目录

[1 分布式微服务 2](#_Toc42542299)

[1.1 微服务 2](#_Toc42542300)

[1.2 分布式 2](#_Toc42542301)

[2 服务治理中心Eureka 3](#_Toc42542302)

[2.1 介绍 3](#_Toc42542303)

[2.2 服务端创建流程 3](#_Toc42542304)

[2.3 客户端的创建流程 4](#_Toc42542305)

[2.4 参数进阶 4](#_Toc42542306)

[3 微服务调用Ribbon和Feign 5](#_Toc42542307)

[3.1 介绍 5](#_Toc42542308)

[3.2 Ribbon的使用 5](#_Toc42542309)

[3.3 Feign的使用 6](#_Toc42542310)

[4 断路器Hystrix 7](#_Toc42542311)

[4.1 介绍 7](#_Toc42542312)

[4.2 使用 7](#_Toc42542313)

[5 网关Zuul 8](#_Toc42542314)

[5.1 介绍 8](#_Toc42542315)

[5.2 Zuul微服务创建流程 8](#_Toc42542316)

[6 配置服务器 10](#_Toc42542317)

[6.1 介绍 10](#_Toc42542318)

[6.2 Server服务器配置 10](#_Toc42542319)

[6.3 Client配置 11](#_Toc42542320)

[6.4 参数进阶 12](#_Toc42542321)

[6.5 配置文件优先级 12](#_Toc42542322)

[7 服务调用链追踪器zipkin 12](#_Toc42542323)

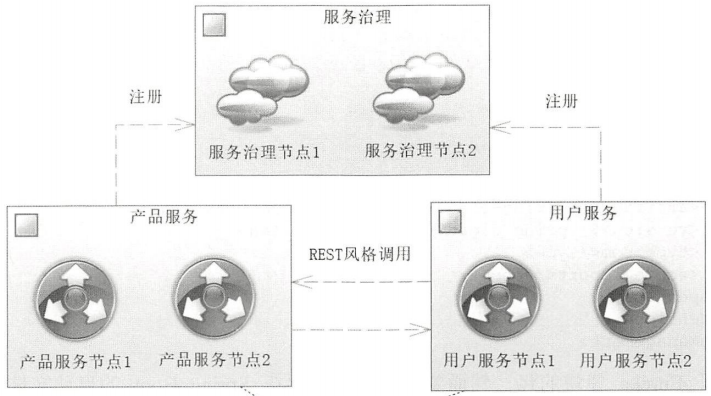
# 分布式微服务

## 微服务

1. 将不同的功能进行拆分，拆分成不同的微服务
2. 微服务之间通过服务治理中心进行交互
3. 服务治理中心每一个微服务都是一个单独的SpringBoot项目

## 分布式

1. 服务治理中心和每个微服务都可以同时部署在多个节点上，提供相同的功能
2. 一方面提高了处理能力，另一方面当某个节点崩溃时，还有其他节点能够维持正常的服务，提高了可用性。

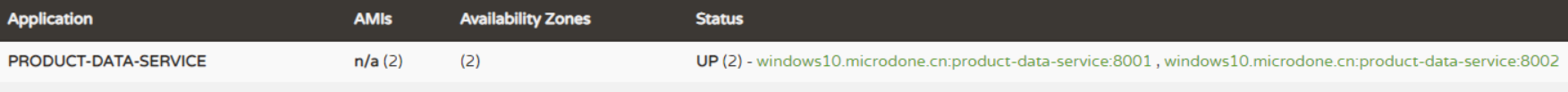


# 服务治理中心Eureka

## 介绍

1. 负责注册和发现各个Spring Boot微服务，并提供监控和管理
2. Eureka将服务分为服务端和客户端两部分，服务端就是服务治理中心，客户端就是各种正常的微服务
3. 服务端本质上也是一种特殊的微服务
4. 服务端和客户端都可以使用分布式的方式部署多个实例提供服务

