转载： <http://blog.csdn.net/forezp/article/details/70148833> 本文出自方志朋的博客

# 第一篇: 服务的注册与发现（Eureka）

## 一、spring cloud简介

spring cloud 为开发人员提供了快速构建分布式系统的一些工具，包括配置管理、服务发现、断路器、路由、微代理、事件总线、全局锁、决策竞选、分布式会话等等。它运行环境简单，可以在开发人员的电脑上跑。另外说明spring cloud是基于springboot的，所以需要开发中对springboot有一定的了解，如果不了解的话可以看这篇文章：[2小时学会springboot](http://blog.csdn.net/forezp/article/details/61472783" \t "_blank)。另外对于“微服务架构” 不了解的话，可以通过搜索引擎搜索“微服务架构”了解下。

## 二、创建服务注册中心

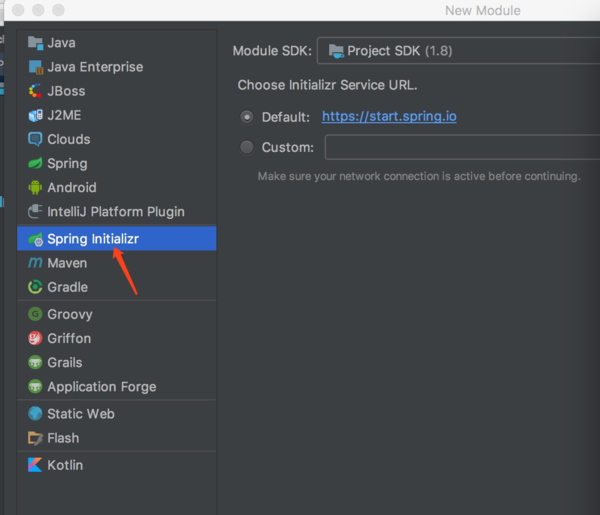
在这里，我们需要用的的组件上Spring Cloud Netflix的Eureka ,eureka是一个服务注册和发现模块。

**2.1 首先创建一个maven主工程。**

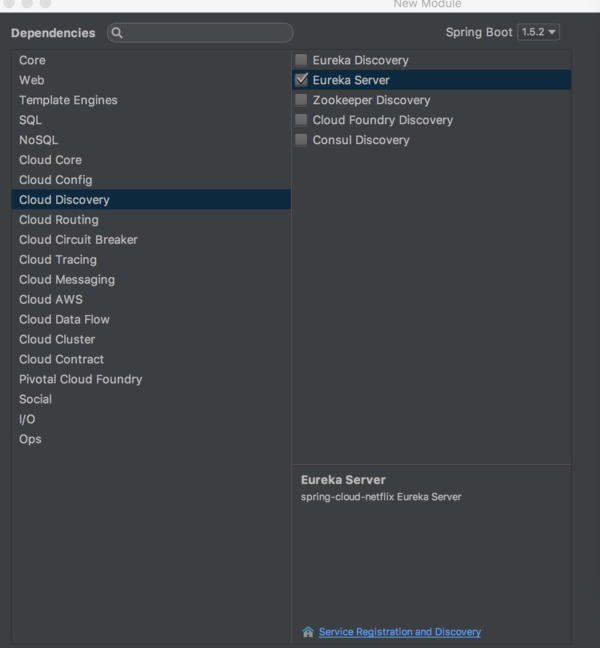
**2.2 然后创建2个model工程:**一个model工程作为服务注册中心，即Eureka Server,另一个作为Eureka Client。

