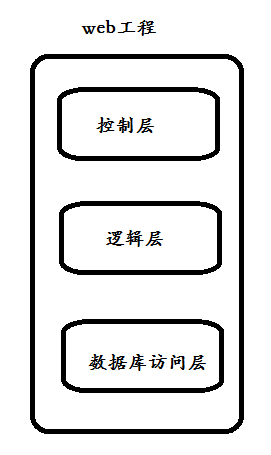
**SpringCloud微服务基础**

# 单点系统架构

## 传统项目架构

传统项目分为三层架构，将业务逻辑层、数据库访问层、控制层放入在一个项目中。

优点:适合于个人或者小团队开发，不适合大团队开发。



## 分布式项目架构

根据业务需求进行拆分成N个子系统，多个子系统相互协作才能完成业务流程子系统之间通讯使用RPC远程通讯技术。

优点:

1.把模块拆分，使用接口通信，降低模块之间的耦合度。

2.把项目拆分成若干个子项目，不同的团队负责不同的子项目。

3.增加功能时只需要再增加一个子项目，调用其它系统的接口就可以。

4.可以灵活的进行分布式部署。

有优点就有缺点，缺点如下：

1.系统之间交互需要使用远程通信，接口开发增加工作量。

2.各个模块有一些通用的业务逻辑无法共用。

为了解决上面分布式架构的缺点，我们引入了soa架构，SOA：Service Oriented Architecture面向服务的架构。也就是把工程拆分成服务层、表现层两个工程。服务层中包含业务逻辑，只需要对外提供服务即可。表现层只需要处理和页面的交互，业务逻辑都是调用服务层的服务来实现。

## 什么是项目集群

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

## RPC远程调用

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，这时候需要通过就可以应用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

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）Hessian，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。

（4） Dubbo是阿里巴巴公司开源的一个高性能优秀的服务框架，使得应用可通过高性能的 RPC 实现服务的输出和输入功能，可以和 Spring框架无缝集成。

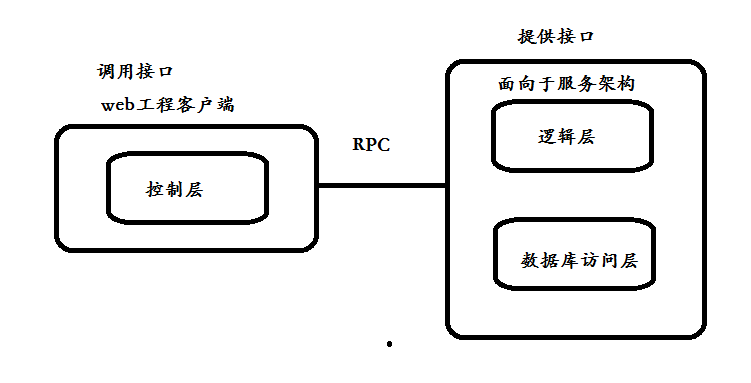
# 面向于服务架构

## 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



# 微服务架构

## 什么是微服务架构

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)

