### 走一边

systemctl stop firewalld.service #停止firewall

systemctl disable firewalld.service #禁止firewall开机启动

firewall-cmd --state #查看默认防火墙状态（关闭后显示notrunning，开启后显示running）

如果是SpringCloud的面试,会问到什么?

* 什么是微服务?
* 微服务之间是怎么通信的?
* SpringCloud和Dubbo有哪些区别?
* SpirngBoot和SpringCloud,请你谈谈对他们的理解?
* 什么是服务熔断?什么是服务降级?
* 微服务的优缺点分别是什么?说下你在项目开发中碰到的坑?
* 你所知道的微服务技术栈有哪些?请列举一二
* eureka和zookeeper都可以提供服务注册与发现的功能,请说说两个的区别?

...

### 微服务是什么?

从技术角度讲:

微服务化的核心就是将传统的一站式应用，根据业务拆分成一个一个的服务，彻底地去耦合,每一个微服务提供单个业务功能的服务，一个服务做一件事，从技术角度看就是一种小而独立的处理过程，类似进程概念，能够自行单独启动或销毁，拥有自己独立的数据库。

### 微服务与微服务架构

#### 微服务

强调的是服务的大小，它关注的是某一个点，是具体解决某一个问题/提供落地对应服务的一个服务应用,狭意的看,可以看作Eclipse里面的一个个微服务工程/或者Module

#### 微服务架构

微服务架构是⼀种架构模式，它提倡将单⼀应⽤程序划分成⼀组⼩的服务，服务之间互相协调、互相配合，为⽤户提供最终价值。每个服务运⾏在其独⽴的进程中，服务与服务间采⽤轻量级的通信机制互相协作（通常是基于HTTP协议的RESTful API）。每个服务都围绕着具体业务进⾏构建，并且能够被独⽴的部署到⽣产环境、类⽣产环境等。另外，应当尽量避免统⼀的、集中式的服务管理机制，对具体的⼀个服务⽽⾔，应根据业务上下⽂，选择合适的语⾔、⼯具对其进⾏构建。

### SpringCloud和SpringBoot是什么关系

SpringBoot专注于快速方便的开发单个个体微服务。

SpringCloud是关注全局的微服务协调整理治理框架，它将SpringBoot开发的一个个单体微服务整合并管理起来，

为各个微服务之间提供，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、分布式会话等等集成服务

SpringBoot可以离开SpringCloud独立使用开发项目，但是SpringCloud离不开SpringBoot，属于依赖的关系.

SpringBoot专注于快速、方便的开发单个微服务个体，SpringCloud关注全局的服务治理框架。

### 我们把SpringCloud VS DUBBO进行一番对比

最大区别：SpringCloud抛弃了Dubbo的RPC通信，采用的是基于HTTP的REST方式。

严格来说，这两种方式各有优劣。虽然从一定程度上来说，后者牺牲了服务调用的性能，但也避免了上面提到的原生RPC带来的问题。而且REST相比RPC更为灵活，服务提供方和调用方的依赖只依靠一纸契约，不存在代码级别的强依赖，这在强调快速演化的微服务环境下，显得更加合适。

品牌机与组装机的区别

很明显，Spring Cloud的功能比DUBBO更加强大，涵盖面更广，而且作为Spring的拳头项目，它也能够与Spring Framework、Spring Boot、Spring Data、Spring Batch等其他Spring项目完美融合，这些对于微服务而言是至关重要的。使用Dubbo构建的微服务架构就像组装电脑，各环节我们的选择自由度很高，但是最终结果很有可能因为一条内存质量不行就点不亮了，总是让人不怎么放心，但是如果你是一名高手，那这些都不是问题；而Spring Cloud就像品牌机，在Spring Source的整合下，做了大量的兼容性测试，保证了机器拥有更高的稳定性，但是如果要在使用非原装组件外的东西，就需要对其基础有足够的了解。

社区支持与更新力度

最为重要的是，DUBBO停止了5年左右的更新，虽然2017.7重启了。对于技术发展的新需求，需要由开发者自行拓展升级（比如当当网弄出了DubboX），这对于很多想要采用微服务架构的中小软件组织，显然是不太合适的，中小公司没有这么强大的技术能力去修改Dubbo源码+周边的一整套解决方案，并不是每一个公司都有阿里的大牛+真实的线上生产环境测试过。

### 微服务技术栈有哪些?

微服务条目落地技术备注服务开发Springboot、Spring、SpringMVC服务配置与管理Netflix公司的Archaius、阿里的Diamond等服务注册与发现Eureka、Consul、Zookeeper等服务调用Rest、RPC、gRPC服务熔断器Hystrix、Envoy等负载均衡Ribbon、Nginx等服务接口调用(客户端调用服务的简化工具)Feign等消息队列Kafka、RabbitMQ、ActiveMQ等服务配置中心管理SpringCloudConfig、Chef等服务路由（API网关）Zuul等服务监控Zabbix、Nagios、Metrics、Spectator等全链路追踪Zipkin，Brave、Dapper等服务部署Docker、OpenStack、Kubernetes等数据流操作开发包SpringCloud Stream（封装与Redis,Rabbit、Kafka等发送接收消息）事件消息总线Spring Cloud Bus......

