# Springcloud实战篇

Springcloud分布式框架是基于Springboot框架的，在以下的每一个服务中都要在Springboot工程的pom.xml中引入Springcloud的依赖。

<dependencyManagement>  
 <dependencies>  
 <dependency>  
 <groupId>org.springframework.cloud</groupId>  
 <artifactId>spring-cloud-dependencies</artifactId>  
 <version>Finchley.RELEASE</version>  
 <type>pom</type>  
 <scope>import</scope>  
 </dependency>  
 </dependencies>  
</dependencyManagement>

## 服务注册与发现

Springcloud中服务注册与发现功能一般使用Eureka作为服务注册与发现组件，该组件主要分为服务端和客户端。服务端充当注册中心，统一管理分布式系统中的各个服务；而客户端则是作为微服务端，要把自身注册到服务端，通过服务端去访问分布式系统中其他服务。

### EurekaServer

1、引入Eureka服务端依赖：在pom.xml文件中引入eureka-server依赖。

<dependency>  
 <groupId>org.springframework.cloud</groupId>  
 <artifactId>spring-cloud-starter-netflix-eureka-server</artifactId>  
</dependency>

2、配置EurekaServer：在application.yml配置文件中对EurekaServer进行配置。

**服务名：**是该服务在分布式系统中的别名（非常重要），分布式系统中都是使用服务名进行服务请求的，但对于EurekaServer而言，若不对EurekaServer进行集群的话，该服务名可以随便取（但建议养成良好习惯，不乱取名）。

**服务开启端口：**该服务在主机上开启的端口。

**服务主机名：**服务所在主机的ip，若本地访问可以设置localhsot

**EurekaServer特殊设置：**register-with-eureka（是否注册自身到注册中心）、fetch-registry（是否从注册中心获取注册信息）。EurekaServer其实本质也是客户端，这两个属性就决定了该服务的服务端性质。

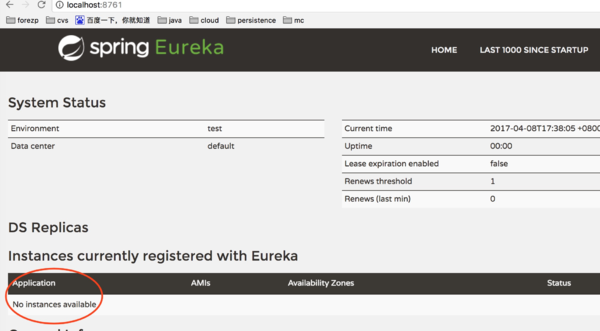
**服务路径：**在EurekaServer没有集群的情况下，EurekaClient都通过该路径访问EurekaServer服务，进行服务注册等操作。

**spring:  
 application:  
 name:** eureka-server *#服务名***server:  
 port:** 9001 *#服务开启端口***eureka:  
 instance:  
 hostname:** localhost *#服务主机名* **client:  
 register-with-eureka: false** *#是否注册自身到注册中心* **fetch-registry: false** *#是否从注册中心获取注册信息* **service-url:  
 defaultZone:** http://${**eureka.instance.hostname**}:${**server.port**}/eureka/ *#服务路径*

3、开启服务：将@EnableEurekaServer注解标注在Springboot启动类上，就可开启EurekaServer服务。

@SpringBootApplication  
@EnableEurekaServer //开启EurekaServer服务  
public class EurekaServerApplication {  
 public static void main(String[] args) {  
 SpringApplication.*run*(EurekaServerApplication.class, args);  
 }  
}

最后通过访问http://localhost:9001进入EurekaServer的控制台，其中DS Replicas处可以查看服务注册中心中已经注册的服务。并且可以看到服务的简略信息（端口，服务进程数等）。



### EurekaClient

1、引入Eureka客户端依赖：在pom.xml文件中引入，eureka-client依赖。

<dependency>  
 <groupId>org.springframework.cloud</groupId>  
 <artifactId>spring-cloud-starter-netflix-eureka-client</artifactId>  
</dependency>

2、配置EurekaClient：在application.yml配置文件中配置下相关信息。其中defaultZone配置注册中心的服务路径，而不是自身服务路径（客户端服务不需要通过服务路径访问，只需要通过服务名访问服务）。

**spring:  
 application:  
 name:** service-hi *#服务名***server:  
 port:** 9102 *#服务开启端口***eureka:  
 client:  
 serviceUrl:  
 defaultZone:** http://localhost:9001/eureka/ *#注册中心服务路径*

3、启动客户端服务：将@EnableEurekaClient注解标注在Springboot启动类上。其中@EnableEurekaClient注解可以用@EnableDiscoveryClient注解代替，但是两者有区别，详见原理篇。

@EnableEurekaClient //表示该项目为eureka注册中心客户端  
@SpringBootApplication  
public class EurekaClient01Application {  
 public static void main(String[] args) {  
 SpringApplication.*run*(EurekaClient01Application.class, args);  
 }  
}