下面以创建server为例子，详细说明创建过程：

右键工程->创建model-> 选择spring initialir 如下图：



下一步->选择cloud discovery->eureka server ,然后一直下一步就行了。



创建完后的工程的pom.xml文件如下：

<?xml version="1.0" encoding="UTF-8"?>

<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">

<modelVersion>4.0.0</modelVersion>

<groupId>com.forezp</groupId>

<artifactId>eurekaserver</artifactId>

<version>0.0.1-SNAPSHOT</version>

<packaging>jar</packaging>

<name>eurekaserver</name>

<description>Demo project for Spring Boot</description>

<parent>

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

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

<version>1.5.2.RELEASE</version>

<relativePath/> <!-- lookup parent from repository -->

</parent>

<properties>

<project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>

<project.reporting.outputEncoding>UTF-8</project.reporting.outputEncoding>

<java.version>1.8</java.version>

</properties>

<dependencies>

<!--eureka server -->

<dependency>

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

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

</dependency>

<!-- spring boot test-->

<dependency>

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

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

<scope>test</scope>

</dependency>

</dependencies>

<dependencyManagement>

<dependencies>

<dependency>

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

<artifactId>spring-cloud-dependencies</artifactId>

<version>Dalston.RC1</version>

<type>pom</type>

<scope>import</scope>

</dependency>

</dependencies>

</dependencyManagement>

<build>

<plugins>

<plugin>

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

<artifactId>spring-boot-maven-plugin</artifactId>

</plugin>

</plugins>

</build>

<repositories>

<repository>

<id>spring-milestones</id>

<name>Spring Milestones</name>

<url>https://repo.spring.io/milestone</url>

<snapshots>

<enabled>false</enabled>

</snapshots>

</repository>

</repositories>

</project>

**2.3 启动一个服务注册中心**，只需要一个注解@EnableEurekaServer，这个注解需要在springboot工程的启动application类上加：

@EnableEurekaServer

@SpringBootApplication

public class EurekaserverApplication {

public static void main(String[] args) {

SpringApplication.run(EurekaserverApplication.class, args);

}

}

\*\*2.4 \*\*eureka是一个高可用的组件，它没有后端缓存，每一个实例注册之后需要向注册中心发送心跳（因此可以在内存中完成），在默认情况下erureka server也是一个eureka client ,必须要指定一个 server。eureka server的配置文件appication.yml：

server:

port: 8761

eureka:

instance:

hostname: localhost

client:

registerWithEureka: false

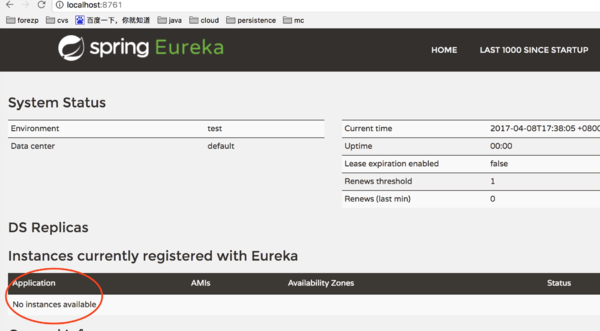
fetchRegistry: false

serviceUrl:

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

通过eureka.client.registerWithEureka：false和fetchRegistry：false来表明自己是一个eureka server.

**2.5** eureka server 是有界面的，启动工程,打开浏览器访问：   
<http://localhost:8761> ,界面如下：



No application available 没有服务被发现 ……^\_^   
因为没有注册服务当然不可能有服务被发现了。

## 三、创建一个服务提供者 (eureka client)

当client向server注册时，它会提供一些元数据，例如主机和端口，URL，主页等。Eureka server 从每个client实例接收心跳消息。 如果心跳超时，则通常将该实例从注册server中删除。

创建过程同server类似,创建完pom.xml如下：

<?xml version="1.0" encoding="UTF-8"?>

<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">

<modelVersion>4.0.0</modelVersion>

<groupId>com.forezp</groupId>

<artifactId>service-hi</artifactId>

<version>0.0.1-SNAPSHOT</version>

<packaging>jar</packaging>

<name>service-hi</name>

<description>Demo project for Spring Boot</description>

<parent>

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

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

<version>1.5.2.RELEASE</version>

<relativePath/> <!-- lookup parent from repository -->

</parent>

<properties>

<project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>

<project.reporting.outputEncoding>UTF-8</project.reporting.outputEncoding>

<java.version>1.8</java.version>

</properties>

<dependencies>

<dependency>

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

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

</dependency>

<dependency>

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

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

</dependency>

<dependency>

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

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

<scope>test</scope>

</dependency>

</dependencies>

<dependencyManagement>

<dependencies>

<dependency>

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

<artifactId>spring-cloud-dependencies</artifactId>

<version>Dalston.RC1</version>

<type>pom</type>

<scope>import</scope>

</dependency>

</dependencies>

</dependencyManagement>

<build>

<plugins>

<plugin>

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

<artifactId>spring-boot-maven-plugin</artifactId>

</plugin>

</plugins>

</build>

<repositories>

<repository>

<id>spring-milestones</id>

<name>Spring Milestones</name>

<url>https://repo.spring.io/milestone</url>

<snapshots>

<enabled>false</enabled>

</snapshots>

</repository>

</repositories>

</project>

通过注解@EnableEurekaClient 表明自己是一个eurekaclient.