## 创建项目

我们来创建一个有pom父工程,一个公共api,一个生产者,一个消费者的project;

/microservice-springcloud #pom父工程

/microservice-springcloud-api #封装的整体Entity/接口/公共配置等

/microservice-springcloud-dept-provider-8001 #微服务落地的服务提供者

/microservice-springcloud-dept-consumer-9001 #微服务调用的客户端使用

### 创建/microservice-springcloud父工程

父工程就一个pom文件,加载一些依赖~

|  |
| --- |
| <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.lk.springcloud</groupId>  <artifactId>microservice-springcloud</artifactId>  <version>0.0.1-SNAPSHOT</version>  <packaging>pom</packaging>  <properties>  <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>  <maven.compiler.source>1.8</maven.compiler.source>  <maven.compiler.target>1.8</maven.compiler.target>  <junit.version>4.12</junit.version>  <log4j.version>1.2.17</log4j.version>  <lombok.version>1.16.18</lombok.version>  </properties>  <dependencyManagement>  <dependencies>  <dependency>  <groupId>org.springframework.cloud</groupId>  <artifactId>spring-cloud-dependencies</artifactId>  <version>Dalston.SR1</version>  <type>pom</type>  <scope>import</scope>  </dependency>  <dependency>  <groupId>org.springframework.boot</groupId>  <artifactId>spring-boot-dependencies</artifactId>  <version>1.5.9.RELEASE</version>  <type>pom</type>  <scope>import</scope>  </dependency>  <dependency>  <groupId>mysql</groupId>  <artifactId>mysql-connector-java</artifactId>  <version>5.1.38</version>  </dependency>  <dependency>  <groupId>com.alibaba</groupId>  <artifactId>druid</artifactId>  <version>1.0.31</version>  </dependency>  <dependency>  <groupId>org.mybatis.spring.boot</groupId>  <artifactId>mybatis-spring-boot-starter</artifactId>  <version>1.3.0</version>  </dependency>  <dependency>  <groupId>ch.qos.logback</groupId>  <artifactId>logback-core</artifactId>  <version>1.2.3</version>  </dependency>  <dependency>  <groupId>junit</groupId>  <artifactId>junit</artifactId>  <version>${junit.version}</version>  <scope>test</scope>  </dependency>  <dependency>  <groupId>log4j</groupId>  <artifactId>log4j</artifactId>  <version>${log4j.version}</version>  </dependency>  </dependencies>  </dependencyManagement>  </project> |

### 创建/microservice-springcloud-api 工程

/microservice-springcloud-api #封装的整体Entity/接口/公共配置等

/microservice-springcloud-dept-provider-8001 #微服务落地的服务提供者

/microservice-springcloud-dept-consumer-9001 #微服务调用的客户端使用

注意,因为是公用的,所以在写完后要

mvn clean install后给其它模块引用，达到通用目的。

也即需要用到部门实体的话，不用每个工程都定义一份，直接引用本模块即可。

pom.xml

|  |
| --- |
| <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>microservice-springcloud-api</groupId>  <artifactId>microservice-springcloud-api</artifactId>  <version>0.0.1-SNAPSHOT</version>  <dependencies>  <dependency>  <groupId>org.projectlombok</groupId>  <artifactId>lombok</artifactId>  <version>1.16.18</version>  <scope>provided</scope>  </dependency>  </dependencies>  </project> |

因为要写实体类,所以引入了lombok

|  |
| --- |
| package com.lk.springcloud.entities;  import java.io.Serializable;  import lombok.AllArgsConstructor;  import lombok.Data;  import lombok.experimental.Accessors;  @SuppressWarnings("serial")  @Accessors(chain = true)  @AllArgsConstructor  @Data  public class Dept implements Serializable {  private Long deptno;  private String dname;  private String db\_source;  public Dept() {  super();  }  } |

### 创建/microservice-springcloud-dept-provider-8001服务生产者

/microservice-springcloud-api #封装的整体Entity/接口/公共配置等

/microservice-springcloud-dept-provider-8001 #微服务落地的服务提供者

/microservice-springcloud-dept-consumer-9001 #微服务调用的客户端使用

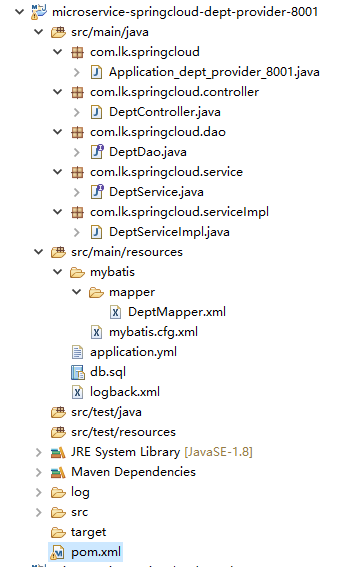
服务生产者继承父类并且依赖公共api类,

这里用jetty作为web容器

所以pom.xml

|  |
| --- |
| <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>microservice-springcloud-dept-provider-8001</groupId>  <artifactId>microservice-springcloud-dept-provider-8001</artifactId>  <version>0.0.1-SNAPSHOT</version>  <parent>  <groupId>com.lk.springcloud</groupId>  <artifactId>microservice-springcloud</artifactId>  <version>0.0.1-SNAPSHOT</version>  </parent>  <dependencies>  <dependency>  <groupId>junit</groupId>  <artifactId>junit</artifactId>  </dependency>  <dependency>  <groupId>mysql</groupId>  <artifactId>mysql-connector-java</artifactId>  </dependency>  <dependency>  <groupId>com.alibaba</groupId>  <artifactId>druid</artifactId>  </dependency>  <dependency>  <groupId>ch.qos.logback</groupId>  <artifactId>logback-core</artifactId>  </dependency>  <dependency>  <groupId>org.mybatis.spring.boot</groupId>  <artifactId>mybatis-spring-boot-starter</artifactId>  </dependency>  <dependency>  <groupId>org.springframework.boot</groupId>  <artifactId>spring-boot-starter-jetty</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>  </dependency>  <dependency>  <groupId>org.springframework</groupId>  <artifactId>springloaded</artifactId>  </dependency>  <dependency>  <groupId>org.springframework.boot</groupId>  <artifactId>spring-boot-devtools</artifactId>  </dependency>  <dependency>  <groupId>microservice-springcloud-api</groupId>  <artifactId>microservice-springcloud-api</artifactId>  <version>0.0.1-SNAPSHOT</version>  </dependency>  </dependencies>  </project> |