## 客户端负载均衡

Springcloud中一般使用ribbon组件实现客户端负载均衡的功能，也可以用Feign组件（该组件集成了ribbon组件）。

### Ribbon+restTemplate

1、引入ribbon依赖：负载均衡要配合分布式系统使用，也就是该服务要注册到分布式系统中，因此不仅要引入ribbon依赖，还得引入EurekaClient依赖。由于EurekaClient依赖中已经集成了ribbon组件，因此不需要另外引入ribbon，只需要引入EurekaClient依赖即可。

<dependency>  
 <groupId>org.springframework.cloud</groupId>  
 <artifactId>spring-cloud-starter-netflix-eureka-client</artifactId>  
</dependency>

2、配置EurekaClient：步骤和服务注册与发现中一致。

3、配置Bean—>RestTemplate：配置一个RestTemplate Bean到ioc容器。

4、开启负载均衡：用@LoadBalance注解标注RestTemplate Bean以开启负载均衡。

@Bean  
@LoadBalanced //开启负载均衡  
RestTemplate restTemplate(){  
 return new RestTemplate();  
}

5、请求服务：利用RestTemplate的方法getForObject方法获取http请求的body，还可用getForEntity获取整个http请求的信息，还有post、put、delete等请求方式的方法。

public String hiService(String name) {  
 System.*out*.println("进入HtlloService.hiService()!");  
 return restTemplate.getForObject(

"http://SERVICE-HI/hi?name="+name,//请求路径，把主机名改为服务名，否则不可负载均衡

String.class //想要获取的返回值类型

);  
}

### Feign

Feign组件集成了ribbon组件，但Feign简化了我们的工作，只需要编写接口就可以实现负载据均衡的功能。

1、引入Feign依赖：在pom.xml文件中引入openfeign依赖。除此之外还需要引入EurekaClient

依赖，让该服务注册到分布式系统中。

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

<dependency>  
 <groupId>org.springframework.cloud</groupId>  
 <artifactId>spring-cloud-starter-netflix-eureka-client</artifactId>  
</dependency>

2、开启Feign功能：在启动类上用@EnableFeignClients注解，开启负载均衡。

@SpringBootApplication  
@EnableFeignClients //开启Feign功能  
public class ServiceFeignApplication {  
 public static void main(String[] args) {  
 SpringApplication.*run*(ServiceFeignApplication.class, args);  
 }  
}

3、请求服务：利用@FeignClient注解和@GetMapping来配置访问的服务路径，并且只需要编写接口即可。

@FeignClient(value = "service-hi") //需要访问的系统名，不能用下划线，否则报不正确的hostname格式  
public interface SchedualServiceHi {  
 @GetMapping("/hi") //需要访问的系统的具体服务的映射路径  
 String sayHiFromClientOne(@RequestParam(value = "name") String name);  
}

## 分布式配置

Springcloud中一般使用config组件实现分布式配置中心的功能，而分布式配置中心也分为服务端和客户端两种。

### Config Server

1、引入ConfigServer依赖：在pom.xml文件中引入spring-cloud-starter-config-server依赖。

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

2、开启ConfigServer功能：在启动类上用@EnableConfigServer注解标注，开启此功能。

@SpringBootApplication  
@EnableConfigServer //开启配置中心服务端功能  
public class ConfigServerApplication {  
 public static void main(String[] args) {  
 SpringApplication.*run*(ConfigServerApplication.class, args);  
 }  
}

3、配置ConfigServer：在application.yml中配置相关信息。

**spring**:  
 **application**:  
 **name**: config-server  
 **cloud**:  
 **config**:  
 **server**:  
 **git**:  
 **uri**: https://github.com/BuQingYan/Practice.git *#git仓库地址* **searchPaths**: SpringcloudConfigCenter *#配置文件路径，若有多个可以用“，”分隔* **username**:  
 **password**:  
 **label**: master *#配置文件所在分支***server**:  
 **port**: 8888

### Config Client

1、引入config依赖：除了spring-cloud-starter-config依赖外，还需要添加spring-boot-starter-web依赖，因为config依赖没有依赖web，启动时会因为tomcat启动不成功而启动失败。

<dependency>  
 <groupId>org.springframework.boot</groupId>  
 <artifactId>spring-boot-starter-web</artifactId>  
</dependency>  
<dependency>  
 <groupId>org.springframework.cloud</groupId>  
 <artifactId>spring-cloud-starter-config</artifactId>  
</dependency>