## SOA架构与微服务架构区别

SOA架构主要针对企业级、采用ESB服务（ESB企业服务总线），非常重，需要序列化和反序列化，采用XML格式传输。

微服务架构主要互联网公司，轻量级、小巧，独立运行，基于Http+Rest+JSON格式传输。

ESB也可以说是传统中间件技术与XML、Web服务等技术相互结合的产物。

# 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、负载均衡、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。

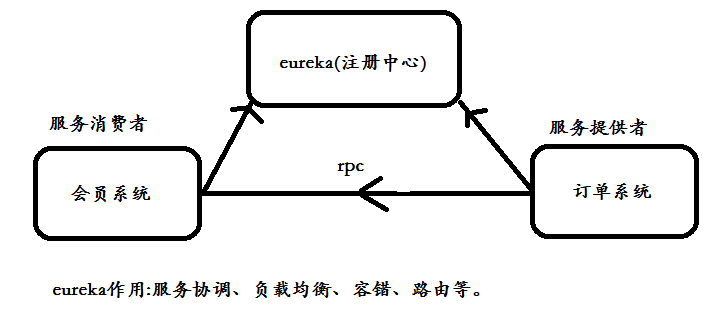
## 服务提供者与消费关系

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。



### 什么是Eureka

官方的介绍在这里[Eureka wiki](https://github.com/Netflix/eureka/wiki/Eureka-at-a-glance)。Eureka是Netflix开源的一个RESTful服务，主要用于服务的注册发现。Eureka由两个组件组成：Eureka服务器和Eureka客户端。Eureka服务器用作服务注册服务器。Eureka客户端是一个java客户端，用来简化与服务器的交互、作为轮询负载均衡器，并提供服务的故障切换支持。Netflix在其生产环境中使用的是另外的客户端，它提供基于流量、资源利用率以及出错状态的加权负载均衡。  
在我看来，Eureka的吸引力来源于以下几点：

**开源**：大家可以对实现一探究竟，甚至修改源码。

**可靠**：经过Netflix多年的生产环境考验，使用应该比较靠谱省心

**功能齐全**：不但提供了完整的注册发现服务，还有Ribbon等可以配合使用的服务。

**基于Java**：对于Java程序员来说，使用起来，心里比较有底。

**spring cloud**可以使用Spring Cloud, 与Eureka进行了很好的集成，使用起来非常方便。

### 实现服务注册

#### 创建EureKaserver 项目

#### 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> |

#### 配置application.yml

|  |
| --- |
| server:  port: 8888  eureka:  instance:  hostname: localhost  client:  registerWithEureka: **false**  fetchRegistry: **false**  serviceUrl:  defaultZone: http://${eureka.instance.hostname}:${server.port}/eureka/ |

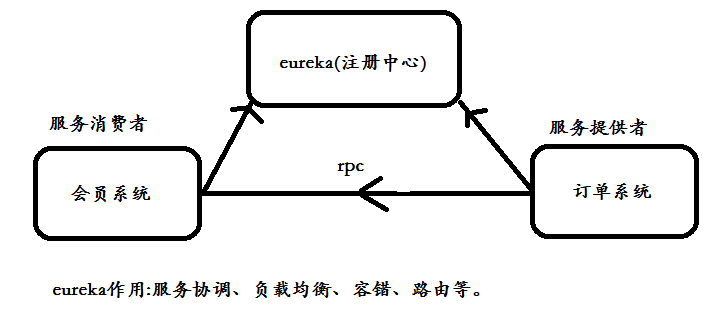
#### 启动EurekaServer

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

#### 打开eureka server 界面的

|  |
| --- |
| [http://localhost:8761](http://localhost:8761/) ,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。 |

## 实现案例订单服务调用会员服务查询用户信息



## 服务提供者

创建一个服务提供者 会员服务工程 (eurekaMember),提供会员查询服务信息

### 创建项目service-member

### 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> |

### application.yml配置

|  |
| --- |
| eureka:  client:  serviceUrl:  defaultZone: http://localhost:8888/eureka/  server:  port: 8762  spring:  application:  name: service-member |

### 服务接口

|  |
| --- |
| @RestController  **public** **class** MemberController {  @RequestMapping("/getUserList")  **public** List<String> getUserList() {  List<String> listUser = **new** ArrayList<String>();  listUser.add("zhangsan");  listUser.add("lisi");  listUser.add("yushengjun");  **return** listUser;  }  } |