因为要连接数据库自查



这里主要看看application.yml和MainApplication.java启动类

application.yml

|  |
| --- |
| server:  port: 8001  mybatis:  config-location: classpath:mybatis/mybatis.cfg.xml # mybatis配置文件所在路径  type-aliases-package: com.lk.springcloud.entities # 所有Entity别名类所在包  mapper-locations:  - classpath:mybatis/mapper/\*\*/\*.xml # mapper映射文件  spring:  application:  name: MICROSERVICECLOUD-DEPT  datasource:  type: com.alibaba.druid.pool.DruidDataSource # 当前数据源操作类型  driver-class-name: org.gjt.mm.mysql.Driver # mysql驱动包  url: jdbc:mysql://192.168.106.34:3306/cloudDB01 # 数据库名称  username: root  password: root  dbcp2:  min-idle: 5 # 数据库连接池的最小维持连接数  initial-size: 5 # 初始化连接数  max-total: 5 # 最大连接数  max-wait-millis: 200 # 等待连接获取的最大超时时间 |

启动类

|  |
| --- |
| package com.lk.springcloud;  import org.springframework.boot.SpringApplication;  import org.springframework.boot.autoconfigure.SpringBootApplication;  @SpringBootApplication  public class Application\_dept\_provider\_8001 {  public static void main(String[] args) throws Exception {  SpringApplication.run(Application\_dept\_provider\_8001.class, args);  }  } |

由于是自己自足的环境,所以这里的生产者可以直连数据库做操作.

这里也是为了为后面的消费者调用服务做准备

### 创建/microservice-springcloud-dept-consumer-9001服务消费者

/microservice-springcloud-api #封装的整体Entity/接口/公共配置等

/microservice-springcloud-dept-provider-8001 #微服务落地的服务提供者

/microservice-springcloud-dept-consumer-9001 #微服务调用的客户端使用

pom.xml

|  |
| --- |
| <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>  <parent>  <groupId>com.lk.springcloud</groupId>  <artifactId>microservice-springcloud</artifactId>  <version>0.0.1-SNAPSHOT</version>  </parent>  <artifactId>microservice-springcloud-dept-consumer-9001</artifactId>  <dependencies>  <dependency>  <groupId>microservice-springcloud-api</groupId>  <artifactId>microservice-springcloud-api</artifactId>  <version>0.0.1-SNAPSHOT</version>  </dependency>  <dependency>  <groupId>org.mybatis.spring.boot</groupId>  <artifactId>mybatis-spring-boot-starter</artifactId>  </dependency>  <dependency>  <groupId>org.springframework.boot</groupId>  <artifactId>spring-boot-starter-web</artifactId>  </dependency>  <dependency>  <groupId>org.springframework</groupId>  <artifactId>springloaded</artifactId>  </dependency>  <dependency>  <groupId>org.springframework.boot</groupId>  <artifactId>spring-boot-devtools</artifactId>  </dependency>  </dependencies>  </project> |

启动类

由于启动会报一个数据库连接的错误,这里做了一个排除的配置

|  |
| --- |
| package com.lk.springcloud;  import org.springframework.boot.SpringApplication;  import org.springframework.boot.autoconfigure.EnableAutoConfiguration;  import org.springframework.boot.autoconfigure.SpringBootApplication;  import org.springframework.boot.autoconfigure.jdbc.DataSourceAutoConfiguration;  @SpringBootApplication  @EnableAutoConfiguration(exclude={DataSourceAutoConfiguration.class})  public class Application\_dept\_consumer\_9001 {    public static void main(String[] args) {  SpringApplication.run(Application\_dept\_consumer\_9001.class, args);  }  } |

在controller里面用到了

RestTemplate模板类,可以调用远程的服务

这个RestTemplate是作为配置的bean传进来的

configureBean.java

|  |
| --- |
| package com.lk.springcloud.conf;  import org.springframework.context.annotation.Bean;  import org.springframework.context.annotation.Configuration;  import org.springframework.web.client.RestTemplate;  @Configuration  public class configureBean {    @Bean  public RestTemplate restTemplate() {  return new RestTemplate();  }  } |

Dept\_consumer\_controller

调用远程服务的原理是:

这里调用@RequestMapping("/dept/consumer")

@RequestMapping(value = "/add")

然后转发给了private static final String REST\_URL\_PREFIX = "http://localhost:8001/dept/provider"

以及对于的/add方法

其中这个REST\_URL\_PREFIX就是服务提供者的工程名,在yml里面可以配置,很重要~

|  |
| --- |
| package com.lk.springcloud.controller;  import java.util.List;  import javax.annotation.Resource;  import org.springframework.web.bind.annotation.PathVariable;  import org.springframework.web.bind.annotation.RequestMapping;  import org.springframework.web.bind.annotation.RestController;  import org.springframework.web.client.RestTemplate;  import com.lk.springcloud.entities.Dept;  @RestController  @RequestMapping("/dept/consumer")  public class Dept\_consumer\_controller {  /\*\*  \* RestTemplate是什么?  \*  \* RestTemplate提供了多种便捷访问远程Http服务的方法， 是一种简单便捷的访问restful服务模板类，是Spring提供的用于访问Rest服务的客户端模板工具集  \*  \* 使用  \* 使用restTemplate访问restful接口非常的简单粗暴无脑。  \* (url, requestMap, ResponseBean.class)这三个参数分别代表  \* REST请求地址、请求参数、HTTP响应转换被转换成的对象类型。  \*/      private static final String REST\_URL\_PREFIX = "http://localhost:8001/dept/provider";  // private static final String REST\_URL\_PREFIX = "http://MICROSERVICECLOUD-DEPT";  @Resource  private RestTemplate restTemplate;  @RequestMapping(value = "/add")  public boolean add(Dept dept) {  return restTemplate.postForObject(REST\_URL\_PREFIX + "/add", dept, Boolean.class);  }  @RequestMapping(value = "/get/{id}")  public Dept get(@PathVariable("id") Long id) {  return restTemplate.getForObject(REST\_URL\_PREFIX + "/get/" + id, Dept.class);  }  @SuppressWarnings("unchecked")  @RequestMapping(value = "/list")  public List<Dept> list() {  return restTemplate.getForObject(REST\_URL\_PREFIX + "/list", List.class);  }  } |

