eureka应用入门

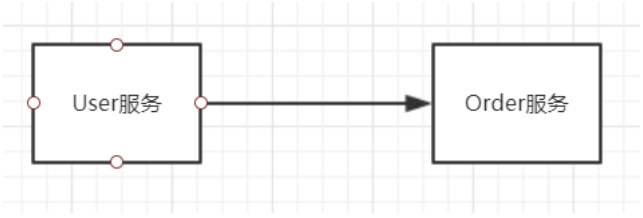
一:没有使用注册中心搭建的分布式应用

1)**服务消费者**(user**服务**) & **提供者**(order**服务**)

名词	定义
服务提供者	服务的被调用方（即：为其他服务提供服务的服务）
服务消费者	服务的调用方（即：依赖其他服务的服务）

2)**如下图**

就是用发发起一个请求，通过用户 Id**查询到该用户下的所有订单信息**



比如现在我的用户服务是占用(User**服务**)8081**端口的服务，此时我的服务提供方**(order**服务端口是**8080)**端口**

我们可以通过RestTemplate **调用方式来进行调用**

//**在用户服务中调用 订单服务**

```
ResponseEntity<List> responseEntity = restTemplate.getForEntity("http://localhost:8080/order/queryOrdersByUserId/"+
```

缺点:

1)**从上面看出的缺点就是，我们的在调用的时候，请求的**Ip**地址和端口是硬编码的**.

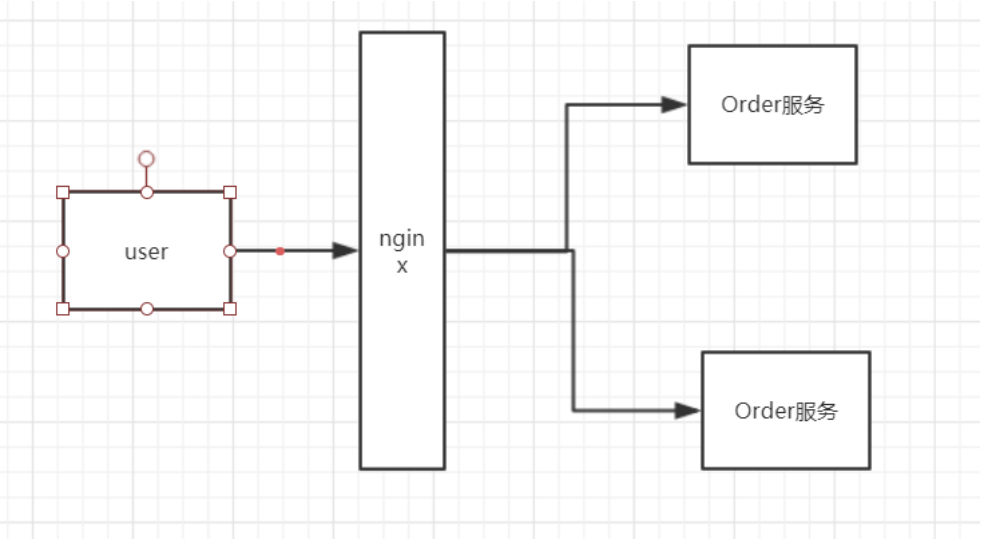
若此时，服务提供方(order)**服务部署的机器换了端口或者是更换了部署机器的**Ip,**那么我们需要修改代码重新发布部署**.

2) **假设我们的**order**服务压力过大，我们需要把**order**服务作为集群，那么意味着** order**是多节点部署**

比如原来的，我们只有一台服务器，现在有多台服务器，那么作为运维人员 需要在服务消费方进行手工维护一份注册表(容易出错)

