# 安装Eureka服务端

## Pom.xml文件新增

<dependency>

<groupId>org.springframework.cloud</groupId>

<artifactId>spring-cloud-starter-netflix-eureka-server</artifactId>

</dependency>

## 启用服务注册中心

@EnableEurekaServer

@SpringBootApplication

public class EurekaserverApplication {

public static void main(String[] args) {

SpringApplication.run(EurekaserverApplication.class, args);

}

}

## Application-xxx.ml中配置

server:

port: 8761

eureka:

instance:

hostname: localhost

client:

registerWithEureka: false

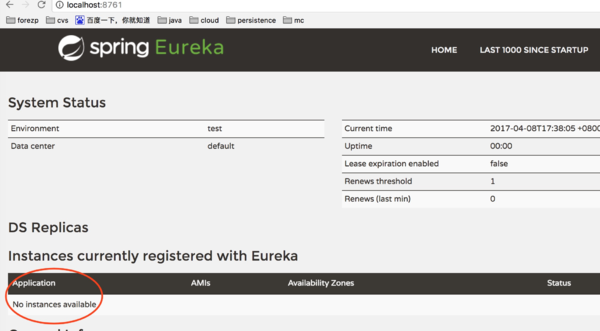
fetchRegistry: false

serviceUrl:

defaultZone: http://${eureka.instance.hostname}:${server.port}/eureka/

## 运行

启动工程，然后访问：[http://localhost:8761](http://localhost:8761/)



# 配置Eureka客户端

## Pom文件新增

<dependency>

<groupId>org.springframework.cloud</groupId>

<artifactId>spring-cloud-starter-netflix-eureka-client</artifactId>

</dependency>

## XxxApplication新增

新增@EnableEurekaClient表明项目是一个Eureka的客户端

@SpringBootApplication

@EnableEurekaClient

public class ServiceHiApplication {

public static void main(String[] args) {

SpringApplication.run(ServiceHiApplication.class, args);

}

}

## 标明服务注册中心的地址

eureka:

client:

serviceUrl:

defaultZone: http://localhost:8761/eureka/

server:

port: 8762

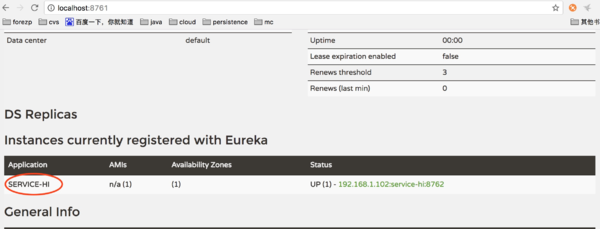
spring:

application:

name: service-hi

## 运行

一个服务已经注册在服务中了，服务名为SERVICE-HI ,端口为7862



# Eureka服务端的高可用配置

在实际项目中，可能有几十个甚至几百个的微服务实例，这时Eureka Server承担了非常高的负载，由于Eureka Server在微服务架构中有着举足轻重的作用，所以需要怼Eureka Server进行高可用配置。

application.yml在配置文件中采用多profile的格式

server:

port: 8761

eureka:

instance:

hostname: peer1

client:

serviceUril:

defaultZone: <http://peer2:8762/eureka/>

server:

port: 8762

eureka:

instance:

hostname: peer2

client:

serviceUril:

defaultZone: <http://peer1:8761/eureka/>

如果是在一台机器上部署，则需要修改hosts文件，如果多台机器，则写ip或者域名。

127.0.0.1 peer1

127.0.0.1 peer2

启动：

java -jar xxx.jar –spring.profiles.active=peer1

java -jar xxx.jar –spring.profiles.active=peer2

# 配置Eureka客户端负载均衡器Feign

Feign是一个声明式的伪Http客户端，它使得写Http客户端变得更简单。使用Feign，只需要创建一个接口并注解。它具有可插拔的注解特性，可使用Feign 注解和JAX-RS注解。Feign支持可插拔的编码器和解码器。Feign默认集成了Ribbon，并和Eureka结合，默认实现了负载均衡的效果。