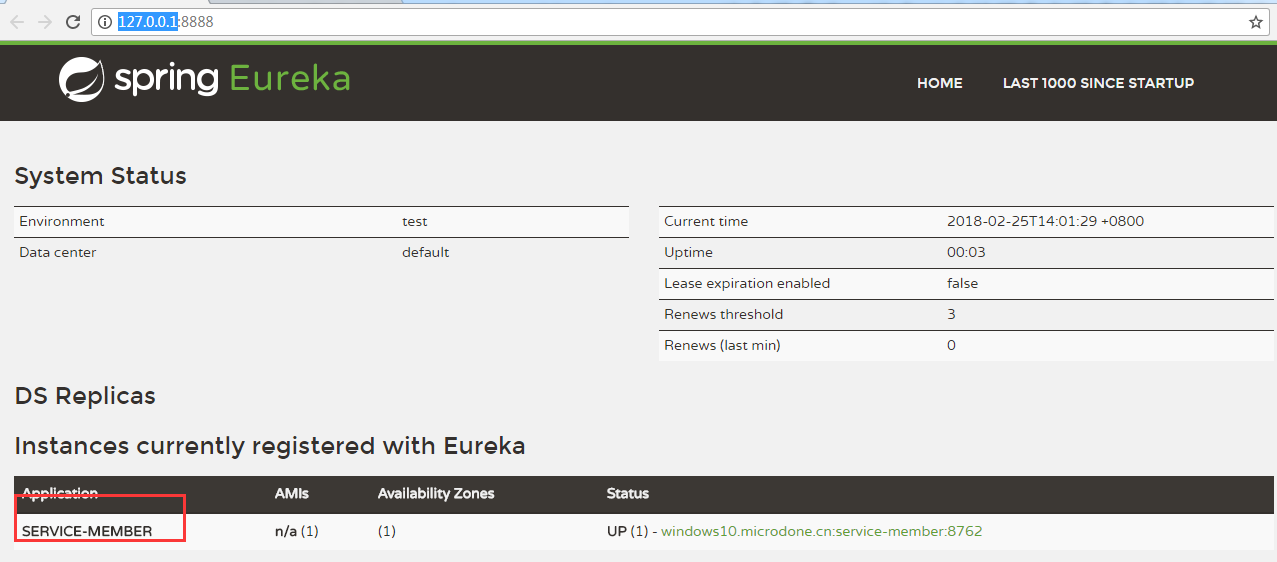
### 发布服务

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

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

### 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127.0.0.1:8888 ，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://127.0.0.1:8762/getUserList ，你会在浏览器上看到 :

["zhangsan","lisi","yushengjun"]

## 服务消费者

### 创建项目sercice-order

### 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.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> |

### application.yml配置

|  |
| --- |
| eureka:  client:  serviceUrl:  defaultZone: http://localhost:8888/eureka/  server:  port: 8764  spring:  application:  name: service-order |

### 编写service,调用service-member

|  |
| --- |
| @SuppressWarnings("unchecked")  @Service  **public** **class** MemberService {  @Autowired  RestTemplate restTemplate;  **public** List<String> getOrderByUserList() {  **return** restTemplate.getForObject("http://service-member/getUserList", List.**class**);  }  } |

### 演示效果

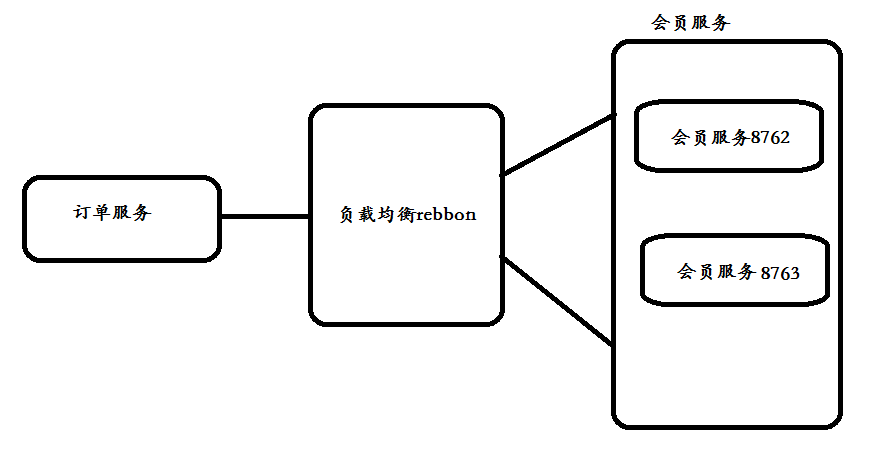
|  |
| --- |
| @RestController  **public** **class** OrderController {  @Autowired  **private** OrderMemberService orderMemberService;    @RequestMapping("/getOrderUserAll")  **public** List<String> getOrderUserAll(){  System.***out***.println("订单服务开始调用会员服务");  **return** orderMemberService.getOrderUserAll();  }  }  @EnableEurekaClient  @SpringBootApplication  **public** **class** AppOrder {  **public** **static** **void** main(String[] args) {  SpringApplication.*run*(AppOrder.**class**, args);  }  @Bean  @LoadBalanced  RestTemplate restTemplate() {  **return** **new** RestTemplate();  }  } |

|  |
| --- |
|  |

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。

## 使用ribbon实现负载均衡

启动两个会员服务工程，端口号分别为8762、8763，订单服务 使用负载均衡策略轮训到会员服务接口。



### 什么是ribbon

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

### 修改会员服务工程代码区分端口项目

|  |
| --- |
| @Value("${server.port}")  **private** String serverPort;  @RequestMapping("/getUserList")  **public** List<String> getUserList() {  List<String> listUser = **new** ArrayList<String>();  listUser.add("zhangsan");  listUser.add("lisi");  listUser.add("yushengjun");  listUser.add("端口号:"+serverPort);  **return** listUser;  } |

