**SpringCloud**

# 微服务架构

## 什么是分布式

不同模块部署在不同服务器上

作用：分布式解决网站高并发带来问题

## 什么是集群

多台服务器部署相同应用构成一个集群

作用：通过负载均衡设备共同对外提供服务

## 什么是RPC

RPC 的全称是 Remote Procedure Call 是一种进程间通信方式。  
它允许程序调用另一个地址空间（通常是共享网络的另一台机器上）的过程或函数，而不用程序员显式编码这个远程调用的细节。即无论是调用本地接口/服务的还是远程的接口/服务，本质上编写的调用代码基本相同。  
**比如两台服务器A，B，一个应用部署在A服务器上，想要调用B服务器上应用提供的函数或者方法，由于不在一个内存空间，不能直接调用，这时候需要通过RPC框架的实现来解决**

### restful、soap、rpc

（1）RESTful是一种架构设计风格，提供了设计原则和约束条件，而不是架构。而满足这些约束条件和原则的应用程序或设计就是 RESTful架构或服务。  
（2）SOAP，简单对象访问协议是一种数据交换协议规范  
是一种轻量的、简单的、基于XML的协议的规范。SOAP协议和HTTP协议一样，都是底层的通信协议，只是请求包的格式不同而已，SOAP包是XML格式的。  
 SOAP的消息是基于xml并封装成了符合http协议，因此，它符合任何路由器、 防火墙或代理服务器的要求。  
 SOAP可以使用任何语言来完成，只要发送正确的soap请求即可，基于soap的服务可以在任何平台无需修改即可正常使用。  
（3）RPC就是从一台机器（客户端）上通过参数传递的方式调用另一台机器（服务器）上的一个函数或方法（可以统称为服务）并得到返回的结果。  
RPC 会隐藏底层的通讯细节（不需要直接处理Socket通讯或Http通讯）  
RPC 是一个请求响应模型。客户端发起请求，服务器返回响应（类似于Http的工作方式）  
RPC 在使用形式上像调用本地函数（或方法）一样去调用远程的函数（或方法）。

### rpc远程调用框架

几种比较典型的RPC的实现和调用框架。   
（1）RMI实现，利用java.rmi包实现，基于Java远程方法协议(Java Remote Method Protocol) 和java的原生序列化。   
（2）Hession，是一个轻量级的remoting onhttp工具，使用简单的方法提供了RMI的功能。 基于HTTP协议，采用二进制编解码。   
（3）thrift是一种可伸缩的跨语言服务的软件框架。thrift允许你定义一个描述文件，描述数据类型和服务接口。依据该文件，编译器方便地生成RPC客户端和服务器通信代码。

（4）SpringCloud 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。

## 什么是SOA

业务系统分解为多个组件，让每个组件都独立提供离散，自治，可复用的服务能力

通过服务的组合和编排来实现上层的业务流程

作用：简化维护,降低整体风险,伸缩灵活

## 什么是微服务

架构设计概念,各服务间隔离（分布式也是隔离）,自治（分布式依赖整体组合）其它特性(单一职责,边界,异步通信,独立部署)是分布式概念的跟严格执行

SOA到微服务架构的演进过程

作用：各服务可独立应用，组合服务也可系统应用(巨石应用[monolith]的简化实现策略-平台思想)

## 使用RPC http技术实现会员与订单系统通讯

# 微服务架构

# SpringCloud

SpringCloud 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蚂蚁课堂SpringBoot课程。

# 服务提供者与消费关系

服务提供者:提供服务被人调用

消费者:调用被人服务

# 服务的注册与发现(Eureka )

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。

## 4.1 服务注册

### 4.1.1创建eurekaserver 项目

### 4.1.2引入maven依赖

|  |
| --- |
| **<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>** |

### 4.3配置application.yml

|  |
| --- |
| server:  port: 8761  eureka:  instance:  hostname: localhost  client:  registerWithEureka: **false**  fetchRegistry: **false**  serviceUrl:  defaultZone: http://${eureka.instance.hostname}:${server.port}/eureka/ |

### 4.4启动EurekaServer

|  |
| --- |
| **@SpringBootApplication**  **@EnableEurekaServer**  **public class EurekaServerApplication {**  **public static void main(String[] args) {**  **SpringApplication.*run*(EurekaServerApplication.class, args);**  **}**  **}** |

1. eureka.client.registerWithEureka=**true**  #是否将自身注册
2. eureka.client.fetchRegistry=**false**    #如果为**true**，启动时报警.