### 搭建/microservice-springcloud-eurekaserver-7001

搭建eureka server服务注册中心,类似于dubbo里面用到的广播或者zookeeper等

我们搭建起这个web服务先

pom.xml

|  |
| --- |
| <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>  <parent>  <groupId>com.lk.springcloud</groupId>  <artifactId>microservice-springcloud</artifactId>  <version>0.0.1-SNAPSHOT</version>  </parent>  <artifactId>microservice-springcloud-eureka-7001</artifactId>  <dependencies>  <!--eureka-server服务端 -->  <dependency>  <groupId>org.springframework.cloud</groupId>  <artifactId>spring-cloud-starter-eureka-server</artifactId>  </dependency>  <!-- 修改后立即生效，热部署 -->  <dependency>  <groupId>org.springframework</groupId>  <artifactId>springloaded</artifactId>  </dependency>  <dependency>  <groupId>org.springframework.boot</groupId>  <artifactId>spring-boot-devtools</artifactId>  </dependency>  </dependencies>  </project> |

application.yml

|  |
| --- |
| server:  port: 7001    eureka:  instance:  hostname: localhost #eureka服务端的实例名称  #hostname: eureka7001.com  client:  register-with-eureka: false #false表示不向注册中心注册自己。  fetch-registry: false #false表示自己端就是注册中心，我的职责就是维护服务实例，并不需要去检索服务  service-url:  defaultZone: http://${eureka.instance.hostname}:${server.port}/eureka/  #设置与Eureka Server交互的地址查询服务和注册服务都需要依赖这个地址（单机）。  #defaultZone: http://eureka7002.com:7002/eureka,http://eureka7003.com:7003/eureka |

启动类

注意这里使用一个@EnableEurekaServer表明这个服务是开启eureka服务注册

|  |
| --- |
| package com.lk.springcloud;  import org.springframework.boot.SpringApplication;  import org.springframework.boot.autoconfigure.SpringBootApplication;  import org.springframework.cloud.netflix.eureka.server.EnableEurekaServer;  @SpringBootApplication  @EnableEurekaServer  public class Application\_eurekaserver\_7001 {  public static void main(String[] args) {  SpringApplication.run(Application\_eurekaserver\_7001.class, args);  }  } |

如果单单是启动web服务的话,这里就足够了,但是只有一个页面,没有服务生产者注册进来~

#### eureka是什么

Eureka是Netflix的一个子模块，也是核心模块之一。Eureka是一个基于REST的服务，用于定位服务，以实现云端中间层服务发现和故障转移。

服务注册与发现对于微服务架构来说是非常重要的，有了服务发现与注册，只需要使用服务的标识符，就可以访问到服务，而不需要修改服务调用的配置文件了。功能类似于dubbo的注册中心，比如Zookeeper。

#### eureka的基本架构

Spring Cloud 封装了 Netflix 公司开发的 Eureka 模块来实现服务注册和发现(请对比Zookeeper)。

Eureka 采用了 C-S 的设计架构。Eureka Server 作为服务注册功能的服务器，它是服务注册中心。

而系统中的其他微服务，使用 Eureka 的客户端连接到 Eureka Server并维持心跳连接。这样系统的维护人员就可以通过 Eureka Server 来监控系统中各个微服务是否正常运行。SpringCloud 的一些其他模块（比如Zuul）就可以通过 Eureka Server 来发现系统中的其他微服务，并执行相关的逻辑。

请注意和Dubbo的架构对比

Eureka包含两个组件：Eureka Server和Eureka Client

Eureka Server提供服务注册服务

各个节点启动后，会在EurekaServer中进行注册，这样EurekaServer中的服务注册表中将会存储所有可用服务节点的信息，服务节点的信息可以在界面中直观的看到

EurekaClient是一个Java客户端，用于简化Eureka Server的交互，客户端同时也具备一个内置的、使用轮询(round-robin)负载算法的负载均衡器。在应用启动后，将会向Eureka Server发送心跳(默认周期为30秒)。如果Eureka Server在多个心跳周期内没有接收到某个节点的心跳，EurekaServer将会从服务注册表中把这个服务节点移除（默认90秒）

#### eureka的三大角色

Eureka Server 提供服务注册和发现

Service Provider服务提供方将自身服务注册到Eureka，从而使服务消费方能够找到

Service Consumer服务消费方从Eureka获取注册服务列表，从而能够消费服务

### 将已有的部门微服务/microservice-springcloud-dept-provider-8001注册进eureka服务中心

需要先启动eureka服务,再启动其他服务

在/microservice-springcloud-dept-provider-8001中添加

添加pom.xml

这是eureka客户端

|  |
| --- |
| <!-- 将微服务provider侧注册进eureka -->  <dependency>  <groupId>org.springframework.cloud</groupId>  <artifactId>spring-cloud-starter-eureka</artifactId>  </dependency>  <dependency>  <groupId>org.springframework.cloud</groupId>  <artifactId>spring-cloud-starter-config</artifactId>  </dependency> |

