空字
                return
        }
        idlContent = string(resp.Body())
        return
}
```

- **参数：**

字段	说明	类型	备注	是否必填
serviceName	服务名称	string		是

- **返回值：**

字段	说明	类型	备注	是否必填
idlContent	IDL内容	string		是

4.2 需接口

模块	接口方法

IDL Management Platform模块	`func (s *ManageServiceClient) DownloadByName(context context.Context, req *IDLManage.NameBasedReq, reqOpt ...config.RequestOption)`
IDL Management Platform模块	`func NewManageServiceClient(hostUrl string, ops ...Option) (Client, error)`

五、IDL Management Platform模块

本模块实现IDL管理平台，含增/删/改/查界面；提供 API 给 IDL Provider 调用。

5.1 供接口

5.1.1 返回新建管理服务的客户端

- **方法名：**

  `NewManageServiceClient`

- **代码实现：**

```go
func NewManageServiceClient(hostUrl string, ops ...Option) (Client, error) {
        opts := getOptions(append(ops, withHostUrl(hostUrl))...)
        cli, err := newClient(opts)
        if err != nil {
                return nil, err
        }
        return &ManageServiceClient{
                client: cli,
        }, nil
}
```

- **参数：**

字段	说明	类型	备注	是否必填
hostUrl	主机URL	string		是

- **返回值：**

字段	说明	类型	备注	是否必填
ops	选项列表	...Option		是

5.1.1 通过名字下载对应服务

- **方法名：**

  `DownloadByName`

- **代码实现：**

```
func (s *ManageServiceClient) DownloadByName(context context.Context, req *IDLMa
        httpResp := &IDLManage.DownloadResp{}
        ret, err := s.client.r().
                setContext(context).
                setQueryParams(map[string]interface{}{
                        "name": req.GetName(),
                }).
                setPathParams(map[string]string{}).
                setHeaders(map[string]string{}).
                setFormParams(map[string]string{}).
                setFormFileParams(map[string]string{}).
                setBodyParam(req).
                setRequestOption(reqOpt...).
                setResult(httpResp).
                execute("GET", "/api/download")
        if err != nil {
                return nil, nil, err
        }

        resp = httpResp
        rawResponse = ret.rawResponse
        return resp, rawResponse, nil
}
```

- **参数：**

字段	说明	类型	备注	是否必填
context	内容	context.Context		是
req	请求信息	*IDLManage.NameBasedReq		是
reqOpt	请求选项	...config.RequestOption		是

- **返回值：**

字段	说明	类型	备注	是否必填
resp	回复信息	*IDLManage.DownloadResp		是
rawResponse	初始回复	*protocol.Response		是
err	错误信息	error		是

5.2 需接口

模块	接口方法