### 4.5打开eureka server 界面的

|  |
| --- |
| [http://localhost:8761](http://localhost:8761/" \t "_blank) ,界面如下：  Paste_Image.png  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。 |

## 4.2 服务提供者

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。

### 4.2.1 创建项目eurekaclient

### 4.2.2 引入maven依赖

|  |
| --- |
| <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> |

### 4.2.3 application.yml配置

|  |
| --- |
| **eureka:**  **client:**  **serviceUrl:**  **defaultZone: http://localhost:8761/eureka/**  **server:**  **port: 8762**  **spring:**  **application:**  **name: service-hi** |

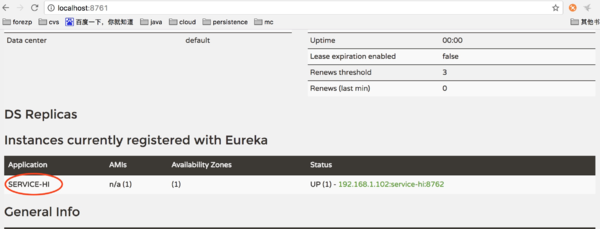
### 4.2.4 发布服务

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;**  **}**  **}** |

### 4.2.5 演示效果

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](http://localhost:8761/" \t "_blank) ，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](http://localhost:8762/hi?name=forezp" \t "_blank) ，你会在浏览器上看到 :

hi forezp,i am from port:8762

# 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。

## 5.1.1什么是ribbon

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

## 5.1.2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：



## 5.1.3 建立一个消费者

### 5.1.3.1 创建一个工程为service-ribbon

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：

|  |
| --- |
| **<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>** |

### 5.1.3.2application.yml配置

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 |

### 5.1.3.3启动类@EnableDiscoveryClient

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。

|  |
| --- |
| @EnableAutoConfiguration  @ComponentScan(basePackages={"com.itmayiedu.controller","com.itmayiedu.service","com.itmayiedu.app"})  @EnableDiscoveryClient  **public** **class** ServiceRibbonApplication {  **public** **static** **void** main(String[] args) {  SpringApplication.*run*(ServiceRibbonApplication.**class**, args);  }  @Bean  @LoadBalanced  RestTemplate restTemplate() {  **return** **new** RestTemplate();  }  } |

### 5.1.3.4 编写一个service

|  |
| --- |
| @Service  **public** **class** HelloService {  @Autowired  RestTemplate restTemplate;  **public** String hiService(String name) {  **return** restTemplate.getForObject("http://SERVICE-HI/hi?name="+name,String.**class**);  }  } |

### 5.1.3.5 编写一个控制器层

|  |
| --- |
| @RestController  **public** **class** HelloControler {  @Autowired  HelloService helloService;  @RequestMapping(value = "/hi")  **public** String hi(@RequestParam String name){  **return** helloService.hiService(name);  }  } |

### 5.1.3.6 演示效果

|  |
| --- |
| 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=](http://service-hi/hi?name=" \t "_blank)“+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。 |

### 5.1.3.7 此时架构



* 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；
* @LoadBalanced注解表明这个restRemplate开启负载均衡的功能。