### 开启ribbon

@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)

## 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

## 创建service-order-feign工程

## 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.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.cloud</groupId>  <artifactId>spring-cloud-starter-feign</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> |

## application.yml配置

|  |
| --- |
| eureka:  client:  serviceUrl:  defaultZone: http://localhost:8888/eureka/  server:  port: 8765  spring:  application:  name: service-order-feign |

### 编写service,调用service-member

|  |
| --- |
| @FeignClient("service-member")  **public** **interface** MemberFeign {  @RequestMapping("/getUserList")  **public** List<String> getOrderByUserList();  } |

@FeignClient 需要调用服务名称，@RequestMapping服务请求名称

### 演示效果

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

# 路由网关(zuul)

## 什么是网关

Zuul的主要功能是路由转发和过滤器。路由功能是微服务的一部分，比如／api/user转发到到user服务，/api/shop转发到到shop服务。zuul默认和Ribbon结合实现了负载均衡的功能， 类似于nginx转发。

## 搭建SpringCloud网关

## 创建工程service-zuul

## Maven依赖

## 创建工程service-zuul

|  |
| --- |
| <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> |

## application.yml配置

|  |
| --- |
| eureka:  client:  serviceUrl:  defaultZone: http://localhost:8888/eureka/  server:  port: 8769  spring:  application:  name: service-zuul  zuul:  routes:  api-a:  path: /api-member/\*\*  service-id: service-member  api-b:  path: /api-order/\*\*  service-id: service-order |

发送请求http://127.0.0.1:8769/api-member/getMemberAll

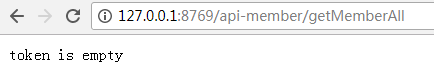
转发到http://127.0.0.1:8762/getMemberAll

开启网关 @EnableZuulProxy

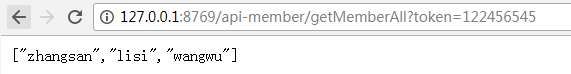
## 服务过滤

|  |
| --- |
| @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;  }  **public** **boolean** shouldFilter() {  **return** **true**;  }  **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**) {  **return** **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**;  }  } |

如果请求参数中没有传入token参数 直接返回报错信息



传入token参数可以正常访问请求



# 断路器(Hystrix)

## 为什么需要 Hystrix?

在微服务架构中，我们将业务拆分成一个个的服务，服务与服务之间可以相互调用（RPC）。为了保证其高可用，单个服务又必须集群部署。由于网络原因或者自身的原因，服务并不能保证服务的100%可用，如果单个服务出现问题，调用这个服务就会出现网络延迟，此时若有大量的网络涌入，会形成任务累计，导致服务瘫痪，甚至导致服务“雪崩”。为了解决这个问题，就出现断路器模型。

Hystrix 是一个帮助解决分布式系统交互时超时处理和容错的类库, 它同样拥有保护系统的能力.什么是服务雪崩

分布式系统中经常会出现某个基础服务不可用造成整个系统不可用的情况, 这种现象被称为服务雪崩效应. 为了应对服务雪崩, 一种常见的做法是手动服务降级. 而Hystrix的出现,给我们提供了另一种选择.

### 服务雪崩应对策略

针对造成服务雪崩的不同原因, 可以使用不同的应对策略:

1. 流量控制
2. 改进缓存模式
3. 服务自动扩容
4. 服务调用者降级服务

**流量控制** 的具体措施包括:

* 网关限流
* 用户交互限流
* 关闭重试

### 服务雪崩解决办法

#### 1.服务雪崩的原因

（1）某几个机器故障：例如机器的硬驱动引起的错误，或者一些特定的机器上出现一些的bug（如，内存中断或者死锁）。

（2）服务器负载发生变化：某些时候服务会因为用户行为造成请求无法及时处理从而导致雪崩，例如阿里的双十一活动，若没有提前增加机器预估流量则会造服务器压力会骤然增大二挂掉。

（3）人为因素：比如代码中的路径在某个时候出现bug

#### 2.解决或缓解服务雪崩的方案

一般情况对于服务依赖的保护主要有3中解决方案：

（1）熔断模式：这种模式主要是参考电路熔断，如果一条线路电压过高，保险丝会熔断，防止火灾。放到我们的系统中，如果某个目标服务调用慢或者有大量超时，此时，熔断该服务的调用，对于后续调用请求，不在继续调用目标服务，直接返回，快速释放资源。如果目标服务情况好转则恢复调用。

（2）隔离模式：这种模式就像对系统请求按类型划分成一个个小岛的一样，当某个小岛被火少光了，不会影响到其他的小岛。例如可以对不同类型的请求使用线程池来资源隔离，每种类型的请求互不影响，如果一种类型的请求线程资源耗尽，则对后续的该类型请求直接返回，不再调用后续资源。这种模式使用场景非常多，例如将一个服务拆开，对于重要的服务使用单独服务器来部署，再或者公司最近推广的多中心。

（3）限流模式：上述的熔断模式和隔离模式都属于出错后的容错处理机制，而限流模式则可以称为预防模式。限流模式主要是提前对各个类型的请求设置最高的QPS阈值，若高于设置的阈值则对该请求直接返回，不再调用后续资源。这种模式不能解决服务依赖的问题，只能解决系统整体资源分配问题，因为没有被限流的请求依然有可能造成雪崩效应。

#### 3.熔断设计

在熔断的设计主要参考了hystrix的做法。其中最重要的是三个模块：熔断请求判断算法、熔断恢复机制、熔断报警

（1）熔断请求判断机制算法：使用无锁循环队列计数，每个熔断器默认维护10个bucket，每1秒一个bucket，每个blucket记录请求的成功、失败、超时、拒绝的状态，默认错误超过50%且10秒内超过20个请求进行中断拦截。

（2）熔断恢复：对于被熔断的请求，每隔5s允许部分请求通过，若请求都是健康的（RT<250ms）则对请求健康恢复。

（3）熔断报警：对于熔断的请求打日志，异常请求超过某些设定则报警

#### 4.隔离设计

隔离的方式一般使用两种

（1）线程池隔离模式：使用一个线程池来存储当前的请求，线程池对请求作处理，设置任务返回处理超时时间，堆积的请求堆积入线程池队列。这种方式需要为每个依赖的服务申请线程池，有一定的资源消耗，好处是可以应对突发流量（流量洪峰来临时，处理不完可将数据存储到线程池队里慢慢处理）

（2）信号量隔离模式：使用一个原子计数器（或信号量）来记录当前有多少个线程在运行，请求来先判断计数器的数值，若超过设置的最大线程个数则丢弃改类型的新请求，若不超过则执行计数操作请求来计数器+1，请求返回计数器-1。这种方式是严格的控制线程且立即返回模式，无法应对突发流量（流量洪峰来临时，处理的线程超过数量，其他的请求会直接返回，不继续去请求依赖的服务）

#### 5 超时机制设计

超时分两种，一种是请求的等待超时，一种是请求运行超时。

等待超时：在任务入队列时设置任务入队列时间，并判断队头的任务入队列时间是否大于超时时间，超过则丢弃任务。

运行超时：直接可使用线程池提供的get方法

## 什么是熔断机制

熔断机制，就是下游服务出现问题后，为保证整个系统正常运行下去，而提供一种降级服务的机制，通过返回缓存数据或者既定数据，避免出现系统整体雪崩效应。在springcloud中，该功能可通过配置的方式加入到项目中。

## Hystrix作用

1.断路器机制

断路器很好理解, 当Hystrix Command请求后端服务失败数量超过一定比例(默认50%), 断路器会切换到开路状态(Open). 这时所有请求会直接失败而不会发送到后端服务. 断路器保持在开路状态一段时间后(默认5秒), 自动切换到半开路状态(HALF-OPEN). 这时会判断下一次请求的返回情况, 如果请求成功, 断路器切回闭路状态(CLOSED), 否则重新切换到开路状态(OPEN). Hystrix的断路器就像我们家庭电路中的保险丝, 一旦后端服务不可用, 断路器会直接切断请求链, 避免发送大量无效请求影响系统吞吐量, 并且断路器有自我检测并恢复的能力.

2.Fallback

Fallback相当于是降级操作. 对于查询操作, 我们可以实现一个fallback方法, 当请求后端服务出现异常的时候, 可以使用fallback方法返回的值. fallback方法的返回值一般是设置的默认值或者来自缓存.

3.资源隔离

在Hystrix中, 主要通过线程池来实现资源隔离. 通常在使用的时候我们会根据调用的远程服务划分出多个线程池. 例如调用产品服务的Command放入A线程池, 调用账户服务的Command放入B线程池. 这样做的主要优点是运行环境被隔离开了. 这样就算调用服务的代码存在bug或者由于其他原因导致自己所在线程池被耗尽时, 不会对系统的其他服务造成影响. 但是带来的代价就是维护多个线程池会对系统带来额外的性能开销. 如果是对性能有严格要求而且确信自己调用服务的客户端代码不会出问题的话, 可以使用Hystrix的信号模式(Semaphores)来隔离资源.

## 服务的降级

### 什么是服务降级

所有的RPC技术里面服务降级是一个最为重要的话题，所谓的降级指的是当服务的提供方不可使用的时候，程序不会出现异常，而会出现本地的操作调

### service-order工程新增Maven依赖

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

### Rest方式使用断路器

|  |
| --- |
| Rest请求方式接口改造  @HystrixCommand(fallbackMethod = "orderError")  **public** List<String> getOrderUserAll() {  **return** restTemplate.getForObject("http://service-member/getMemberAll", List.**class**);  }  **public** List<String> orderError() {  List<String> listUser = **new** ArrayList<String>();  listUser.add("not orderUser list");  **return** listUser;  }  启动方式  @EnableEurekaClient  @EnableHystrix  @SpringBootApplication  **public** **class** OrderApp {  **public** **static** **void** main(String[] args) {  SpringApplication.*run*(OrderApp.**class**, args);  }  @Bean  @LoadBalanced  RestTemplate restTemplate() {  **return** **new** RestTemplate();  }  } |

@HystrixCommand 作用:服务发生错误，回调方法。

@EnableHystrix 启动断路器

### Fegin使用断路器

|  |
| --- |
| 改造service-order-feign工程  @FeignClient(value="service-member",fallback=MemberFeignService.**class**)  **public** **interface** MemberFeign {  @RequestMapping("/getMemberAll")  **public** List<String> getOrderByUserList();  }  @Component  **public** **class** MemberFeignService **implements** MemberFeign {  **public** List<String> getOrderByUserList() {  List<String> listUser = **new** ArrayList<String>();  listUser.add("not orderUser list");  **return** listUser;  }  }  配置文件新增  feign:  hystrix:  enabled: **true** |

|  |
| --- |
| eureka:  client:  serviceUrl:  defaultZone: <http://localhost:8888/eureka/>  #### tomcat最大接线程数  server:  port: 8765  tomcat:  max-threads: 50  spring:  application:  name: service-order-feign  feign:  hystrix:  enabled: **true**  **###超时时间**  hystrix:  command:  default:  execution:  isolation:  thread:  timeoutInMilliseconds: 4000 |

# 分布式配置中心

## 什么是配置中心

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。

## 创建git地址

1. 使用码云创建git地址 https://gitee.com/itmayi
2. config-client-dev.properties ---dev环境
3. 上传配置文件

|  |
| --- |
| userName=yushengjun |

4. <http://localhost:8889/foo/dev> 查询配置中心

## 创建config-server项目

|  |
| --- |
| <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> |

### application.properties 配置文件

|  |
| --- |
| spring.application.name=config-server  server.port=8889  spring.cloud.config.server.git.uri=https://gitee.com/itmayi/cfg2.git  spring.cloud.config.server.git.searchPaths=respo  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

### 启动config-server项目

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

## 创建config-client项目

|  |
| --- |
| <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> |

### 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/> 指明配置服务中心的网址。

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