## Application中新增@EnableFeignClients

@SpringBootApplication

@EnableEurekaClient

@EnableFeignClients

public class ServiceFeignApplication {

public static void main(String[] args) {

SpringApplication.run(ServiceFeignApplication.class, args);

}

}

## 定义一个Feign接口

@FeignClient(value = "service-hi")

public interface SchedualServiceHi {

@RequestMapping(value = "/hi",method = RequestMethod.GET)

String sayHiFromClientOne(String name);

}

## Feign接口使用

@RestController

public class HiController {

@Autowired

SchedualServiceHi schedualServiceHi;

@RequestMapping(value = "/hi",method = RequestMethod.GET)

public String sayHi(@RequestParam String name){

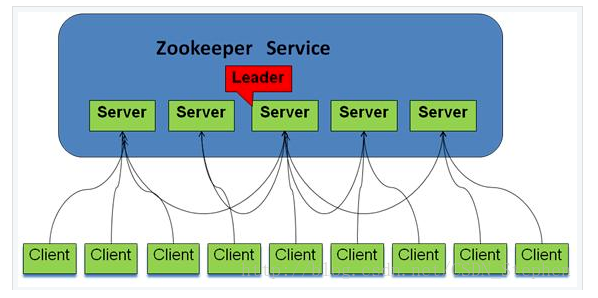
return schedualServiceHi.sayHiFromClientOne(name);

}

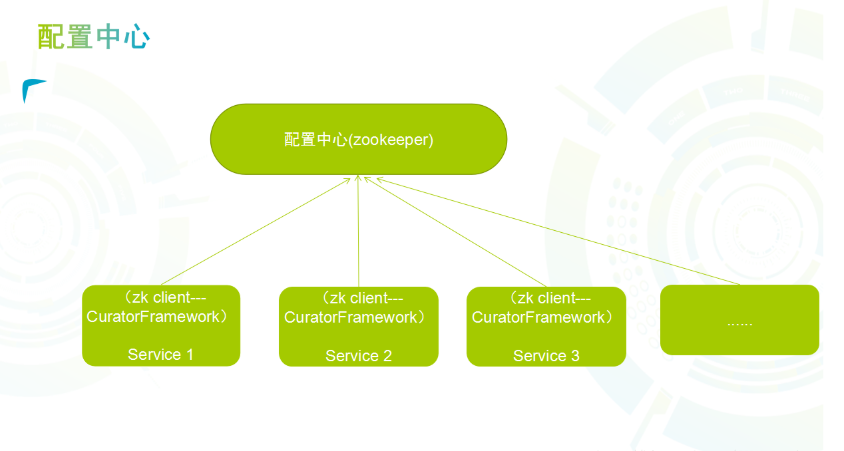
}

# 分布式配置中心

## Zookeeper集群架构图



## Spring Cloud Config架构图



## Pom文件新增

<dependency>

<groupId>org.springframework.cloud</groupId>

<artifactId>spring-cloud-starter-zookeeper-config</artifactId>

</dependency>

<dependency>

<groupId>org.springframework.cloud</groupId>

<artifactId>spring-boot-starter-actuator</artifactId>

</dependency>

## Bootstrap.yml进行如下配置

spring:

application:

name: config-zookeeper

cloud:

zookeeper:

enabled: true

connect-string: localhost:2181

config:

root: /config

enabale: true

watcher:

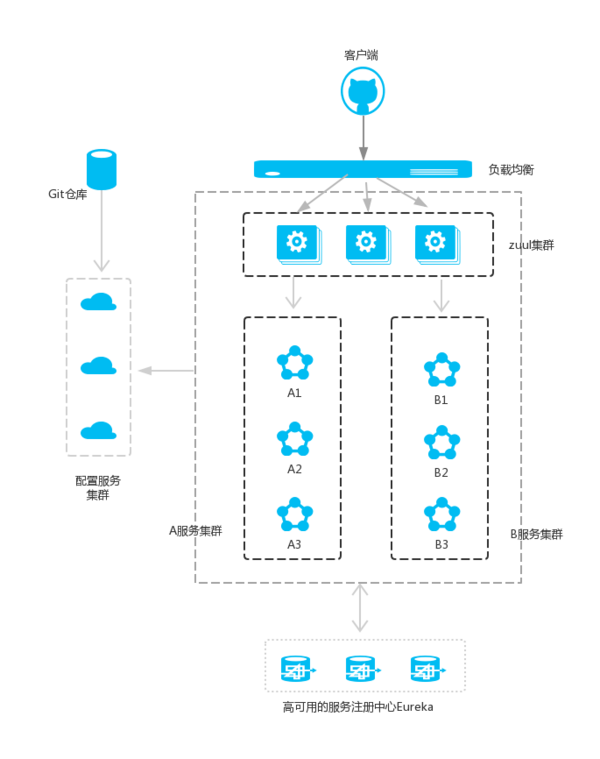
enable: true

profiles:  
 active: dev

# 路由网关Zuul

## 路由网关在分布式系统中所处的位置和角色

在微服务架构中，需要几个基础的服务治理组件，包括服务注册与发现、服务消费、负载均衡、断路器、智能路由、配置管理等，由这几个基础组件相互协作，共同组建了一个简单的微服务系统。一个简答的微服务系统如下图：



## 新增依赖

<dependency>

<groupId>org.springframework.cloud</groupId>

<artifactId>spring-cloud-starter-zuul</artifactId>

</dependency>

## 入口Application类上加注解@EnableZuulProxy

@EnableZuulProxy

@EnableEurekaClient

@SpringBootApplication

public class ServiceZuulApplication {

public static void main(String[] args) {

SpringApplication.run(ServiceZuulApplication.class, args);

}

}

## application.yml文件加上如下配置：

zuul:

routes:

api-a:

path: /api-a/\*\*

serviceId: service-ribbon

api-b:

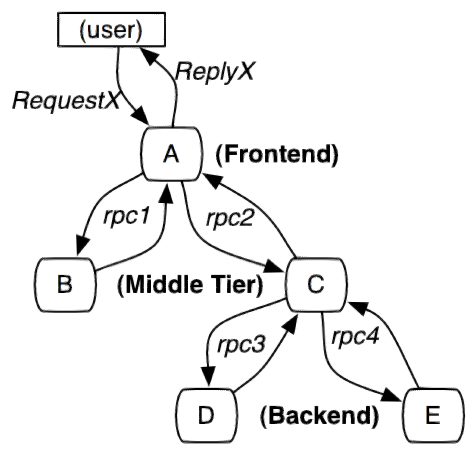
path: /api-b/\*\*

serviceId: service-feign

以/api-a/ 开头的请求都转发给service-ribbon服务；以/api-b/开头的请求都转发给service-feign服务；

# 服务链路追踪

服务追踪组件zipkin，Spring Cloud Sleuth集成了zipkin组件。



## 搭建zipkin server

Zipkin server用来收集调用数据，并展示。

### Pom文件新增依赖

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter</artifactId>

</dependency>

<dependency>

<groupId>io.zipkin.java</groupId>

<artifactId>zipkin-server</artifactId>

</dependency>

<dependency>

<groupId>io.zipkin.java</groupId>

<artifactId>zipkin-autoconfigure-ui</artifactId>

</dependency>

### Application 加@EnableZipkinServer

@SpringBootApplication

@EnableZipkinServer

public class ServerZipkinApplication {

public static void main(String[] args) {

SpringApplication.run(ServerZipkinApplication.class, args);

}

}

### Yml配置

server:  
 port: 7001  
management:  
 metrics:  
 web:  
 server:  
 auto-time-requests: false  
spring:  
 application:  
 name: zip-server

## 创建Service

### 服务加入zipkin依赖

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

### Yml进行配置

server:  
 port: 8002  
spring:  
 zipkin:  
 base-url: http://localhost:7001  
 application:  
 name: zipkin-2  
 sleuth:  
 sampler:  
 probability: 1.0