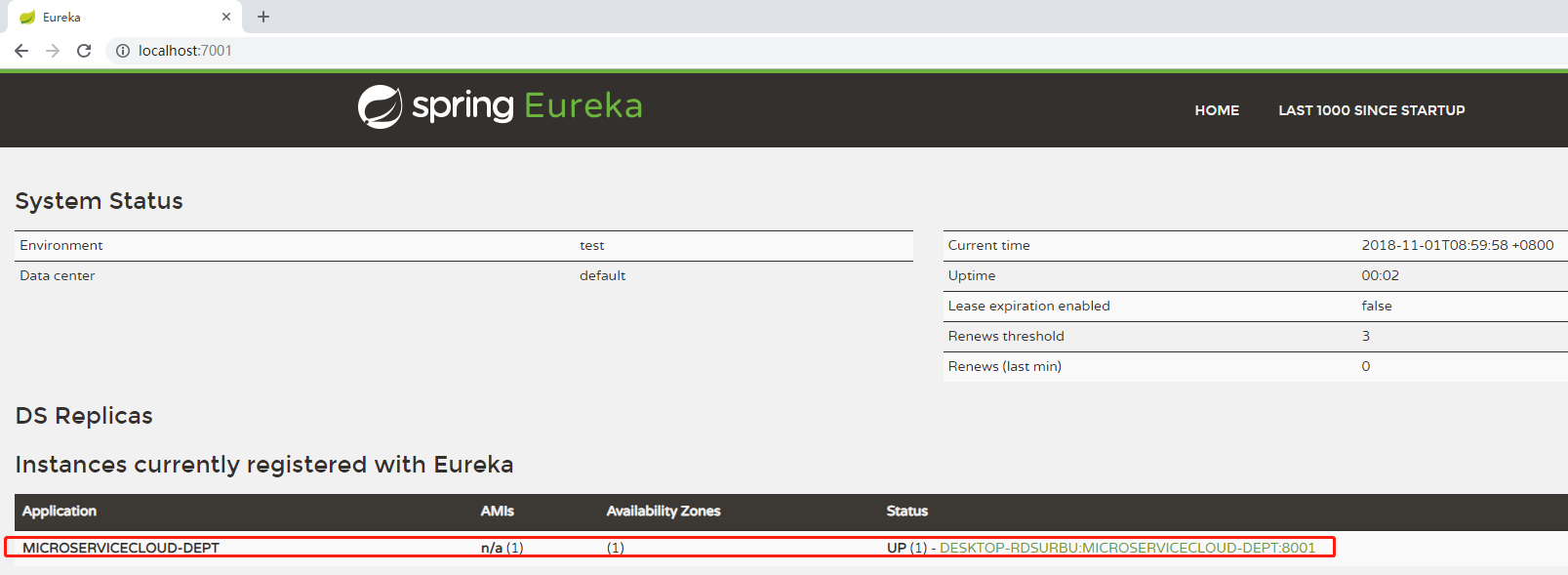
yml添加

将本服务注册到7001注册中心

|  |
| --- |
| eureka:    client: #客户端注册进eureka服务列表内      service-url:        defaultZone: http://localhost:7001/eureka |

启动类要添加注解

@EnableEurekaClient



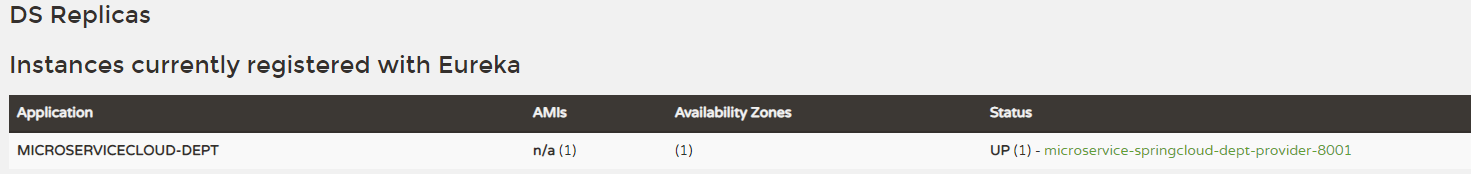
#### actuator与注册微服务信息完善

##### 主机名称:服务名称修改

其中可以配置在status找那个的注册的服务的名称:

在服务端的yml中

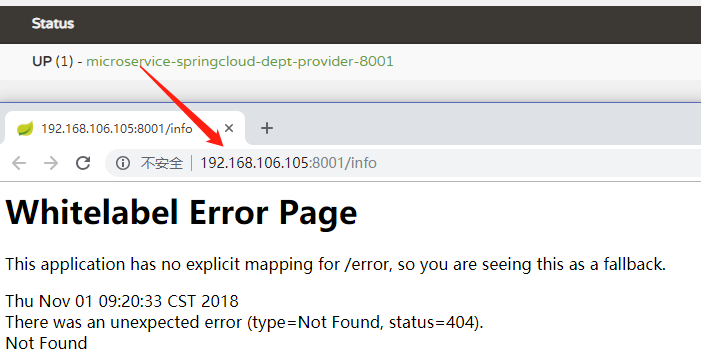
|  |
| --- |
| eureka:  client: #客户端注册进eureka服务列表内  service-url:  defaultZone: http://localhost:7001/eureka  instance:  instance-id: microservice-springcloud-dept-provider-8001 |



##### 访问信息有IP信息提示

|  |
| --- |
| eureka:  client: #客户端注册进eureka服务列表内  service-url:  defaultZone: http://localhost:7001/eureka  instance:  instance-id: microservice-springcloud-dept-provider-8001  prefer-ip-address: true #访问路径可以显示IP地址 |

点击跳转可以在url显示ip地址



##### 微服务info内容详细信息

当前问题:点击超链接会报error信息

修改/microservice-springcloud-dept-provider-8001的pom.xml

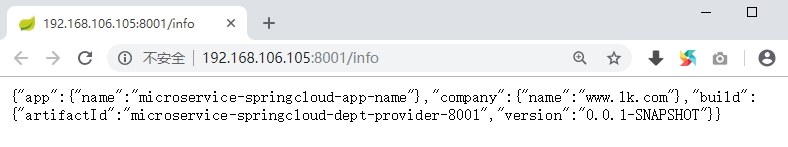
|  |
| --- |
| <dependency>  <groupId>org.springframework.boot</groupId>  <artifactId>spring-boot-starter-actuator</artifactId>  </dependency> |

修改/microservice-springcloud的pom.xml

|  |
| --- |
| <build>  <finalName>microservicecloud</finalName>  <resources>  <resource>  <directory>src/main/resources</directory>  <filtering>true</filtering>  </resource>  </resources>  <plugins>  <plugin>  <groupId>org.apache.maven.plugins</groupId>  <artifactId>maven-resources-plugin</artifactId>  <configuration>  <delimiters>  <delimit>$</delimit>  </delimiters>  </configuration>  </plugin>  </plugins>  </build> |

修改/microservice-springcloud-dept-provider-8001的yml

|  |
| --- |
| info:  app.name: microservice-springcloud-app-name  company.name: www.lk.com  build.artifactId: $project.artifactId$  build.version: $project.version$ |



#### eureka自我保护

什么是自我保护模式？

默认情况下，如果EurekaServer在一定时间内没有接收到某个微服务实例的心跳，EurekaServer将会注销该实例（默认90秒）。但是当网络分区故障发生时，微服务与EurekaServer之间无法正常通信，以上行为可能变得非常危险了——因为微服务本身其实是健康的，此时本不应该注销这个微服务。Eureka通过“自我保护模式”来解决这个问题——当EurekaServer节点在短时间内丢失过多客户端时（可能发生了网络分区故障），那么这个节点就会进入自我保护模式。一旦进入该模式，EurekaServer就会保护服务注册表中的信息，不再删除服务注册表中的数据（也就是不会注销任何微服务）。当网络故障恢复后，该Eureka Server节点会自动退出自我保护模式。

在自我保护模式中，Eureka Server会保护服务注册表中的信息，不再注销任何服务实例。当它收到的心跳数重新恢复到阈值以上时，该Eureka Server节点就会自动退出自我保护模式。它的设计哲学就是宁可保留错误的服务注册信息，也不盲目注销任何可能健康的服务实例。一句话讲解：好死不如赖活着

综上，自我保护模式是一种应对网络异常的安全保护措施。它的架构哲学是宁可同时保留所有微服务（健康的微服务和不健康的微服务都会保留），也不盲目注销任何健康的微服务。使用自我保护模式，可以让Eureka集群更加的健壮、稳定。

在Spring Cloud中，可以使用eureka.server.enable-self-preservation = false 禁用自我保护模式。

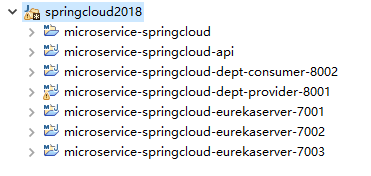
一句话：某时刻某一个微服务不可用了，eureka不会立刻清理，依旧会对该微服务的信息进行保存