## 服务端创建流程

1. 创建一个新Spring Boot项目eureka-server
2. 引入依赖包
   1. spring-cloud-starter-netflix-eureka-server
3. 启动类增加注解
   1. @EnableEurekaServer
4. 设置基本的服务端参数
   1. server.port=8761
      1. 服务端端口号，默认8761
   2. spring.application.name=eureka-server
      1. Spring应用名称，用来唯一标识一个微服务
      2. **同一个微服务可以创建多个实例(多个节点)，但他们的应用名称是相同的。**
   3. eureka.instance.hostname=localhost
      1. 当前实例的主机名
      2. 每一个微服务实例都有一个唯一的表示id，会用到这个参数
   4. eureka.client.register-with-eureka=false
      1. 是否注册给服务治理中心，默认为true
      2. 如果当前项目就是服务治理中心，则不需注册，设为false
   5. eureka.client.fetch-registry=false
      1. 是否获取服务器的注册信息，默认为true
      2. 如果当前项目就是服务治理中心，则不需注册，设为false
   6. eureka.client.serviceUrl.defaultZone=http://${eureka.instance.hostname}:${server.port}/eureka/
      1. 服务治理中心公布给客户端进行注册和访问的地址
5. 访问http://127.0.0.1:8761/即可看到注册中心的管理界面
   1. 微服务的应用名称自动转为全大写字母
      1. 

## 客户端的创建流程

1. 创建一个新Spring Boot项目product-data-service
2. 引入依赖包
   1. spring-cloud-starter-netflix-eureka-client
3. 启动类增加注解
   1. @EnableEurekaClient
      1. 使得服务治理中心能够发现该服务，只适用于Eureka服务治理中心
      2. 也可以使用@EnableDiscoveryClient，它能适应所有类型的服务治理中心
4. 设置基本的客户端参数
   1. server.port=8001
      1. 服务端端口号，也可以在启动类内动态设置
   2. spring.application.name=product-data-service
      1. Spring应用名称，用来唯一标识一个微服务
   3. eureka.client.serviceUrl.defaultZone=http://localhost:8761/eureka/
      1. 服务治理中心公布给客户端进行注册和访问的地址
5. 访问http://127.0.0.1:8761/即可看到注册中心的管理界面，里面增加了PRODUCT-DATA-SERVICE微服务

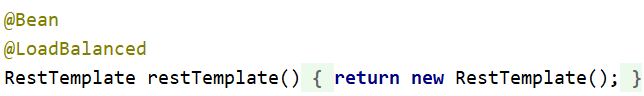
## 参数进阶

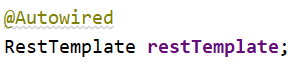
# 微服务调用Ribbon和Feign

## 介绍

1. Ribbon是一个RestTemplate对象，在一个微服务中通过发送REST风格的HTTP请求访问另一个微服务，获取数据
2. Feign则是对Ribbon的封装，使用注解代替Ribbon的代码操作，更加方便使用

## Ribbon的使用

1. 前提
   1. 服务治理中心：http://localhost:8761/eureka/
   2. 数据微服务
      1. spring.application.name=product-data-service
      2. server.port=8001
      3. 提供了http://localhost: 8001/products/接口用于获取数据
   3. 视图微服务
      1. spring.application.name=product-view-service-ribbon
      2. server.port=8010
      3. 提供了http://localhost: 8010/products/接口用于显示数据
   4. 视图微服务的接口实现需要先访问数据微服务提供的接口获取数据
2. 视图微服务的改造
   1. 启动类注册RestTemplate对象实例
      1. 
   2. 在处理http://localhost: 8010/products/请求时
      1. 先通过自动装载获取RestTemplate对象实例