# [服务消费者（Feign）](http://blog.csdn.net/forezp/article/details/69808079)

## 6.1 什么是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。

简而言之：

* Feign 采用的是基于接口的注解
* Feign 整合了ribbon

## 6.2 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.

### 6.2.1 准备工创建一个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：

|  |
| --- |
| **<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>** |

### 6.2.2 application.yml配置

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** |

### 6.2.3 定义一个feign接口

|  |
| --- |
| **@FeignClient(value = "service-hi")**  **public interface SchedualServiceHi {**  **@RequestMapping(value = "/hi", method = RequestMethod.*GET*)**  **String sayHiFromClientOne(@RequestParam(value = "name") String name);**  **}** |

### 6.2.4一个”/hi”的API接口

|  |
| --- |
| @RestController  **public** **class** HiController {  @Autowired  SchedualServiceHi schedualServiceHi;  @RequestMapping(value = "/hi",method = RequestMethod.***GET***)  **public** String sayHi(@RequestParam String name){  **return** schedualServiceHi.sayHiFromClientOne(name);  }  } |

### 6.2.5启动方式

|  |
| --- |
| **@SpringBootApplication**  **@EnableDiscoveryClient**  **@EnableFeignClients**  **public class SericeFeign {**  **public static void main(String[] args) {**  **SpringApplication.*run*(SericeFeign.class, args);**  **}**  **}** |

### 6.2.6 演示效果

|  |
| --- |
| 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 |

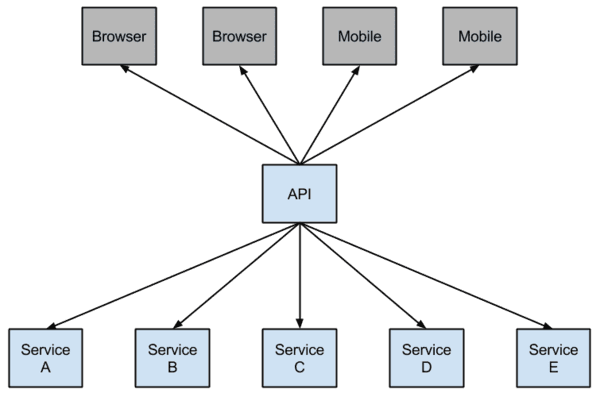
# Hystrix断路器

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。

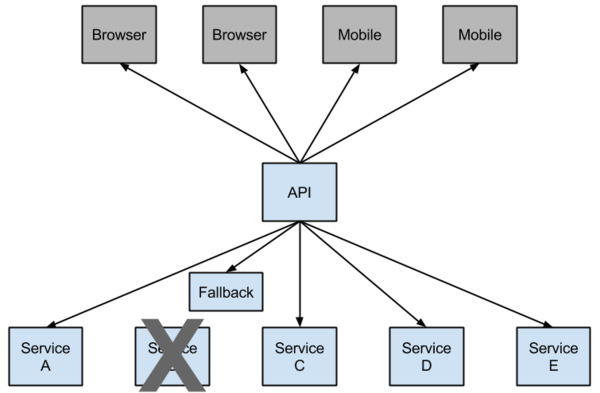
为了解决这个问题，业界提出了断路器模型。

## 7.1 什么是Hystrix

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。

## 7.2 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。

### 7.2.1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> |

### 7.2.2改造service

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!";  }  } |

### 7.2.3演示效果

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。

## 7.3 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 {**  **public String sayHiFromClientOne(String name) {**  **return "sorry " + name;**  **}**  **}** |

开启hystrix

feign:

hystrix:

enabled: true

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

基于service-ribbon 改造，Feign的改造和这一样。

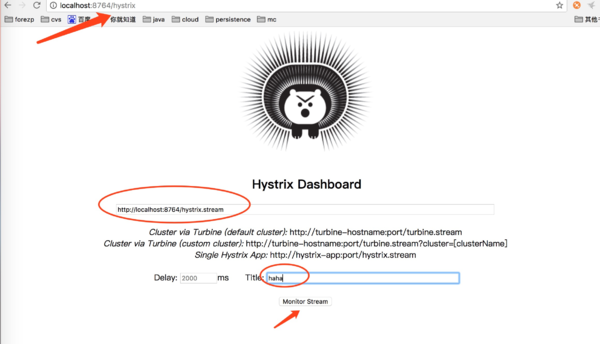
首选在pom.xml引入spring-cloud-starter-hystrix-dashboard的起步依赖：