### 创建eureka server的集群搭建

#### 创建集群项目

/microservice-springcloud-eurekaserver-7002

/microservice-springcloud-eurekaserver-7003



#### 修改win的hosts文件

找到C:\Windows\System32\drivers\etc路径下的hosts文件

127.0.0.1 eureka7001.com

127.0.0.1 eureka7002.com

127.0.0.1 eureka7003.com

#### 修改3台eureka服务器的yml配置

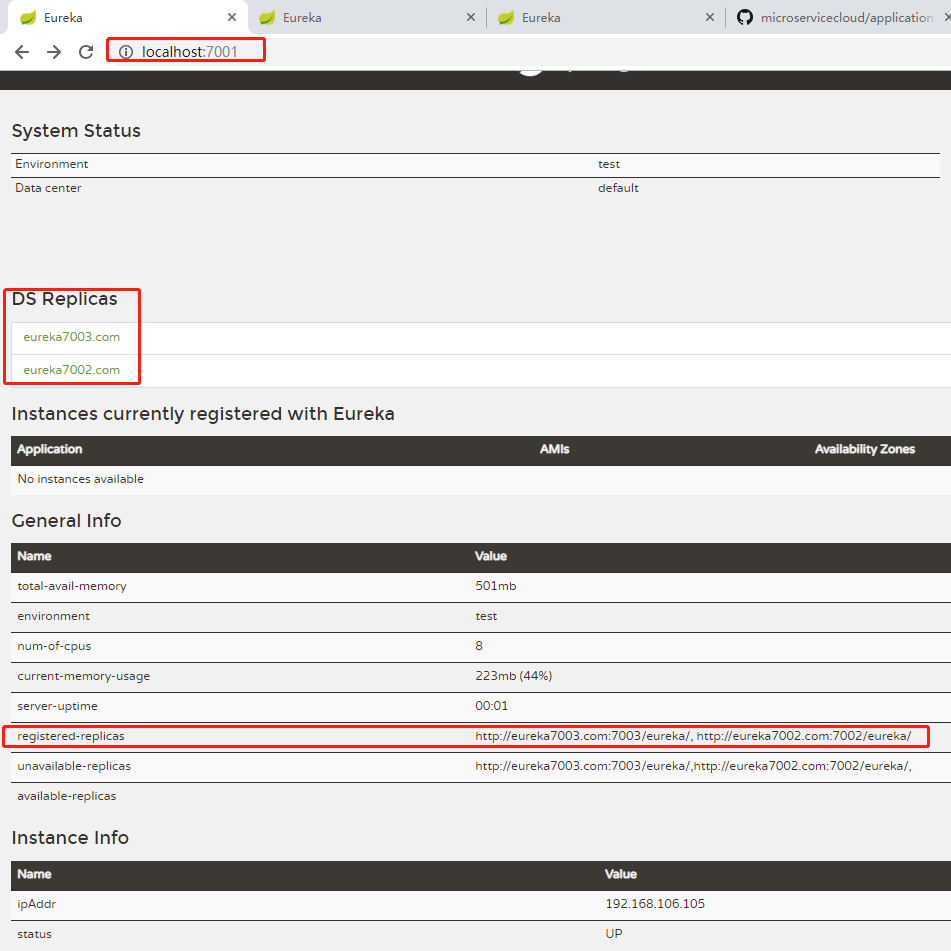
7001.yml

|  |
| --- |
| server:  port: 7001    eureka:  instance:  #hostname: localhost #eureka服务端的实例名称  hostname: eureka7001.com  client:  register-with-eureka: false #false表示不向注册中心注册自己。  fetch-registry: false #false表示自己端就是注册中心，我的职责就是维护服务实例，并不需要去检索服务  service-url:  #defaultZone: http://${eureka.instance.hostname}:${server.port}/eureka/  #设置与Eureka Server交互的地址查询服务和注册服务都需要依赖这个地址（单机）。  defaultZone: http://eureka7002.com:7002/eureka,http://eureka7003.com:7003/eureka |

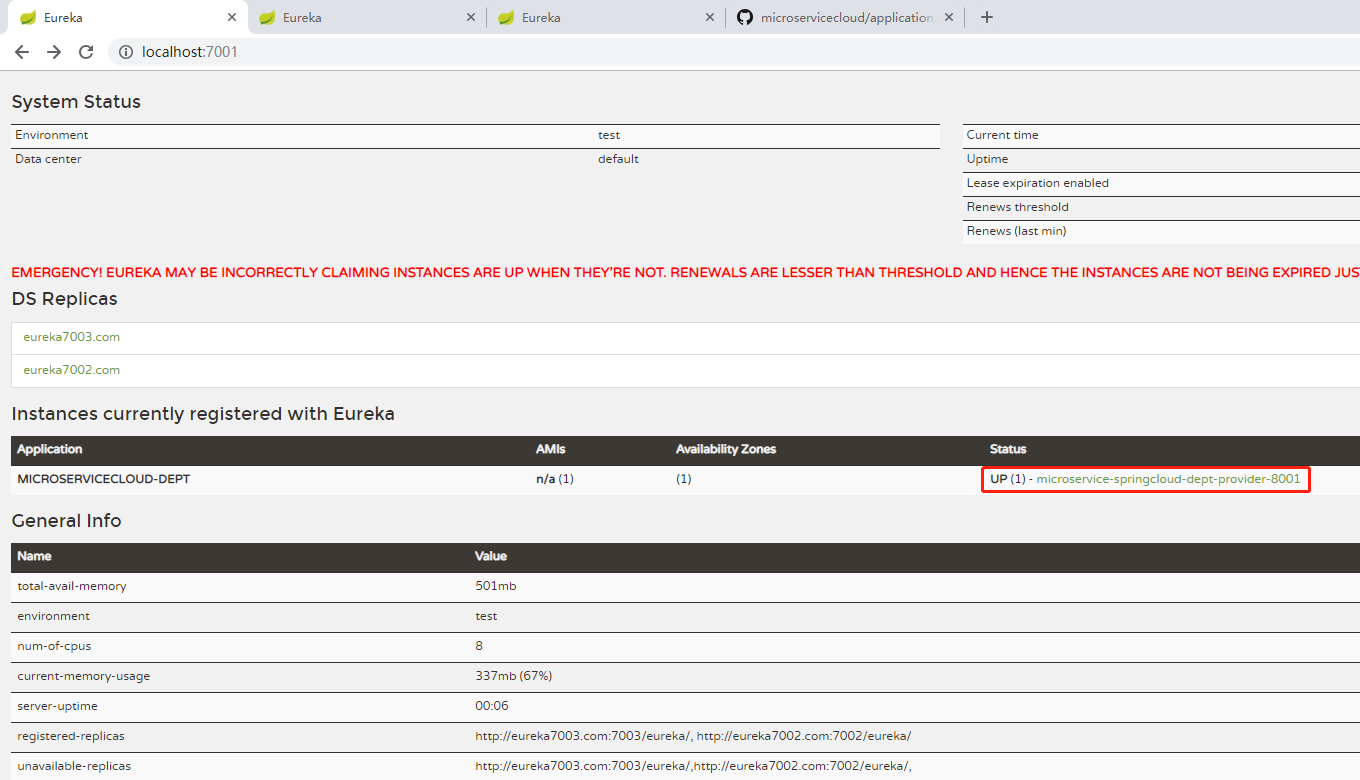
#### /microservice-springcloud-dept-provider-8001微服务发布到上面3台eureka集群配置中

|  |
| --- |
| server:  port: 8001  mybatis:  config-location: classpath:mybatis/mybatis.cfg.xml # mybatis配置文件所在路径  type-aliases-package: com.lk.springcloud.entities # 所有Entity别名类所在包  mapper-locations:  - classpath:mybatis/mapper/\*\*/\*.xml # mapper映射文件  spring:  application:  name: MICROSERVICECLOUD-DEPT  datasource:  type: com.alibaba.druid.pool.DruidDataSource # 当前数据源操作类型  driver-class-name: org.gjt.mm.mysql.Driver # mysql驱动包  url: jdbc:mysql://192.168.106.34:3306/cloudDB01 # 数据库名称  username: root  password: root  dbcp2:  min-idle: 5 # 数据库连接池的最小维持连接数  initial-size: 5 # 初始化连接数  max-total: 5 # 最大连接数  max-wait-millis: 200 # 等待连接获取的最大超时时间  eureka:  client: #客户端注册进eureka服务列表内  service-url:  #defaultZone: http://localhost:7001/eureka  defaultZone: http://eureka7001.com:7001/eureka/,http://eureka7002.com:7002/eureka/,http://eureka7003.com:7003/eureka/  instance:  instance-id: microservice-springcloud-dept-provider-8001  prefer-ip-address: true #访问路径可以显示IP地址  info:  app.name: microservice-springcloud-app-name  company.name: www.lk.com  build.artifactId: $project.artifactId$  build.version: $project.version$ |

启动注册中心集群后



再启动提供者服务后



done

走到这,你肯定发现了,只有服务注册,没有服务发现与消费啊!