@SpringBootApplication

@EnableEurekaClient

@RestController

public class ServiceHiApplication {

public static void main(String[] args) {

SpringApplication.run(ServiceHiApplication.class, args);

}

@Value("${server.port}")

String port;

@RequestMapping("/hi")

public String home(@RequestParam String name) {

return "hi "+name+",i am from port:" +port;

}

}

仅仅@EnableEurekaClient是不够的，还需要在配置文件中注明自己的服务注册中心的地址，application.yml配置文件如下：

eureka:

client:

serviceUrl:

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

server:

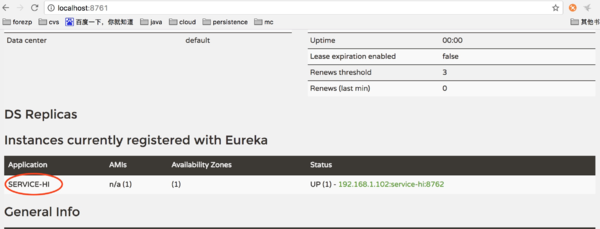
port: 8762

spring:

application:

name: service-hi

需要指明spring.application.name,这个很重要，这在以后的服务与服务之间相互调用一般都是根据这个name 。   
启动工程，打开<http://localhost:8761> ，即eureka server 的网址：



你会发现一个服务已经注册在服务中了，服务名为SERVICE-HI ,端口为7862

这时打开 <http://localhost:8762/hi?name=forezp> ，你会在浏览器上看到 :

hi forezp,i am from port:8762

# 第二篇: 服务消费者（rest+ribbon）

在微服务架构中，业务都会被拆分成一个独立的服务，服务与服务的通讯是基于http restful的。Spring cloud有两种服务调用方式，一种是ribbon+restTemplate，另一种是feign。在这一篇文章首先讲解下基于ribbon+rest。

### 一、ribbon简介

Ribbon is a client side load balancer which gives you a lot of control over the behaviour of HTTP and TCP clients. Feign already uses Ribbon, so if you are using @FeignClient then this section also applies.

—–摘自官网

ribbon是一个负载均衡客户端，可以很好的控制http和tcp的一些行为。Feign默认集成了ribbon。

ribbon 已经默认实现了这些配置bean：

* IClientConfig ribbonClientConfig: DefaultClientConfigImpl
* IRule ribbonRule: ZoneAvoidanceRule
* IPing ribbonPing: NoOpPing
* ServerList ribbonServerList: ConfigurationBasedServerList
* ServerListFilter ribbonServerListFilter: ZonePreferenceServerListFilter
* ILoadBalancer ribbonLoadBalancer: ZoneAwareLoadBalancer

### 二、准备工作

这一篇文章基于上一篇文章的工程，启动eureka-server 工程；启动service-hi工程，它的端口为8762；将service-hi的配置文件的端口改为8763,并启动，这时你会发现：service-hi在eureka-server注册了2个实例，这就相当于一个小的集群。访问localhost:8761如图所示：



### 三、建一个服务消费者

重新新建一个spring-boot工程，取名为：service-ribbon;   
在它的pom.xml文件分别引入起步依赖spring-cloud-starter-eureka、spring-cloud-starter-ribbon、spring-boot-starter-web，代码如下：