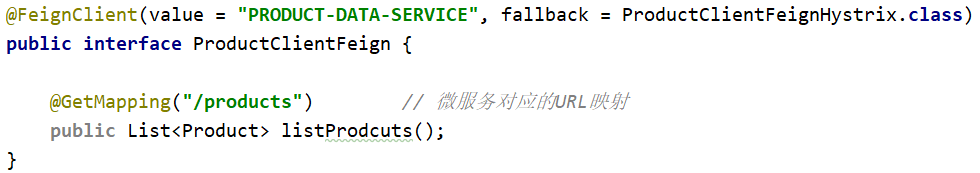
* + 1. 再通过访问数据微服务接口获取数据



* + - 1. 使用数据微服务的应用名称PRODUCT-DATA-SERVICE来代替IP和端口
      2. 第二个参数是返回的数据类型
    1. 最后再进行数据渲染工作

## Feign的使用

1. 前提同上
2. 视图微服务的改造
   1. 引入插件
      1. spring-cloud-starter-openfeign
   2. 启动类增加注解
      1. @EnableFeignClients
   3. 定义Feign接口



* + 1. value指明调用的微服务应用名称PRODUCT-DATA-SERVICE
    2. fallback指明微服务不可用时的反馈处理操作
    3. Feign接口类的方法与被调微服务提供的接口要一致
  1. 在处理http://localhost: 8010/products/请求时
     1. 先通过自动装载获取Feign接口接口对象实例



* + 1. 通过该对象实例获取数据
    2. 进行后续渲染操作

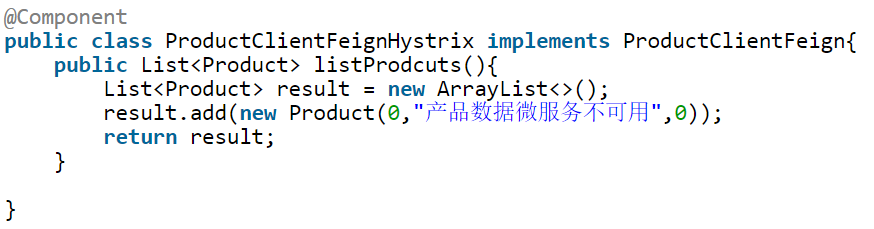
# 断路器Hystrix

## 介绍

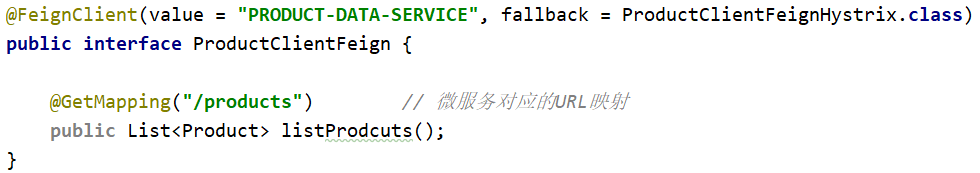
1. 断路器：当被访问的微服务无法使用时，当前服务能够感知到这个问题，并进行反馈处理

## 使用

1. 前提同上
2. 改造调用微服务，也就是视图微服务
   1. 引入插件
      1. spring-cloud-starter-netflix-hystrix
   2. 创建ProductClientFeignHystrix类，用于断路处理



* + 1. 断路器类内方法要和Feign接口类内方法一致，当相应微服务不可用时进行替换
    2. 通过@Component注册到容器中
  1. 在Feign接口类的@FeignClient中设置fallback参数，指明断路处理类



# 网关Zuul

## 介绍

1. 一般情况下，微服务的访问都是通过IP和端口指定微服务实例，当微服务较多时，记录这些对应关系对访问者而言是一个很大的负担
2. 网关Zuul就是通过创建一个特殊的微服务管理这些映射关系，访问者只需要访问Zuul微服务，指定所需要访问的微服务的标识，由Zuul微服务完成请求的映射