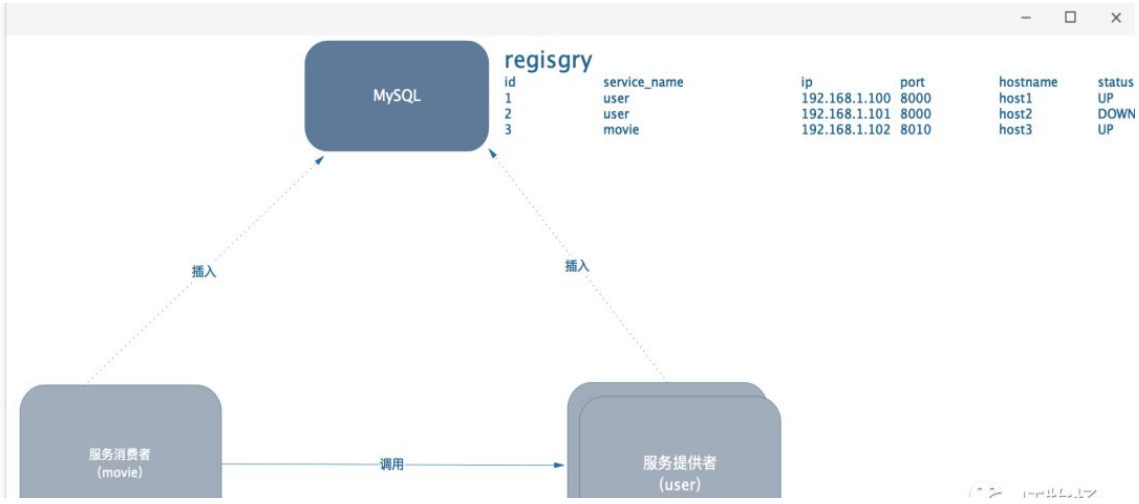
3)**有人马上回驳我说，我可以通过**ng**来做负载均衡,对，我首先认为这是可行的，当时微服务成百上千的服务，难道我们要那成百上千**

ng**么？或者使用一个**Ng **那么我们能想一下哪个**ng**的配置文件有多么复杂。**



3)服务发现原理初探

其实，服务发现机制非常简单，不妨用大家熟悉的 MySQL(**也可以用**redis**来做存储)来类比——只需一张表（图中的**registry**表）即可实现服务发现**



- 应用启动时，自动往registry表中插入一条数据，数据包括服务名称、IP、端口等信息。
- 应用停止时，自动把自己在registry表中的数据的status设为 `DOWN` 。
- 服务调用，每次调用前，去mysql中去查询 select * from register where server_name='服务名称' and status='UP'的记录，然后替换原来的调用路径地址

缺点: **上面这个毕竟是一个很简陋的服务注册中心**

缺点1:若当前服务宕机，我们必须要把自己的对应的数据标记为down

缺点2：服务每次调用都要发送select 语句，mysql的压力很大

缺点3：若mysql（服务注册中心挂了，我们也应该提供服务调用）

真正的服务注册发现组件应该满足哪些条件才称为一个合格的服务发现注册组件

1）**提供一个服务注册接口，其他微服务启动的时候，把自己的**IP,**端口都在服务端注册下来**

2）**提供一个服务发现接口，也就是微服务可以通过**service_id **来获取到 对应微服务的网络地址和端口**

3)**各个微服务与服务注册组件，使用心跳检查机制，若服务注册组件长时间和某微服务 没有通信，那么就应该剔除服务**

4)**客户端缓存**，各个微服务将需要调用服务的地址缓存在本地，并使用一定机制更新（例如定时任务更新、事件推送更新等）。这样既能降低服务发现组件的压力，同　　时，即使服务发现组件出问题，也不会影响到服务之间的调用

……

二:Eureka**入门**

eureka**分为服务端和客户端**

2.1**）搭建**eureka**服务端**

三板斧:

①加依赖

```
<dependency>
      <groupId>org.springframework.cloud</groupId>
      <artifactId>spring-cloud-starter-netflix-eureka-server</artifactId>
</dependency>
```

②:写注解 在主启动类上 写上@EnableEurekaServer**注解**

```
@EnableEurekaServer
public class Tulingvip01MsEurekaServerApplication {
}
```

③:编写配置文件

```
#eureka服务端应用的端口默认是8761
server.port=9000
#表示是否将自己注册到Eureka Server,默认为true,由于当前应用就是Eureka Server,故而设为false
eureka.client.register-with-eureka=false
# 表示是否从Eureka Server获取注册信息,默认为true,因为这是一个单点的Eureka Server,不需要同步其他的Eureka Server节点的数据
eureka.client.fetch-registry=false
#暴露给其他eureka client 的注册地址
eureka.client.service-url.defaultZone: http://localhost:9000/eureka/
```

2.2)**搭建**eureka**客户端**

2.2.1)**服务消费者** tulingvip01-ms-consumer-user-8001

①：加依赖

```
<dependency>
      <groupId>org.springframework.cloud</groupId>
      <artifactId>spring-cloud-starter-netflix-eureka-client</artifactId>
</dependency>
```

②：加入注解@EnableDiscoveryClient

```
@EnableDiscoveryClient
public class Tulingvip01MsConsumerUser8001Application
```