<?xml version="1.0" encoding="UTF-8"?>

<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">

<modelVersion>4.0.0</modelVersion>

<groupId>com.forezp</groupId>

<artifactId>service-ribbon</artifactId>

<version>0.0.1-SNAPSHOT</version>

<packaging>jar</packaging>

<name>service-ribbon</name>

<description>Demo project for Spring Boot</description>

<parent>

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

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

<version>1.5.2.RELEASE</version>

<relativePath/> <!-- lookup parent from repository -->

</parent>

<properties>

<project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>

<project.reporting.outputEncoding>UTF-8</project.reporting.outputEncoding>

<java.version>1.8</java.version>

</properties>

<dependencies>

<dependency>

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

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

</dependency>

<dependency>

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

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

</dependency>

<dependency>

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

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

</dependency>

<dependency>

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

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

<scope>test</scope>

</dependency>

</dependencies>

<dependencyManagement>

<dependencies>

<dependency>

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

<artifactId>spring-cloud-dependencies</artifactId>

<version>Dalston.RC1</version>

<type>pom</type>

<scope>import</scope>

</dependency>

</dependencies>

</dependencyManagement>

<build>

<plugins>

<plugin>

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

<artifactId>spring-boot-maven-plugin</artifactId>

</plugin>

</plugins>

</build>

<repositories>

<repository>

<id>spring-milestones</id>

<name>Spring Milestones</name>

<url>https://repo.spring.io/milestone</url>

<snapshots>

<enabled>false</enabled>

</snapshots>

</repository>

</repositories>

</project>

在工程的配置文件指定服务的注册中心地址为[http://localhost:8761/eureka/](http://localhost:8761/eureka/" \t "_blank)，程序名称为 service-ribbon，程序端口为8764。配置文件application.yml如下：

eureka:

client:

serviceUrl:

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

server:

port: 8764

spring:

application:

name: service-ribbon

在工程的启动类中,通过@EnableDiscoveryClient向服务中心注册；并且向程序的ioc注入一个bean: restTemplate;并通过@LoadBalanced注解表明这个restRemplate开启负载均衡的功能。

@SpringBootApplication

@EnableDiscoveryClient

public class ServiceRibbonApplication {

public static void main(String[] args) {

SpringApplication.run(ServiceRibbonApplication.class, args);

}

@Bean

@LoadBalanced

RestTemplate restTemplate() {

return new RestTemplate();

}

}

写一个测试类HelloService，通过之前注入ioc容器的restTemplate来消费service-hi服务的“/hi”接口，在这里我们直接用的程序名替代了具体的url地址，在ribbon中它会根据服务名来选择具体的服务实例，根据服务实例在请求的时候会用具体的url替换掉服务名，代码如下：

@Service

public class HelloService {

@Autowired

RestTemplate restTemplate;

public String hiService(String name) {

return restTemplate.getForObject("http://SERVICE-HI/hi?name="+name,String.class);

}

}

写一个controller，在controller中用调用HelloService 的方法，代码如下：

@RestController

public class HelloControler {

@Autowired

HelloService helloService;

@RequestMapping(value = "/hi")

public String hi(@RequestParam String name){

return helloService.hiService(name);

}

}

在浏览器上多次访问[http://localhost:8764/hi?name=forezp](http://localhost:8764/hi?name=forezp" \t "_blank)，浏览器交替显示：

hi forezp,i am from port:8762

hi forezp,i am from port:8763

这说明当我们通过调用restTemplate.getForObject(“<http://SERVICE-HI/hi?name=>“+name,String.class)方法时，已经做了负载均衡，访问了不同的端口的服务实例。

### 四、此时的架构



* 一个服务注册中心，eureka server,端口为8761
* service-hi工程跑了两个实例，端口分别为8762,8763，分别向服务注册中心注册
* sercvice-ribbon端口为8764,向服务注册中心注册
* 当sercvice-ribbon通过restTemplate调用service-hi的hi接口时，因为用ribbon进行了负载均衡，会轮流的调用service-hi：8762和8763 两个端口的hi接口；

# 第三篇: 服务消费者（Feign）

这篇文章主要讲述如何通过Feign去消费服务。

### 一、Feign简介

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。

简而言之：

* Feign 采用的是基于接口的注解
* Feign 整合了ribbon

### 二、准备工作

继续用上一节的工程， 启动eureka-server，端口为8761; 启动service-hi 两次，端口分别为8762 、8773.