## Zuul微服务创建流程

1. 创建一个新Spring Boot项目product-service-zuul
2. 引入依赖包
   1. spring-cloud-starter-netflix-eureka-client
   2. spring-cloud-starter-netflix-zuul
3. 启动类增加注解
   1. @EnableZuulProxy
   2. @EnableEurekaClient
4. 设置基本的服务端参数
   1. server.port=8040
      1. 服务端端口号，默认8080
   2. spring.application.name=product-service-zuul
      1. Spring应用名称
   3. eureka.client.serviceUrl.defaultZone=http://localhost:8761/eureka/
      1. 服务治理中心公布给客户端进行注册和访问的地址
   4. 创建一个映射api-a，将 api-data 映射到数据微服务
      1. zuul.routes.api-a.path=/api-data/\*\*
      2. zuul.routes.api-a.service-id=PRODUCT-DATA-SERVICE
   5. 创建一个映射api-b，将 api-view 映射到视图微服务
      1. zuul.routes.api-b.path=/api-view/\*\*
      2. zuul.routes.api-b.service-id=PRODUCT-VIEW-SERVICE-FEIGN
5. 此时对数据微服务的访问变成：http://localhost:8040/api-data/products

对视图微服务的访问变成：http://localhost:8040/api-view/products

# 配置服务器

## 介绍

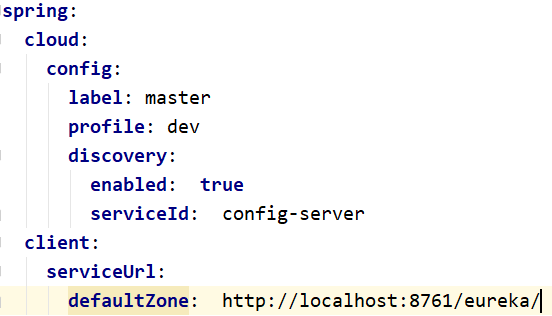
1. 一种特殊的微服务，用于提供统一的配置参数
2. 其他微服务可以通过访问配置服务器获取所需的配置参数，不再需要在各个微服务中单独设置配置参数
3. 配置文件一般存储与github中

## Server服务器配置

1. 创建一个新Spring Boot项目config-server
2. 引入依赖包
   1. spring-cloud-starter-netflix-eureka-client
   2. spring-cloud-config-server
3. 启动类增加注解
   1. @EnableConfigServer
   2. @EnableEurekaClient
4. 设置基本的服务端参数
   1. server.port=8030
   2. spring.application.name=config-server
      1. Spring应用名称
   3. eureka.client.serviceUrl.defaultZone=http://localhost:8761/eureka/
      1. 服务治理中心公布给客户端进行注册和访问的地址
   4. spring.cloud.config.label=master
      1. git端分支，和spring.cloud.config.server.default-label=master参数等同
   5. spring.cloud.config.server.git.uri=https://github.com/how2j/springcloudConfig/
      1. git仓库地址
   6. spring.cloud.config.server.git.search-paths=respo
      1. 仓库下配置文件存放目录

## Client配置

1. 基于已有的视图微服务进行修改
2. 添加新的依赖
   1. spring-cloud-starter-config
3. 新增配置文件bootstrap.yml，配置远程服务器相关参数



* 1. label是git分支
  2. profile指明配置文件后缀
     1. 远程配置文件会被配置服务器转换成web访问接口，可以按以下规则进行访问，application指的是微服务标识名称
        1. /{application}/{profile}[/{label}]
        2. /{application}-{profile}.yml
        3. /{label}/{application}-{profile}.yml
        4. /**{application}-{profile}.properties**
        5. /{label}/{application}-{profile}.properties
  3. discovery:自动发现配置服务器的功能，默认enabled为false
     1. serciceId指明配置服务的名称
     2. 不使用discovery时通过spring.cloud.config.uri指明配置服务器访问IP和端口

1. 使用自动装配获取远程配置文件参数



## 参数进阶

## 配置文件优先级

1. 配置文件加载优先级：
   1. bootstrap.yml > 远程服务器配置文件 > application.yml/application.properties
2. 所以远程服务配置信息必须定义在bootstrap.yml文件中，而不能放在application.properties文件中，因此这时已经过了载远程服务器上的配置文件的阶段了。

# 服务调用链追踪器zipkin

需要先启动zipkin server