# SpringCloud-Eureka

服务发现是基于微服务架构的关键原则之一。尝试配置每个客户端或某种形式的约定可能非常困难，可以非常脆弱。Netflix服务发现服务器和客户端是Eureka。可以将服务器配置和部署为高可用性，每个服务器将注册服务的状态复制到其他服务器

类似于Dubbo的服务之间的连接,保存不同服务直接的信息,通过fegin来完成服务直接的调用

在Idea中可以直接创建Eureka服务端



**server**:  
 **port**: 8080  
**eureka**:  
 **client**:  
 **service-url**:  
 *#Eureka服务端的服务地址* **defaultZone**: http://127.0.0.1:8080/eureka/  
 **register-with-eureka**: **false** *#不在注册中心注册自己为服务* **fetch-registry**: **false**

在Idea中可以直接创建Eureka客户端



**server**:  
 **port**: 8081  
  
*# 本服务的名字,便于发现***spring**:  
 **application**:  
 **name**: eureka-provider  
  
*#注册中心地址,注册到注册中心***eureka**:  
 **client**:  
 **service-url**:  
 **defaultZone**: http://127.0.0.1:8080/eureka/

# SpringCloud-Fegin

[声明式服务调用](https://www.cnblogs.com/huangjuncong/p/9053576.html)

<**dependency**>  
 <**groupId**>org.springframework.cloud</**groupId**>  
 <**artifactId**>spring-cloud-starter-openfeign</**artifactId**>  
</**dependency**>

Service层

(**不再使用@Service,因为是Fegin通过RestFul风格http调用**)

@RestController  
**public class** ProviderServiceImpl **implements** ProviderService {  
  
 @Value(**"${server.port}"**)  
 **private** String **serverPort**;  
  
 @Override  
 @RequestMapping(**"/getMsg"**)  
 **public** String getMsg(@RequestParam String name) {  
 **return "获取到了提供者的信**

**息,Port:"**+**serverPort**+**",Name:"**+name;  
 }  
}

Controller层:

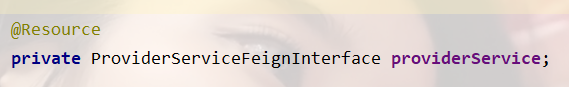


需要创建一个调用的接口

(方法和service的一样,注意:如果有参数传递必须使用注解声明)

@FeignClient(**"eureka-provider"**) //service服务的名称  
**public interface** ProviderServiceFeignInterface {  
  
 @RequestMapping(**"/getMsg"**)  
 String getMsg(@RequestParam String name);  
  
}

Controller中调用接口时:



配置:

# SpringCloud-Security

安全认证

<**dependency**>  
 <**groupId**>org.springframework.boot</**groupId**>  
 <**artifactId**>spring-boot-starter-security</**artifactId**>  
</**dependency**>

为注册中心设置账号和密码

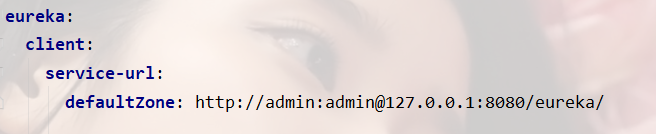
服务端:

*#设置eureka登录权限***spring**:  
 **security**:  
 **user**:  
 **name**: admin  
 **password**: admin

还需要加入以下配置, 关闭csrf检验

@EnableWebSecurity  
**public class** WebSecurityConfig **extends** WebSecurityConfigurerAdapter {  
  
 @Override  
 **protected void** configure(HttpSecurity http) **throws** Exception {  
 http.csrf().disable().authorizeRequests().anyRequest().authenticated().and().httpBasic();  
 }  
}

客户端:



# SpringCloud-Hystrix

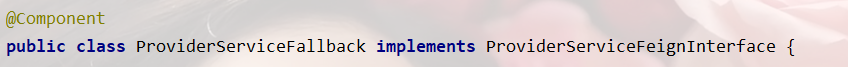
熔断处理

不需要导包,Feign中已经包含的Hystrix依赖

配置

**feign**:  
 **hystrix**:  
 **enabled**: **true** *#开启hystrix*

创建一个Feign客户端接口实现类



Feign客户端接口修改注解



# SpringCloud-Ribbon

负载均衡策略

不需要导包,Eureka服务发现已经导入了依赖

Ribbon默认开启

当有服务集群的时候,会对集群进行轮询访问

# SpringCloud-Zuul

网关

主要用作:反向代理、Filter扩展、动态加载、动态路由、压力测试、弹性扩展、审查监控、安全检查等

(此服务为独立服务,需要注册到Eureka中,端口号为80)

<**dependency**>  
 <**groupId**>org.springframework.cloud</**groupId**>  
 <**artifactId**>spring-cloud-starter-netflix-zuul</**artifactId**>  
</**dependency**>

**server**:  
 **port**: 80 *#80端口为总端口,防止暴露接口端口号***spring**:  
 **application**:  
 **name**: server-zuul  
**eureka**:  
 **client**:  
 **service-url**:  
 **defaultZone**: http://admin:admin@127.0.0.1:8080/eureka/  
**zuul**:  
 **routes**:  
 *#前半部分为 服务名称 ,后半部分为 跳转路径* **eureka-customer**: /customer/\*\*

(注册到Eureka中,并且开启Zuul网关代理)



Zuul过滤器的应用(eg:判断是否登录)

@Component  
**public class** TokenFilter **extends** ZuulFilter {  
 @Override  
 **public** String filterType() {  
 *//前过滤 FilterConstants.PRE\_TYPE 后过滤 FilterConstants.POST\_TYPE* **return** FilterConstants.***PRE\_TYPE***;  
 }  
  
 @Override  
 *//过滤器优先级* **public int** filterOrder() {  
 **return** 0;  
 }  
  
 @Override  
 *//是否应该起作用* **public boolean** shouldFilter() {  
 **return true**;  
 }  
  
 @Override  
 **public** Object run() **throws** ZuulException {  
 RequestContext context = RequestContext.*getCurrentContext*();  
 HttpServletRequest request = context.getRequest();  
 String token = request.getParameter(**"token"**);  
 **if** (StringUtils.*isEmpty*(token)||token.equals(**""**)){  
 context.setSendZuulResponse(**false**); *//是否放行* context.setResponseStatusCode(401); *//返回状态码* context.setResponseBody(**"token is null!"**);*//返回信息* }  
 **return null**;  
 }  
}

# SpringCloud-Config

分布式配置中心

有一个独立的Config服务

<**dependency**>  
 <**groupId**>org.springframework.cloud</**groupId**>  
 <**artifactId**>spring-cloud-config-server</**artifactId**>  
</**dependency**>



**server**:  
 **port**: 8081  
**spring**:  
 **application**:  
 **name**: server-config  
 **cloud**:  
 **config**:  
 *#分支* **label**: master  
 **server**:  
 **git**:  
 *#码云Git地址* **uri**: https://gitee.com/jishixin12/config.git  
 *#分支文件夹* **search-paths**: config

在需要使用Config配置中心的微服务中添加依赖

*<!--Config配置中心客户端-->*<**dependency**>  
 <**groupId**>org.springframework.cloud</**groupId**>  
 <**artifactId**>spring-cloud-config-client</**artifactId**>  
</**dependency**>  
*<!--actuator监控,配合可支持手动刷新-->*<**dependency**>  
 <**groupId**>org.springframework.boot</**groupId**>  
 <**artifactId**>spring-boot-starter-actuator</**artifactId**>  
</**dependency**>

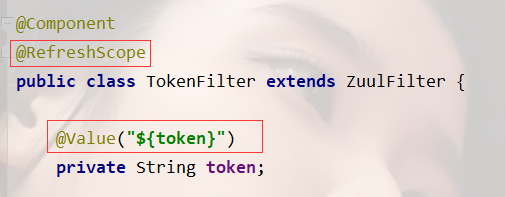
创建bootstrap.yml文件

bootstrap.yml(不改变的配置)用来程序引导时执行，应用于更加早期配置信息读取，如可以使用来配置application.yml中使用到参数等

application.yml(会改变的配置)应用程序特有配置信息，可以用来配置后续各个模块中需使用的公共参数等。

bootstrap.yml 先于 application.yml 加载

**server**:  
 **port**: 80  
**spring**:  
 **application**:  
 **name**: server-zuul  
 **cloud**:  
 **config**:  
 *#配置中心地址* **uri**: http://127.0.0.1:8081  
 *#开发阶段* **profile**: dev  
 **label**: master  
*#开启监控断点(开启后才能进行手动刷新)***management**:  
 **endpoints**:  
 **web**:  
 **exposure**:  
 **include**: **"\*"  
eureka**:  
 **client**:  
 **service-url**:  
 **defaultZone**: http://admin:admin@127.0.0.1:8080/eureka/



(Port: config-client 的端口号,必须以Post方式调用)

<http://127.0.0.1:Port/actuator/refresh>

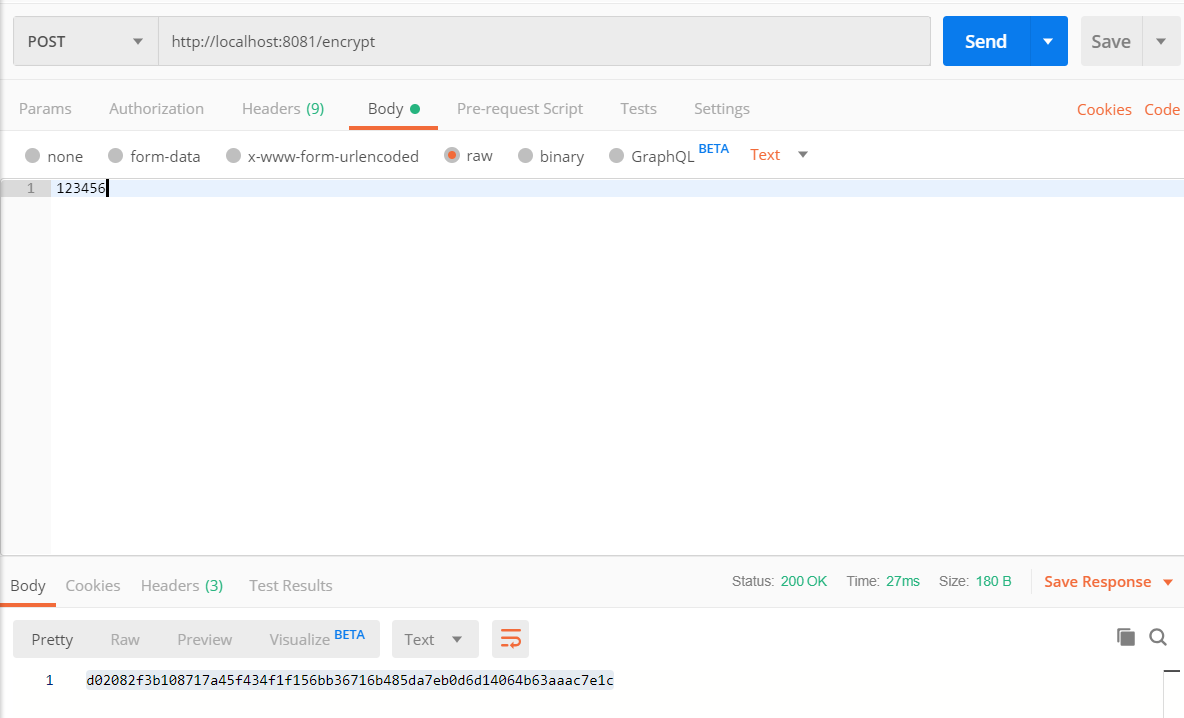
数据传输中的加密与解密

在JDK中添加JCE加密jar

<http://localhost:8000/encrypt/status> 端口为configServer 查看加密状态

创建bootstrap.yml文件(必须)

**encrypt**:  
 **key**: jishixin #加密Key

(加密)

解密: <http://localhost:8080/decrypt>

**加密数据:**