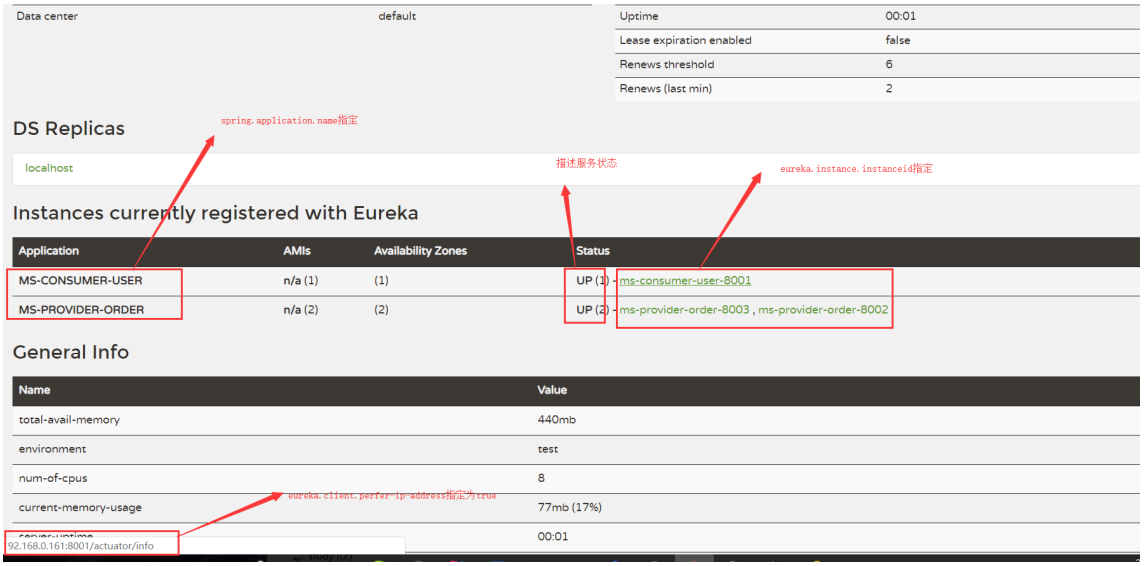
③：编写配置文件

```
server.port=8001
#注册到eureka服务端的微服务名称
spring.application.name=ms-consumer-user
#注册到eureka服务端的地址
eureka.client.service-url.defaultZone=http://localhost:9000/eureka/
#点击具体的微服务，右下角是否显示ip
eureka.instance.prefer-ip-address=true
#显示微服务的名称
eureka.instance.instance-id=ms-consumer-user-8001

mybatis.configuration.map-underscore-to-camel-case=true
#配置数据库
spring.datasource.url=jdbc:mysql://47.104.128.12:3306/tuling-cloud
spring.datasource.driver-class-name=com.mysql.jdbc.Driver
spring.datasource.username=root
spring.datasource.password=123456
logging.level.com.tuling.dao=debug
```