2、配置ConfigClient：在application.yml和bootstrap.yml中配置相关信息。配置中心相关信息必须配置在bootstrap.yml中，因为bootstrap的加载顺序较前，可以在项目启动前通过配置中心拿到对应的配置文件（在项目启动需要一些配置文件时尤为重要）；若放在appliction.yml中，有可能因为缺少项目启动时所需要的配置文件而启动失败。

application.yml

**spring**:  
 **application**:  
 **name**: config-client

**server**:  
 **port**: 8889

bootstrap.yml

**spring**:  
 **cloud**:  
 **config**:  
 *#这部分配置是配置中心的配置，要放在bootstrap中配置，否则启动可能报错* **label**: master *#配置文件所在分支* **profile**: dev *#配置文件的环境（开发、测试、生产）* **uri**: http://localhost:8888/ *#配置中心访问路径*

### 配置文件映射规则

配置文件会被转换成web请求，请求和配置文件命名映射如下：profile为分支

Web请求：<http://host:port/applicationName/profile>

文件命名：applicationName-profile.yml

applicationName-profile.properties

## 断路器

Springcloud中的断路器功能使用hystrix组件实现，主要用于防止雪崩效应的发生。

### Hystrix

1、引入hystrix依赖：当然此例子时建立在该类时eureka客户端的情况下的，在pom.xml文件中引入spring-cloud-starter-netflix-hystrix依赖。

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

2、开启hystrix功能：在配置EurekaClient基础上，在启动类上用@EnableHystrix注解标注，开启此功能。

@SpringBootApplication  
@EnableHystrix  
public class ServiceHystrixApplication {  
 public static void main(String[] args) {  
 SpringApplication.*run*(ServiceHystrixApplication.class, args);  
 }  
}

3、配置hystrix：在D版本后，hystrix功能就不是默认开启的了，在配置EurekaClient基础上，需要在application.yml中开启。

**feign**:  
 **hystrix**:  
 **enabled**: true *#打开断路器*

4、设置回调类：在需要使用断路器的方法/类上使用@HystrixCommand，并指定fallbackMethod属性，设置回调类。

@Service  
public class HelloService {  
 @Resource  
 RestTemplate restTemplate;  
 @HystrixCommand(fallbackMethod = "hiError")  
 public String hiService(String name) {  
 System.*out*.println("进入HtlloService.hiService()!");  
 return restTemplate.getForObject("http://SERVICE-HI/hi?name="+name,String.class);  
 }  
 public String hiError(String name){  
 return "服务调用失败,name="+name;  
 }  
}

## 路由网关

Springcloud中的路由网关功能可用zull组件实现。

### Zull

Zull组件的功能很多，包括路由、过滤、鉴权等，这里主要讲述路由和过滤功能的使用。以下各功能都需要在eureka-client的基础上引入zull依赖。

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

#### 路由

1、开启zull功能：在eureka-client的基础上，在启动类上用@EnableZullProxy注解标注，开启此功能。类似功能的注解还有@EnableZullServer注解，区别详见原理篇。

@SpringBootApplication  
@EnableZuulProxy  
public class ServiceZullApplication {  
 public static void main(String[] args) {  
 SpringApplication.*run*(ServiceZullApplication.class, args);  
 }  
}

2、配置路由信息：在eureka-client的基础上，在application.yml文件中配置zull路由信息。

**zuul**:  
 **routes**:  
 **api-a**: *#第一个路由配置，下面的以此类推* **path**: /api-a/\*\* *#所有路径中带有/api-a/的请求都转发到下面的服务* **serviceId**: service-ribbon *#满足路径要求时转到的服务的服务名* **api-b**:  
 **path**: /api-b/\*\*  
 **serviceId**: service-feign

zull有默认的路由规则，就是path：/service-name/\*\* serviceId：service-name，zull会对在同一个分布式系统中的所有服务（即在同一注册中心[ 集群 ]注册了的服务）进行以上规则的路由配置。

#### 过滤

1. 继承zullFilter类：新建自己的过滤器继承ZullFilter类。并实现其中的方法。

@Component  
public class MyZullFilter extends ZuulFilter {  
 @Override //过滤器类型：路由前pre、路由时routing、路由后post、错误时error  
 public String filterType() { return "pre"; }  
 @Override //过滤器过滤顺序，数字越小越先过滤  
 public int filterOrder() { return 0; }  
 @Override //过滤器是否执行，此处可以进行逻辑判断，决定是否执行过滤器  
 public boolean shouldFilter() { return true; }  
 @Override //过滤器具体业务操作  
 public Object run() throws ZuulException {  
 RequestContext requestContext = RequestContext.*getCurrentContext*();  
 String filterFlag = requestContext.getRequest().getParameter("filterFlag");  
 if (null != filterFlag && "false".equals(filterFlag)){  
 try {  
 requestContext.setSendZuulResponse(false);  
 requestContext.setResponseStatusCode(401);  
 requestContext.getResponse().getWriter().write("zull过滤器拦截");  
 } catch (IOException e) {  
 e.printStackTrace();  
 }  
 }  
 return null;  
 }  
}

# Springcloud原理篇

## 服务注册与发现

## 客户端负载均衡

## 分布式配置

## 断路器

## 服务网关