|  |
| --- |
| **<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：

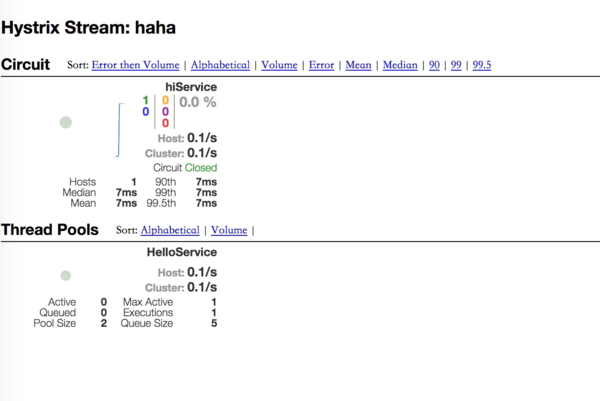
|  |
| --- |
| @EnableAutoConfiguration  @ComponentScan(basePackages = { "com.itmayiedu.controller", "com.itmayiedu.service", "com.itmayiedu.app" })  @EnableHystrixDashboard  @EnableDiscoveryClient  @EnableHystrix  **public** **class** ServiceRibbonApplication {  **public** **static** **void** main(String[] 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：



# 使用Zuul构建API Gateway

## 8.1什么是API Gateway？

在Spring Cloud微服务系统中，一种常见的负载均衡方式是，客户端的请求首先经过负载均衡（zuul、Ngnix），再到达服务网关（zuul集群），然后再到具体的服务。

## 8.2 什么是Zuul？

* Routing in an integral part of a microservice architecture. For example, / may be mapped to your web application, /api/users is mapped to the user service and /api/shop is mapped to the shop service. Zuul is a JVM based router and server side load balancer by Netflix.
* 路由在微服务架构的一个组成部分。 例如，/可以映射到您的Web应用程序，/ api / users映射到用户服务，并且/ api / shop映射到商店服务。 Zuul是Netflix的基于JVM的路由器和服务器端负载均衡器。
* 其功能包括   
  + 验证
  + 见解
  + 压力测试
  + 金丝雀测试
  + 动态路由
  + 服务迁移
  + 减载
  + 安全
  + 静态响应处理
  + 主动/主动流量管理
* Zuul的规则引擎允许规则和过滤器基本上用任何JVM语言编写，内置支持Java和Groovy。

## 8.3 创建service-zuul工程

其pom.xml文件如下：

|  |
| --- |
| **<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-zuul</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>** |

## 8.4 applicaton类

在其入口applicaton类加上注解@EnableZuulProxy，开启zuul的功能：

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

## 8.5 application.yml配置

|  |
| --- |
| eureka:  client:  serviceUrl:  defaultZone: http://localhost:8761/eureka/  server:  port: 8769  spring:  application:  name: service-zuul  zuul:  routes:  api-a:  path: /api-a/\*\*  service-id: service-ribbon  api-b:  path: /api-b/\*\*  service-id: service-feign |

首先指定服务注册中心的地址为[http://localhost:8761/eureka/](http://localhost:8761/eureka/" \t "_blank)，服务的端口为8769，服务名为service-zuul；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；