④: 调用 使用 RestTemplate **（调用的时候不再是通过**Ip **和地址发起调用，而是通过微服务名称进行调用）**

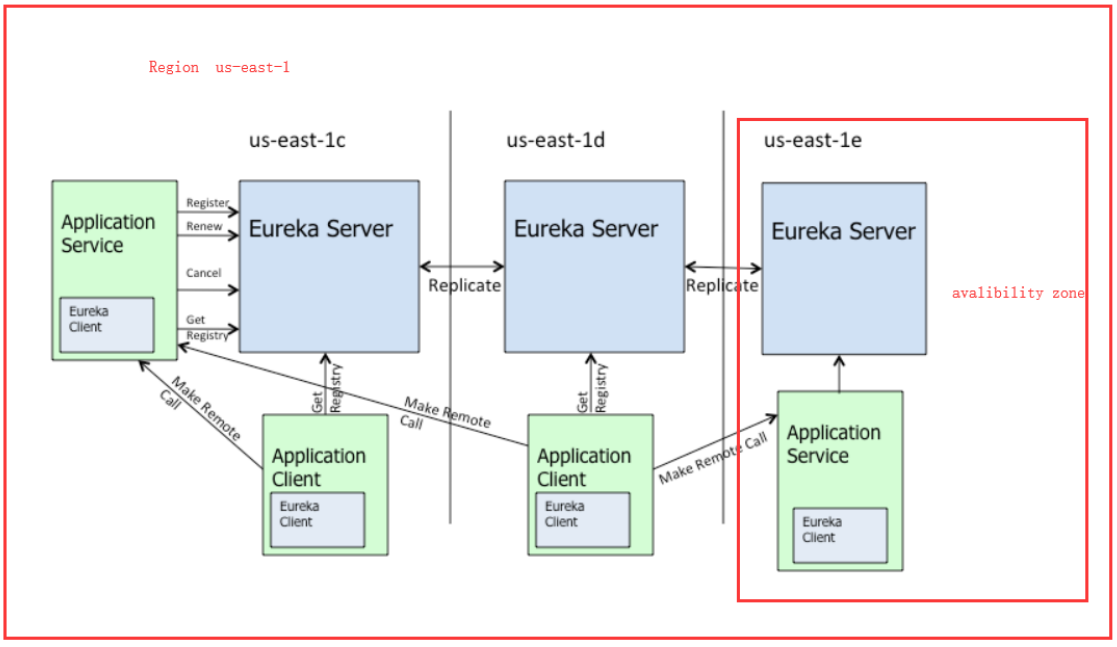
需要加入@LoadBanlance **注解**

```
    @RequestMapping("/queryUserInfoById/{userId}")
    public UserInfoVo queryUserInfoById(@PathVariable("userId") Integer userId) {
        User user = userServiceImpl.queryUserById(userId);

        ResponseEntity<List> responseEntity = restTemplate.getForEntity("http://MS-PROVIDER-ORDER/order/queryOrders
        List<OrderVo> orderVoList = responseEntity.getBody();

        UserInfoVo userInfoVo = new UserInfoVo();
        userInfoVo.setOrderVoList(orderVoList);
        userInfoVo.setUserName(user.getUserName());
        return userInfoVo;
    }

@Configuration
public class MainConfig {

    @Bean
    @LoadBalanced
    public RestTemplate restTemplate() {
        return new RestTemplate();
    }
}
```

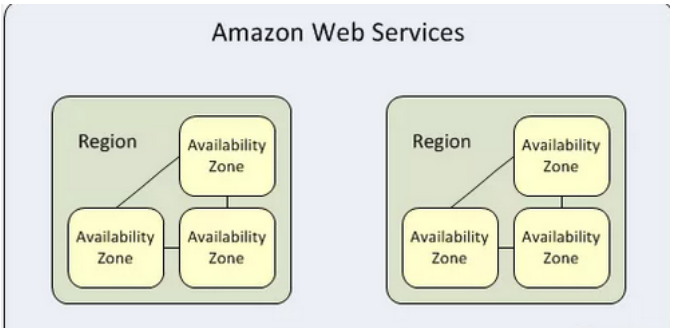


3）Eureka **部署架构**

3.1)**首先，大家看到这个图 脑瓜子是不是嗡嗡嗡的？什么鬼** us-east-1c us-east-1d us-east-1e**是什么东西？？**



我们首先需要了解一下 aws(amazon web service) **可以类比中国的阿里云服务器.**



Region(us-east-1,) **可以类比于 阿里云的不同区域，我们在买阿里云服务器的时候，是不是需要你选择区域么？各个**

Region **之间的内网是不相通的**

区域	编码
亚太（东京）	ap-northeast-1
亚太（新加坡）	ap-southeast-1
亚太（悉尼）	ap-southeast-2
欧洲（爱尔兰）	eu-west-1
南美（圣保罗）	sa-east-1
美东（北佛杰尼亚）	us-east-1
美西（北加利佛尼亚）	us-west-1
美西（俄勒冈）	us-west-2

Availabitlity zone **可用区**(us-east-1c　us-east-1d　us-east-1e) **可以理解为 我们 一个地区比如华南地区，不同城市的机房，内网是相通的**

比如华南1　**华东**1 2 ，**华北**12345

- Eureka Server提供服务发现的能力，各个微服务启动时，会向Eureka Server注册自己的信息（例如IP、端口、微服务名称等），Eureka Server会存储这些信息；
- Eureka Client是一个Java客户端，用于简化与Eureka Server的交互(**启动的时候，会向**server**注册自己的信息**)
- **微服务启动后** renew，会周期性（**默认**30**秒**）地向Eureka Server发送心跳以续约自己的“租期”；
- 如果Eureka Server在一定时间内没有接收到某个微服务实例的心跳（最后一次续约时间开始计算），Eureka Server将会注销该实例（**默认**90**秒**）；
- 默认情况下，Eureka Server**同时也是**Eureka Client**。多个**Eureka Server**实例，互相之间通过增量复制的方式，来实现服务注册表中数据的同步。**Eureka Server默认保证在90秒内，Eureka Server集群内的所有实例中的数据达到一致（从这个架构来看，Eureka Server所有实例所处的角色都是**对等**的，没有类似Zookeeper、选举过程，也不存在主从，**所有的节点都是主节点。**Eureka官方将Eureka Server集群中的所有实例称为“**对等体（**peer**）**”）
- Eureka Client**会缓存服务注册表**中的信息。这种方式有一定的优势——首先，微服务无需每次请求都查询Eureka Server，从而降低了Eureka Server的压力；其次，即使Eureka Server所有节点都宕掉，服务消费者依然可以使用缓存中的信息找到服务提供者并完成调用

3.2)Eureka**配置的总要信息讲解**

①:**服务注册配置**

eureka.client.register-with-eureka=true

该项配置说明,是否向注册中心注册自己,在非集群环境下设置为false，表示自己不注册自己

eureka.client.fetch-registry=true

该项配置说明，注册中心只要维护微服务实例清单，非集群环境下,不需要作检索服务,所有也设置为false

②：服务续约配置

eureka.instance.lease-renewal-interval-in-seconds=30(**默认**)

该配置说明，服务提供者会维持一个心跳告诉eureka server **我还活着，这个就是一个心跳周期**

eureka.instance.lease-expiration-duration-in-seconds=90(**默认**)

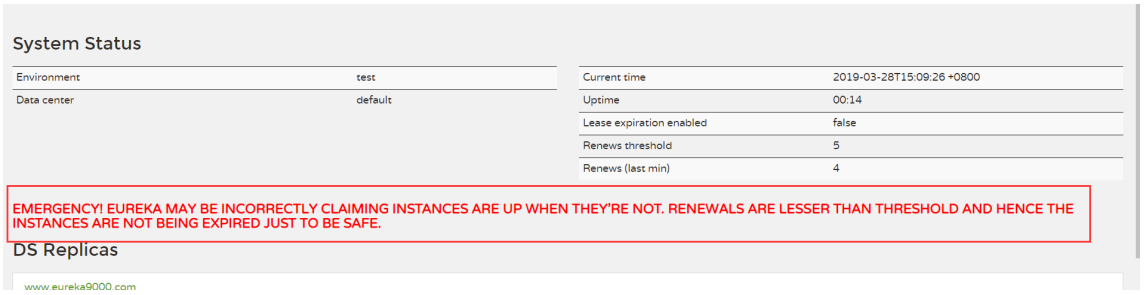
该配置说明,你的最后一次续约时间开始，往后推90s **还没接受到你的心跳,那么我就需要把你剔除.**

③:**获取服务配置 (前提,**eureka.client.fetch-registry**为**true)

eureka.client.registry-fetch-interval-seconds=30

缓存在调用方的微服务实例清单刷新时间

4)eureka **的自我保护功能**



默认情况下，若eureka server **在一段时间内（**90s**）没有接受到某个微服务实例的心跳，那么** eureka server **就会剔除**

该实例，当时由于网络分区故障，导致eureka server **和 服务之间无法通信，此时这个情况就变得很可怕---因为微服务是实例健康的**

,本不应注销该实例.

那么通过eureka server **自我保护模式就起作用了,当**eureka server**节点短时间**(15min**是否低于**85%)**丢失过多客户端是(由于网络分区)，那么该**eureka server**节点就会进入自我保护模式，**eureka server **就会自动保护注册表中的微服务实例，不再删除该注册表中微服务的信息，等到网络恢复，**eureka server **节点就会退出自我保护功能(宁可放过一千，不要错杀一个)**

5)eureka **的高可用 搭建**

启动二个eureka server　9000**端口　和**9001**端口**

5.1)9000 **端口的工程配置**

```
server.port=9000
#表示是否将自己注册到Eureka Server 表示9000的服务端需要向9001工程注册自己
eureka.client.register-with-eureka=true
# 表示是否从Eureka Server获取注册信息,默认为true,需要从9001上同步数据
eureka.client.fetch-registry=true
讲自己注册到9001上去
eureka.client.service-url.defaultZone: http://www.eureka9001.com:9001/eureka/
```

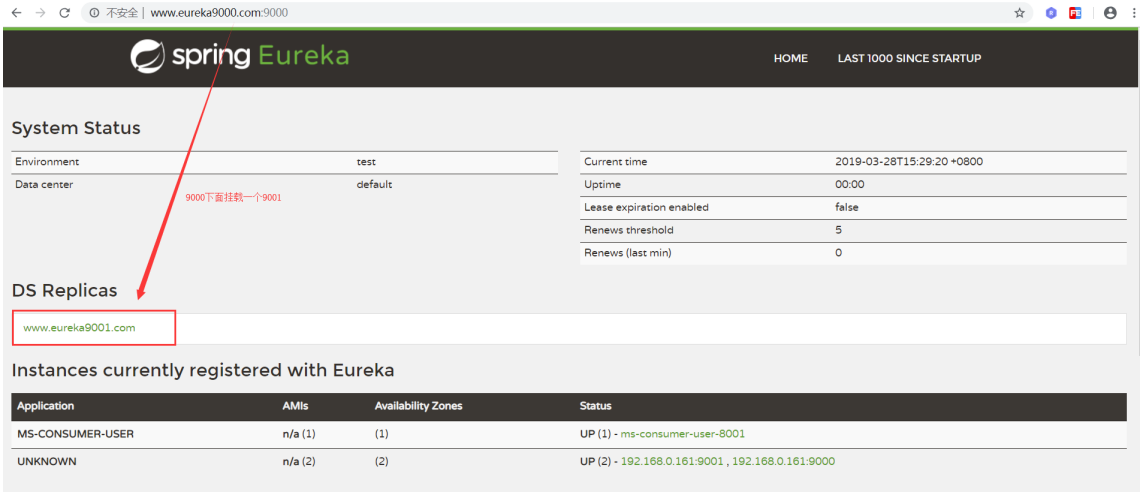
5.2) 9001**端口配置**

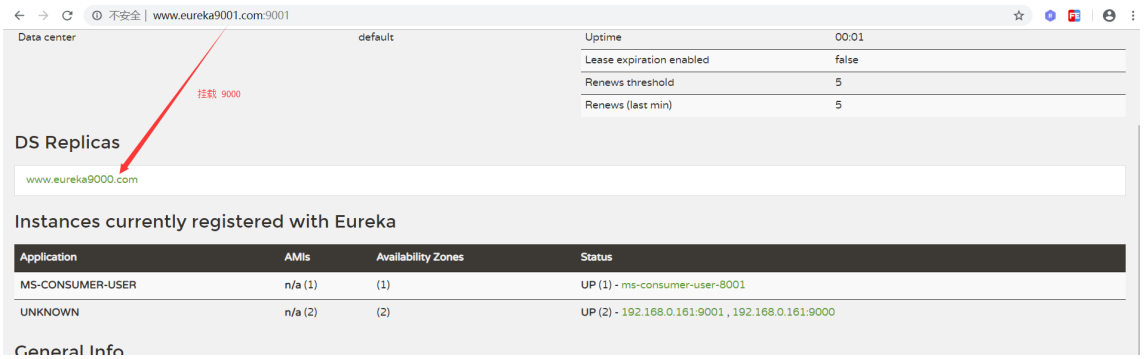
```
server.port=9001
#表示是否将自己注册到Eureka Server 表示9001的服务端需要向9000工程注册自己
eureka.client.register-with-eureka=true
# 表示是否从Eureka Server获取注册信息,默认为true,需要从9000上同步数据
eureka.client.fetch-registry=true
讲自己注册到9000上去
eureka.client.service-url.defaultZone: http://www.eureka9000.com:9000/eureka/
```

5.3)**微服务提供者配置,需要向二个注册中心进行注册**

```
eureka.client.service-url.defaultZone=http://www.eureka9000.com:9000/eureka/,http://www.eureka9001.com:9001/eu
```

5.4)**截图**





6)**安全配置 我们现在访问**eureka server **直接访问 没有授权**

6.1)**导入安全配置 依赖**

```
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-security</artifactId>
        </dependency>
```

6.2)**修改**yml**配置文件**

```
//注册的时候需要带入 用户名 密码  基本格式为  http:用户名:密码@ip:端口/eureka/
eureka.client.service-url.defaultZone: http://${spring.security.user.name}:${spring.security.user.name}@www.eureka9000

//开启默认的认真
spring.security.basic.enable=true
spring.security.user.name=root
spring.security.user.password=123456




==========================================================客户端配置
注册地址修改为这个
eureka.client.service-url.defaultZone=http://${security.login.username}:${security.login.pass}@www.eureka9000.com:90
```