### 三、创建一个feign的服务

新建一个spring-boot工程，取名为serice-feign，在它的pom文件引入Feign的起步依赖spring-cloud-starter-feign、Eureka的起步依赖spring-cloud-starter-eureka、Web的起步依赖spring-boot-starter-web，代码如下：

<?xml version="1.0" encoding="UTF-8"?>

<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">

<modelVersion>4.0.0</modelVersion>

<groupId>com.forezp</groupId>

<artifactId>service-feign</artifactId>

<version>0.0.1-SNAPSHOT</version>

<packaging>jar</packaging>

<name>service-feign</name>

<description>Demo project for Spring Boot</description>

<parent>

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

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

<version>1.5.2.RELEASE</version>

<relativePath/> <!-- lookup parent from repository -->

</parent>

<properties>

<project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>

<project.reporting.outputEncoding>UTF-8</project.reporting.outputEncoding>

<java.version>1.8</java.version>

</properties>

<dependencies>

<dependency>

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

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

</dependency>

<dependency>

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

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

</dependency>

<dependency>

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

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

</dependency>

<dependency>

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

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

<scope>test</scope>

</dependency>

</dependencies>

<dependencyManagement>

<dependencies>

<dependency>

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

<artifactId>spring-cloud-dependencies</artifactId>

<version>Dalston.RC1</version>

<type>pom</type>

<scope>import</scope>

</dependency>

</dependencies>

</dependencyManagement>

<build>

<plugins>

<plugin>

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

<artifactId>spring-boot-maven-plugin</artifactId>

</plugin>

</plugins>

</build>

<repositories>

<repository>

<id>spring-milestones</id>

<name>Spring Milestones</name>

<url>https://repo.spring.io/milestone</url>

<snapshots>

<enabled>false</enabled>

</snapshots>

</repository>

</repositories>

</project>

在工程的配置文件application.yml文件，指定程序名为service-feign，端口号为8765，服务注册地址为[http://localhost:8761/eureka/](http://localhost:8761/eureka/" \t "_blank) ，代码如下：

eureka:

client:

serviceUrl:

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

server:

port: 8765

spring:

application:

name: service-feign

在程序的启动类ServiceFeignApplication ，加上@EnableFeignClients注解开启Feign的功能：

@SpringBootApplication

@EnableDiscoveryClient

@EnableFeignClients

public class ServiceFeignApplication {

public static void main(String[] args) {

SpringApplication.run(ServiceFeignApplication.class, args);

}

}

定义一个feign接口，通过@ FeignClient（“服务名”），来指定调用哪个服务。比如在代码中调用了service-hi服务的“/hi”接口，代码如下：

@FeignClient(value = "service-hi")

public interface SchedualServiceHi {

@GetMapping(value = "/hi")

String sayHiFromClientOne(@RequestParam(value = "name") String name);

}

在Web层的controller层，对外暴露一个”/hi”的API接口，通过上面定义的Feign客户端SchedualServiceHi 来消费服务。代码如下：

@RestController

public class HiController {

@Autowired

SchedualServiceHi schedualServiceHi;

@GetMapping(value = "/hi")

public String sayHi(@RequestParam String name){

return schedualServiceHi.sayHiFromClientOne(name);

}

}

启动程序，多次访问[http://localhost:8765/hi?name=forezp](http://localhost:8765/hi?name=forezp" \t "_blank),浏览器交替显示：

hi forezp,i am from port:8762

hi forezp,i am from port:8763

# 第四篇:断路器（Hystrix）

在微服务架构中，根据业务来拆分成一个个的服务，服务与服务之间可以相互调用（RPC），在Spring Cloud可以用RestTemplate+Ribbon和Feign来调用。为了保证其高可用，单个服务通常会集群部署。由于网络原因或者自身的原因，服务并不能保证100%可用，如果单个服务出现问题，调用这个服务就会出现线程阻塞，此时若有大量的请求涌入，Servlet容器的线程资源会被消耗完毕，导致服务瘫痪。服务与服务之间的依赖性，故障会传播，会对整个微服务系统造成灾难性的严重后果，这就是服务故障的“雪崩”效应。

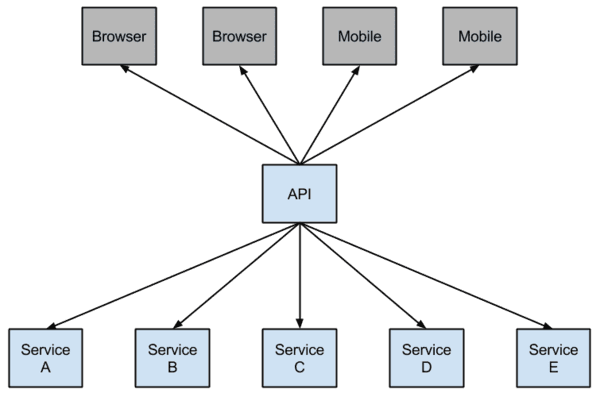
为了解决这个问题，业界提出了断路器模型。

### 一、断路器简介

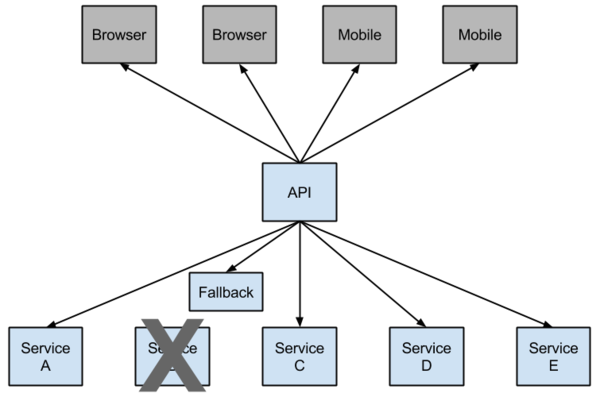
Netflix has created a library called Hystrix that implements the circuit breaker pattern. In a microservice architecture it is common to have multiple layers of service calls.

. —-摘自官网

Netflix开源了Hystrix组件，实现了断路器模式，SpringCloud对这一组件进行了整合。 在微服务架构中，一个请求需要调用多个服务是非常常见的，如下图：



较底层的服务如果出现故障，会导致连锁故障。当对特定的服务的调用的不可用达到一个阀值（Hystric 是5秒20次） 断路器将会被打开。



断路打开后，可用避免连锁故障，fallback方法可以直接返回一个固定值。

### 二、准备工作

这篇文章基于上一篇文章的工程，首先启动上一篇文章的工程，启动eureka-server 工程；启动service-hi工程，它的端口为8762。

### 三、在ribbon使用断路器

改造serice-ribbon 工程的代码，首先在pox.xml文件中加入spring-cloud-starter-hystrix的起步依赖：

<dependency>

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

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

</dependency>

在程序的启动类ServiceRibbonApplication 加@EnableHystrix注解开启Hystrix：

@SpringBootApplication

@EnableDiscoveryClient

@EnableHystrix

public class ServiceRibbonApplication {

public static void main(String[] args) {

SpringApplication.run(ServiceRibbonApplication.class, args);

}

@Bean

@LoadBalanced

RestTemplate restTemplate() {

return new RestTemplate();

}

}

改造HelloService类，在hiService方法上加上@HystrixCommand注解。该注解对该方法创建了熔断器的功能，并指定了fallbackMethod熔断方法，熔断方法直接返回了一个字符串，字符串为”hi,”+name+”,sorry,error!”，代码如下：

@Service

public class HelloService {

@Autowired

RestTemplate restTemplate;

@HystrixCommand(fallbackMethod = "hiError")

public String hiService(String name) {

return restTemplate.getForObject("http://SERVICE-HI/hi?name="+name,String.class);

}

public String hiError(String name) {

return "hi,"+name+",sorry,error!";

}

}

启动：service-ribbon 工程，当我们访问[http://localhost:8764/hi?name=forezp](http://localhost:8764/hi?name=forezp" \t "_blank),浏览器显示：

hi forezp,i am from port:8762

此时关闭 service-hi 工程，当我们再访问[http://localhost:8764/hi?name=forezp](http://localhost:8764/hi?name=forezp" \t "_blank)，浏览器会显示：

hi ,forezp,orry,error!

这就说明当 service-hi 工程不可用的时候，service-ribbon调用 service-hi的API接口时，会执行快速失败，直接返回一组字符串，而不是等待响应超时，这很好的控制了容器的线程阻塞。

### 四、Feign中使用断路器

Feign是自带断路器的，在D版本的Spring Cloud中，它没有默认打开。需要在配置文件中配置打开它，在配置文件加以下代码：

feign.hystrix.enabled=true

基于service-feign工程进行改造，只需要在FeignClient的SchedualServiceHi接口的注解中加上fallback的指定类就行了：

@FeignClient(value = "service-hi",fallback = SchedualServiceHiHystric.class)

public interface SchedualServiceHi {

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

String sayHiFromClientOne(@RequestParam(value = "name") String name);

}

SchedualServiceHiHystric需要实现SchedualServiceHi 接口，并注入到Ioc容器中，代码如下：

@Component

public class SchedualServiceHiHystric implements SchedualServiceHi {

@Override

public String sayHiFromClientOne(String name) {

return "sorry "+name;

}

}

启动四servcie-feign工程，浏览器打开[http://localhost:8765/hi?name=forezp](http://localhost:8765/hi?name=forezp" \t "_blank),注意此时service-hi工程没有启动，网页显示：

sorry forezp

打开service-hi工程，再次访问，浏览器显示：

hi forezp,i am from port:8762

这证明断路器起到作用了。

### 五、Hystrix Dashboard (断路器：Hystrix 仪表盘)

基于service-ribbon 改造，Feign的改造和这一样。

首选在pom.xml引入spring-cloud-starter-hystrix-dashboard的起步依赖：

<dependency>

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

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

</dependency>

<dependency>

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

<artifactId>spring-cloud-starter-hystrix-dashboard</artifactId>

</dependency>

在主程序启动类中加入@EnableHystrixDashboard注解，开启hystrixDashboard：

@SpringBootApplication

@EnableDiscoveryClient

@EnableHystrix

@EnableHystrixDashboard

public class ServiceRibbonApplication {

public static void main(String[] args) {

SpringApplication.run(ServiceRibbonApplication.class, args);

}

@Bean

@LoadBalanced

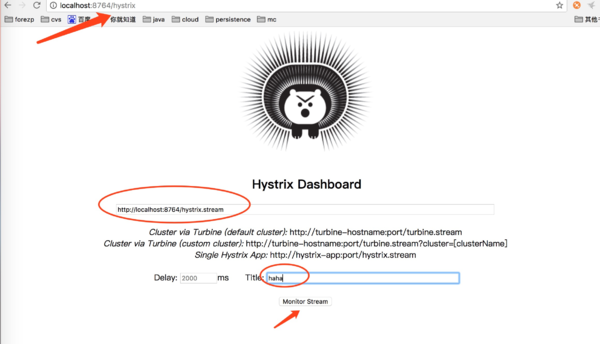
RestTemplate restTemplate() {

return new RestTemplate();

}

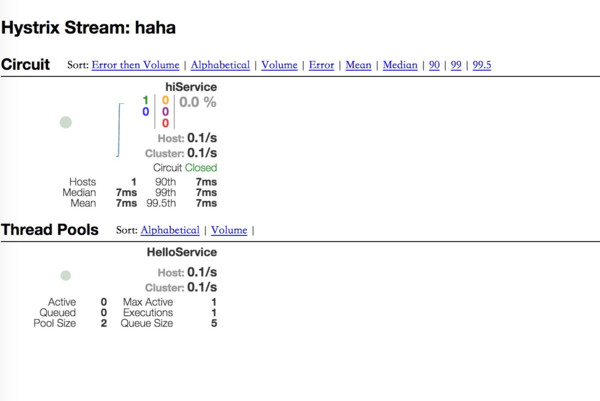
}

打开浏览器：访问[http://localhost:8764/hystrix](http://localhost:8764/hystrix" \t "_blank),界面如下：



点击monitor stream，进入下一个界面，访问：[http://localhost:8764/hi?name=forezp](http://localhost:8764/hi?name=forezp" \t "_blank)

此时会出现监控界面：



# 第五篇: 路由网关(zuul)