依次运行这五个工程;打开浏览器访问：[http://localhost:8769/api-a/hi?name=forezp](http://localhost:8769/api-a/hi?name=forezp" \t "_blank) ;浏览器显示：

hi forezp,i am from port:8762

## 8.6服务过滤

zuul不仅只是路由，并且还能过滤，做一些安全验证。继续改造工程；

|  |
| --- |
| @Component  public class MyFilter extends ZuulFilter{  private static Logger log = LoggerFactory.getLogger(MyFilter.class);  @Override  public String filterType() {  return "pre";  }  @Override  public int filterOrder() {  return 0;  }  @Override  public boolean shouldFilter() {  return true;  }  @Override  public Object run() {  RequestContext ctx = RequestContext.getCurrentContext();  HttpServletRequest request = ctx.getRequest();  log.info(String.format("%s >>> %s", request.getMethod(), request.getRequestURL().toString()));  Object accessToken = request.getParameter("token");  if(accessToken == null) {  log.warn("token is empty");  ctx.setSendZuulResponse(false);  ctx.setResponseStatusCode(401);  try {  ctx.getResponse().getWriter().write("token is empty");  }catch (Exception e){}  return null;  }  log.info("ok");  return null;  }  } |

* filterType：返回一个字符串代表过滤器的类型，在zuul中定义了四种不同生命周期的过滤器类型，具体如下：   
  + pre：路由之前
  + routing：路由之时
  + post： 路由之后
  + error：发送错误调用
  + filterOrder：过滤的顺序
  + shouldFilter：这里可以写逻辑判断，是否要过滤，本文true,永远过滤。
  + run：过滤器的具体逻辑。可用很复杂，包括查sql，nosql去判断该请求到底有没有权限访问。

这时访问：[http://localhost:8769/api-a/hi?name=forezp](http://localhost:8769/api-a/hi?name=forezp" \t "_blank) ；网页显示：

token is empty

# 分布式配置中心

[分布式配置中心(Spring Cloud Config)](http://blog.csdn.net/forezp/article/details/70037291)，在分布式系统中，由于服务数量巨多，为了方便服务配置文件统一管理，实时更新，所以需要分布式配置中心组件。在Spring Cloud中，有分布式配置中心组件spring cloud config ，它支持配置服务放在配置服务的内存中（即本地），也支持放在远程Git仓库中。在spring cloud config 组件中，分两个角色，一是config server，二是config client。

## 9.1 构建Config Server

创建一个spring-boot项目，取名为config-server,其pom.xml:

|  |
| --- |
| **<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-config-server</artifactId>**  **</dependency>**  **<dependency>**  **<groupId>org.springframework.boot</groupId>**  **<artifactId>spring-boot-starter-test</artifactId>**  **<scope>test</scope>**  **</dependency>**  **<dependency>**  **<groupId>org.springframework.cloud</groupId>**  **<artifactId>spring-cloud-starter-eureka</artifactId>**  **</dependency>**  **</dependencies>**  **<dependencyManagement>**  **<dependencies>**  **<dependency>**  **<groupId>org.springframework.cloud</groupId>**  **<artifactId>spring-cloud-dependencies</artifactId>**  **<version>Camden.SR6</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>** |

### 9.1.1 Application类加上@EnableConfigServer

|  |
| --- |
| **@SpringBootApplication**  **@EnableConfigServer**  **public class ConfigServerApplication {**  **public static void main(String[] args) {**  **SpringApplication.*run*(ConfigServerApplication.class, args);**  **}**  **}** |

### 9.1.2 application.properties文件配置以下

|  |
| --- |
| **spring.application.name=config-server**  **server.port=8888**  **spring.cloud.config.server.git.uri=https://gitee.com/itmayi/itmayiedu2.git**  **spring.cloud.config.server.git.searchPaths=/itmayi/itmayiedu2.git**  **spring.cloud.config.label=master**  **spring.cloud.config.server.git.username=**  **spring.cloud.config.server.git.password=** |

* spring.cloud.config.server.git.uri：配置git仓库地址
* spring.cloud.config.server.git.searchPaths：配置仓库路径
* spring.cloud.config.label：配置仓库的分支
* spring.cloud.config.server.git.username：访问git仓库的用户名
* spring.cloud.config.server.git.password：访问git仓库的用户密码

如果Git仓库为公开仓库，可以不填写用户名和密码，如果是私有仓库需要填写，本例子是公开仓库，放心使用。

启动程序：访问[http://localhost:8888/foo/dev](http://localhost:8888/foo/dev" \t "_blank)