别急,看下边

# Ribbon服务发现与负载均衡

在/microservice-springcloud-dept-consumer-9001中

修改pom.xml

### Feign能干什么?

Feign旨在使编写Java Http客户端变得更容易.

前面在使用Ribbon+RestTemplate时,利用RestTemplate对Http请求的封装处理,形成了一套模板化的调用方法,但是实际开发中,由于对服务依赖的调用可能不止一处,往往一个接口会被多处调用,所以通常都会针对每个微服务自行封装一些客户端类来包装这些依赖服务的调用,所以,Feign在此基础上做了进一步封装,由他来帮助我们定义和实现依赖服务接口的定义.在Feign的实现下,我们只需要创建一个接口并使用注解的方式来配置它(以前是Dao接口上面标注Mapper注解,现在是一个微服务接口上面标注一个Feign注解即可),即可完成对服务提供方的接口绑定,简化了Spring cloud Ribbon时,自动封装服务调用客户端的开发量~

### Feign继承了Ribbon

利用Ribbon维护了MicroServiceCloud-Dept的服务列表信息,并且通过轮询实现了客户端的负载均衡,而与Ribbon不同的是,通过Feign只需要定义服务绑定接口且以声明式的方法,优雅而简单的实现了服务调用

# Hystrix断路器

### 概述:

#### Hystrix是什么?

Hystrix是一个用于处理分布式系统的延迟和容错的开源库，在分布式系统里，许多依赖不可避免的会调用失败，比如超时、异常等，Hystrix能够保证在一个依赖出问题的情况下，不会导致整体服务失败，避免级联故障，以提高分布式系统的弹性。

“断路器”本身是一种开关装置，当某个服务单元发生故障之后，通过断路器的故障监控（类似熔断保险丝），向调用方返回一个符合预期的、可处理的备选响应（FallBack），而不是长时间的等待或者抛出调用方无法处理的异常，这样就保证了服务调用方的线程不会被长时间、不必要地占用，从而避免了故障在分布式系统中的蔓延，乃至雪崩。

#### Hystrix能干嘛?

服务降级

服务熔断

服务限流

接近实时的监控

### 服务熔断

#### 服务熔断是什么?

熔断机制是应对雪崩效应的一种微服务链路保护机制。

当扇出链路的某个微服务不可用或者响应时间太长时，会进行服务的降级，进而熔断该节点微服务的调用，快速返回"错误"的响应信息。当检测到该节点微服务调用响应正常后恢复调用链路。在SpringCloud框架里熔断机制通过Hystrix实现。Hystrix会监控微服务间调用的状况，当失败的调用到一定阈值，缺省是5秒内20次调用失败就会启动熔断机制。熔断机制的注解是@HystrixCommand。

#### 依赖:

<dependency>

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

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

</dependency>

#### 注解:

@EnableCircuitBreaker//对hystrixR熔断机制的支持

@HystrixCommand(fallbackMethod = "processHystrix\_Get")

### 服务降级

#### 服务降级是什么?

整体资源快不够了，忍痛将某些服务先关掉，待渡过难关，再开启回来。

### 服务监控hystrixDashboard

#### 概述:

除了隔离依赖服务的调用以外，Hystrix还提供了准实时的调用监控（Hystrix Dashboard），Hystrix会持续地记录所有通过Hystrix发起的请求的执行信息，并以统计报表和图形的形式展示给用户，包括每秒执行多少请求多少成功，多少失败等。Netflix通过hystrix-metrics-event-stream项目实现了对以上指标的监控。Spring Cloud也提供了Hystrix Dashboard的整合，对监控内容转化成可视化界面。

#### 注解:

@EnableHystrixDashboard

#### 依赖

<dependency>

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

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

</dependency>

# Zuul路由网关

### zuul是什么?

Zuul包含了对请求的路由和过滤两个最主要的功能：

其中路由功能负责将外部请求转发到具体的微服务实例上，是实现外部访问统一入口的基础而过滤器功能则负责对请求的处理过程进行干预，是实现请求校验、服务聚合等功能的基础.

Zuul和Eureka进行整合，将Zuul自身注册为Eureka服务治理下的应用，同时从Eureka中获得其他微服务的消息，也即以后的访问微服务都是通过Zuul跳转后获得。

注意：Zuul服务最终还是会注册进Eureka

提供=代理+路由+过滤三大功能

### 能干嘛?

路由

过滤

### 新依赖:

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

# SpringCloud Config分布式配置中心

### 概述:

#### 分布式系统面临的问题—配置问题:

微服务意味着要将单体应用中的额业务拆分成一个个子服务,每个服务的粒度相对较小,因此系统中会出现大量的服务,由于每个服务都需要必要的配置信息才能运行,所以一套集中式的,动态的配置管理设施是必不可少的,springcloud提供了ConfigServer来解决这个问题,我们每一个服务自己带着一个application.yml,上百个的话……

#### springcloud config是什么?

springcloud config为微服务架构中的微服务提供集中化的外部配置支持,配置服务器位各个不同微服务应用的所有环境提供了一个中心化的外部配置

springcloud config 分为服务端和客户端两部分;

服务端也称为分布式配置中心,它是一个独立的微服务应用,用来连接配置服务器并未客户端提供获取配置信息,加密/解密信息等访问接口;

#### springcloud config能干嘛?

1:集中管理配置文件

2:dev/test/prod/beta/release

3:运行期间动态调整配置,不在需要在每个服务部署的机器上编写配置文件,服务会想配置中心统一拉取配置自己的信息

4:当配置发生变动时,服务不要有重启即可感知到配置的变化并应用新的配置

5:将配置信息以REST接口的形式暴露

#### springcloud config与github整合配置?

Git

github