# 使用说明文档

## 一、项目介绍

### 1.1 简介

项目基于cloudwego技术实现api网关的搭建。在项目运行中接收 HTTP 请求，将其转换成Thrift RPC请求，发给目标RPC服务，并将结果转换成JSON返回给客户端。

### 1.2 项目设计

项目分为API层、Routing层、Kitex Client Provider、IDL Provider以及IDL Management Platform。其中使用到hertz与kitex框架以及泛化调用、前端实现、后端实现、数据库等技术。

API Layer：基于Hertz的HTTP Server；接受HTTP POST请求（JSON格式）

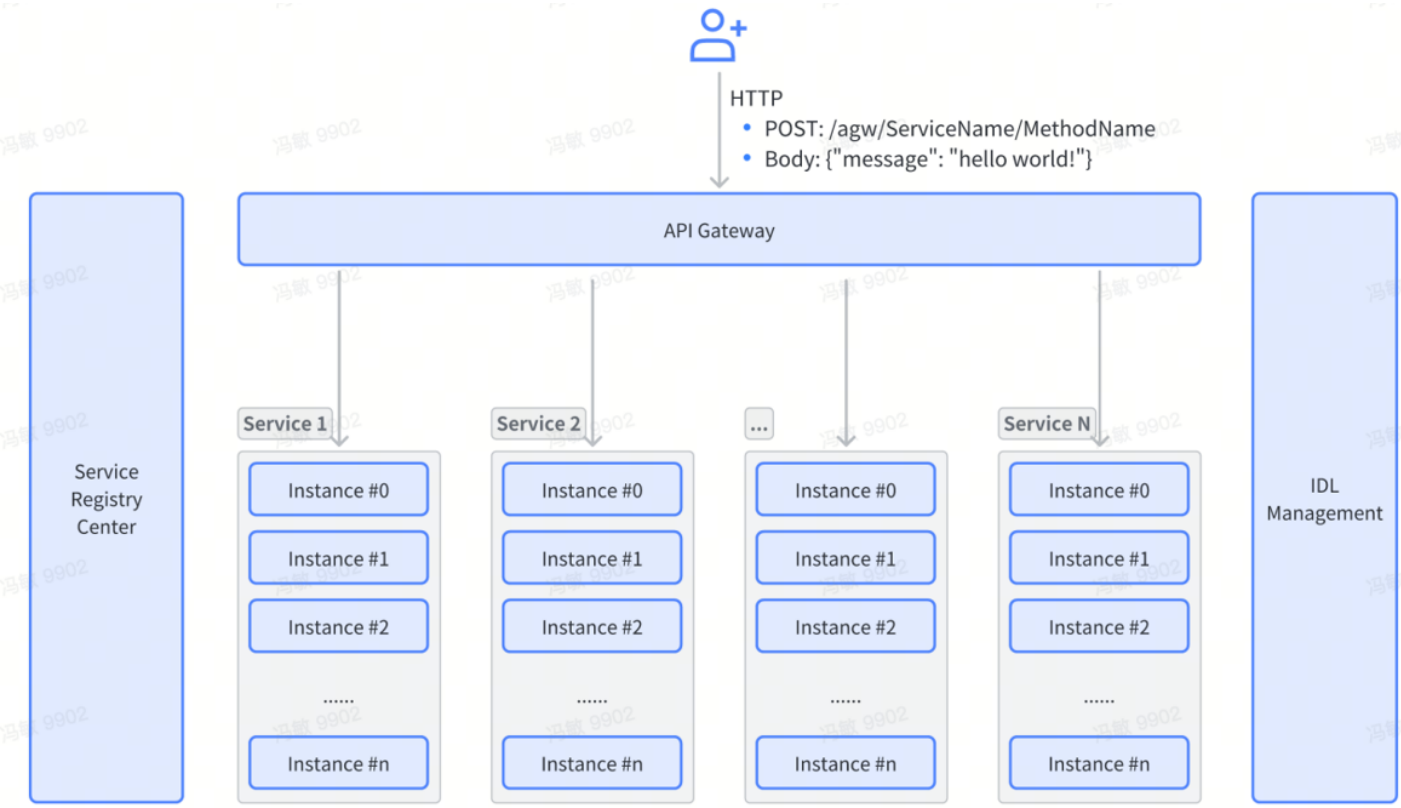
Routing：决定请求应当发往哪个RPC服务

Kitex Client Provider：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

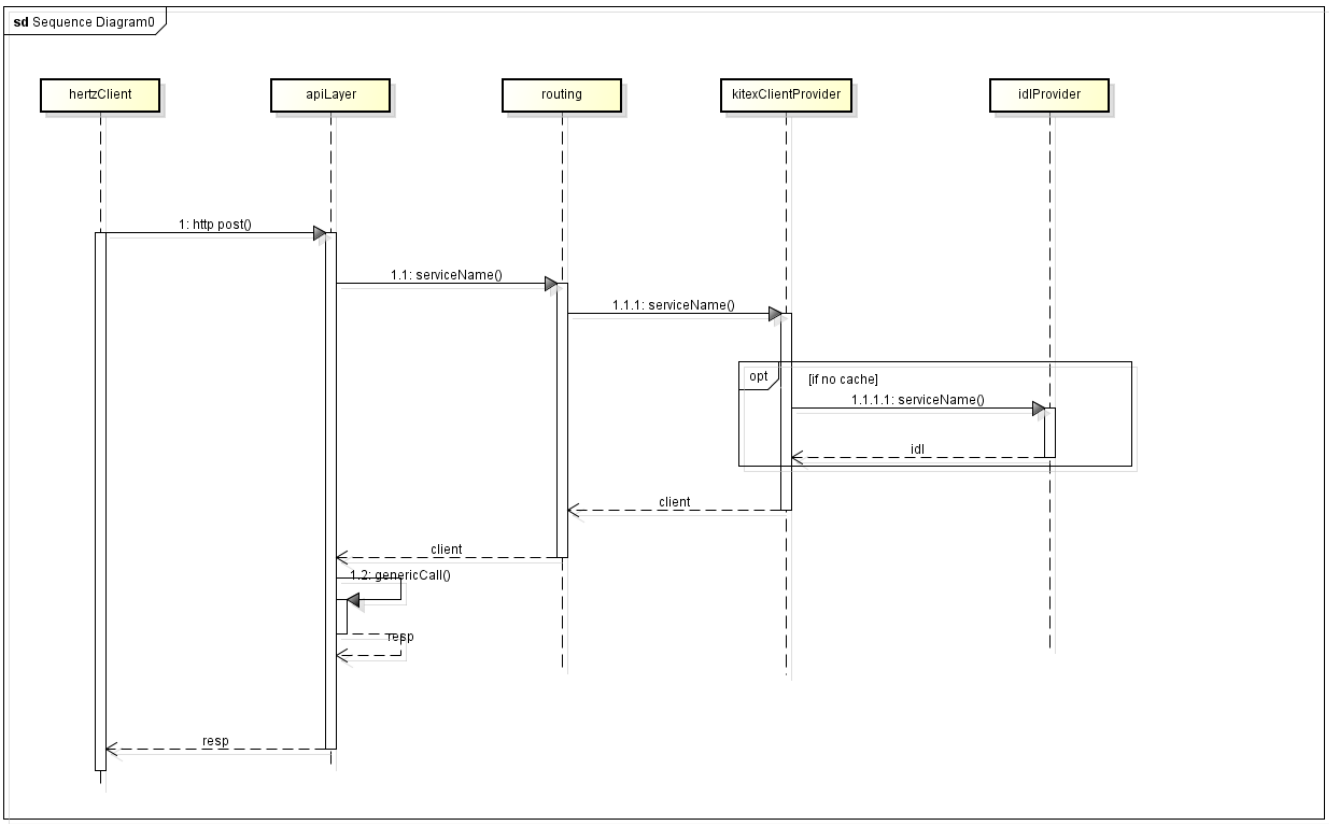
IDL Provider：提供构造泛化调用客户端需要的 IDL 文件；缓存提高性能

IDL Management Platform：实现IDL管理平台，含增/删/改/查界面；提供 API 给 IDL Provider 调用

- **项目架构图示：**




- **流程图：**




# 二、功能介绍

## 2.1 接受与响应HTTP POST请求

项目可以正确接收和响应 请求体为JSON格式的HTTP POST请求。

### 2.1.1 功能实现代码

```
// 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
        }

        c.JSON(consts.StatusOK, resp)
}
```

## 2.2 路由确认目标服务和方法

项目可以根据请求路由确认目标服务和方法。

### 2.2.1 功能实现代码

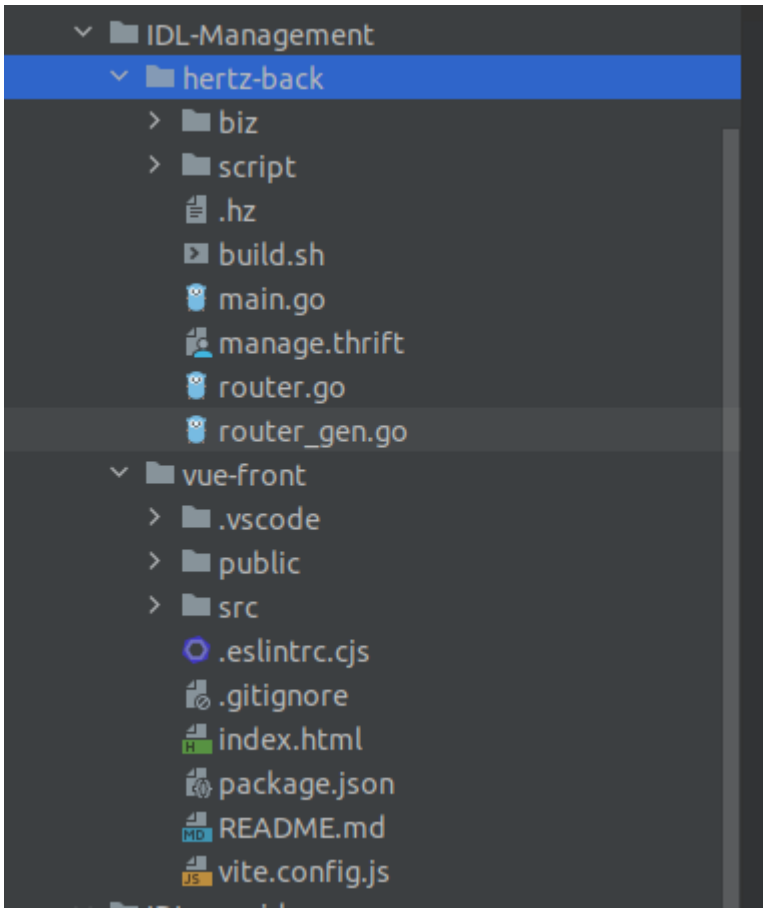
```
func (ptr *defaultRouteHandler) RoutingDistribute(serviceName string) (client ge
        client = ptr.cliProvider.NewGenericClient(serviceName)
        return
}
```

## 2.3 实现IDL管理模块

项目API网关内的 IDL管理模块，可为构造 Kitex Client提供 IDL。

### 2.3.1 功能实现代码

- IDL—Management前后端代码结构：




- 部分实现代码：

```
// SelectAll .
// @router /api/getAll [GET]
func SelectAll(ctx context.Context, c *app.RequestContext) {
        var err error
        var req IDLManage.EmptyReq
        err = c.BindAndValidate(&req)
        if err != nil {
                c.String(consts.StatusBadRequest, err.Error())
                return
        }

        var recoreds []IDLRecorder

        // Get all records
```

```
15          result := db.Find(&recoreds)
16          // SELECT * FROM users;
17          if result.Error != nil {
18                  c.String(consts.StatusBadRequest, err.Error())
19                  return
20          }
21
22          resp := new(IDLManage.QueryResp)
23
24          resp.Ls = make([]*IDLManage.IDLEntity, len(recoreds))
25          for i, v := range recoreds {
26                  resp.Ls[i] = RecorderToEntity(&v)
27          }
28
29          c.JSON(consts.StatusOK, resp)
30  }
31
32  // SelectByName .
33  // @router /api/search [GET]
34  func SelectByName(ctx context.Context, c *app.RequestContext) {
35          var err error
36          var req IDLManage.NameBasedReq
37          err = c.BindAndValidate(&req)
38          if err != nil {
39                  c.String(consts.StatusBadRequest, err.Error())
40                  return
41          }
42          var recorder IDLRecorder
43
44          result := db.Where("service_name = ?", req.Name).First(&recorder)
45          if result.Error != nil {
46                  c.String(consts.StatusBadRequest, err.Error())
47                  return
48          }
49
50          resp := new(IDLManage.QueryResp)
51
52          resp.Ls = append(resp.Ls, RecorderToEntity(&recorder))
53
54          c.JSON(consts.StatusOK, resp)
55  }
56  ...
```

## 2.4 完成Kitex泛化调用

项可以通过目构造Kitex 泛化调用客户端、发起请求并处理影响结果。

### 2.4.1 功能实现代码

- 构造Kitex 泛化调用客户端:

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

        // 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
}
```

- 进行泛化调用:

```
        //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
```

```
14          }
```