{"name":"foo","profiles":["dev"],"label":"master",

"version":"792ffc77c03f4b138d28e89b576900ac5e01a44b","state":null,"propertySources":[]}

## 9.2 构建config-client

重新创建一个springboot项目，取名为config-client,其pom文件：

|  |
| --- |
| **<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-config</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>** |

### 9.2.1其配置文件**bootstrap.properties**

|  |
| --- |
| **spring.application.name=config-client**  **spring.cloud.config.label=master**  **spring.cloud.config.profile=dev**  **spring.cloud.config.uri= http://localhost:8888/**  **server.port=8881** |

* spring.cloud.config.label 指明远程仓库的分支
* spring.cloud.config.profile
  + dev开发环境配置文件
  + test测试环境
  + pro正式环境
* spring.cloud.config.uri= [http://localhost:8888/](http://localhost:8888/" \t "_blank) 指明配置服务中心的网址

### 9.2.2写一个API接口“／hi”

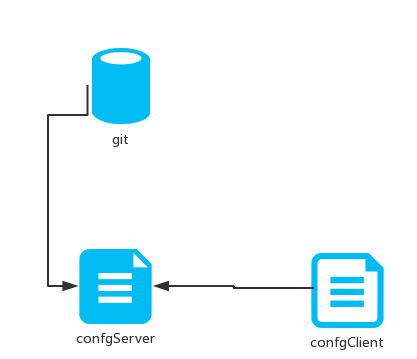
|  |
| --- |
| **@SpringBootApplication**  **@RestController**  **public class ConfigClientApplication {**  **public static void main(String[] args) {**  **SpringApplication.*run*(ConfigClientApplication.class, args);**  **}**  **@Value("${foo}")**  **String foo;**  **@RequestMapping(value = "/hi")**  **public String hi() {**  **return foo;**  **}**  **}** |

### 9.2.3网址访问

：[http://localhost:8881/hi](http://localhost:8881/hi" \t "_blank)，网页显示：

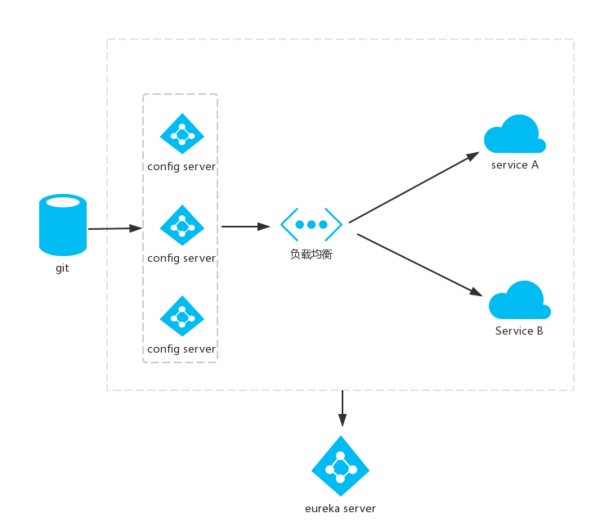
foo version 3

这就说明，config-client从config-server获取了foo的属性，而config-server是从git仓库读取的,如图：



## 9.3 [高可用的分布式配置中心](http://blog.csdn.net/forezp/article/details/70037513)

**[高可用的分布式配置中心(Spring Cloud Config)](http://blog.csdn.net/forezp/article/details/70037513)** 配置中心如何从远程git读取配置文件，当服务实例很多时，都从配置中心读取文件，这时可以考虑将配置中心做成一个微服务，将其集群化，从而达到高可用，架构图如下：



### 9.3.1 准备工作

继续使用上一篇文章的工程，创建一个eureka-server工程，用作服务注册中心。

### 9.3.2 改造config-server

在其pom.xml文件加上EurekaClient的起步依赖spring-cloud-starter-eureka，代码如下: