

Eureka

一 服务发现

1. 服务发现与注册

在微服务架构中，服务发现（Service Discovery）是关键原则之一。手动配置每个客户端或某种形式的约定是很难做的，并且很脆弱。Spring Cloud提供了多种服务发现的实现方式，例如：Eureka、Consul、Zookeeper。Spring Cloud支持得最好的是Eureka，其次是Consul，最次是Zookeeper。

2. 服务发现模式

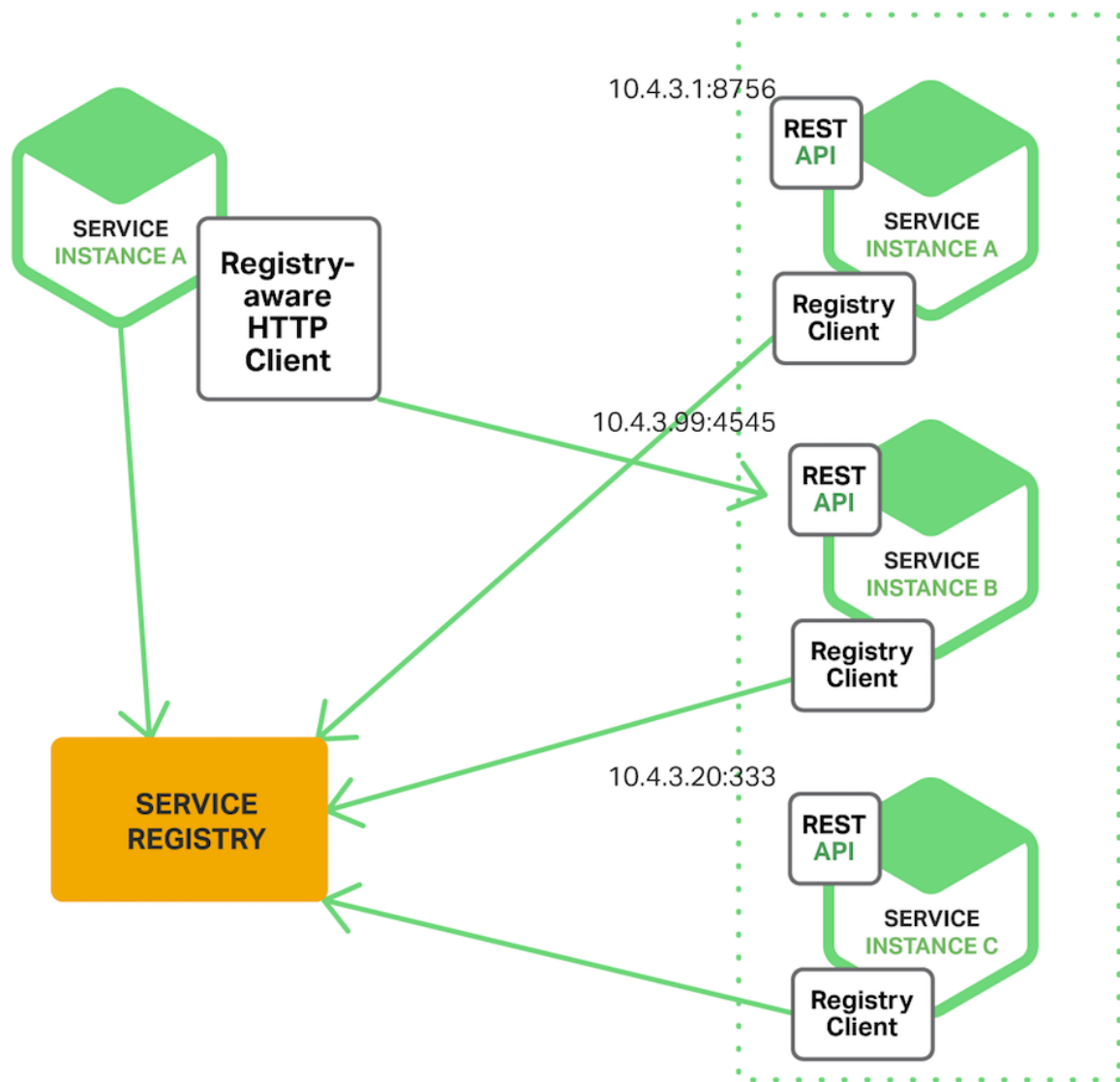
服务发现有两种模式,一是客户端发现模式,二是服务端发现模式

2.1 客户端发现模式

使用客户端发现模式时，客户端(服务消费者)决定相应服务实例(服务提供者)的网络位置，并且对请求实现负载均衡。客户端查询服务注册表，服务注册表是一个保存有所有可用服务的一个数据库等具有存储功能的地方；然后客户端使用负载均衡算法从中选择一个提供者实例，并发出请求。相当于我们去吃自助,吃什么,吃多少都是我们自己决定的,哪怕其中的菜每种都有多盘,我们也可以自己决定从哪盘中拿,他们只需要把所有的东西摆放在那里就行

下图显示了这种模式的架构：

下图中可以看出左侧的消费者(客户端)自己从注册中心查询右侧的所有的服务,然后自己决定去请求哪个服务



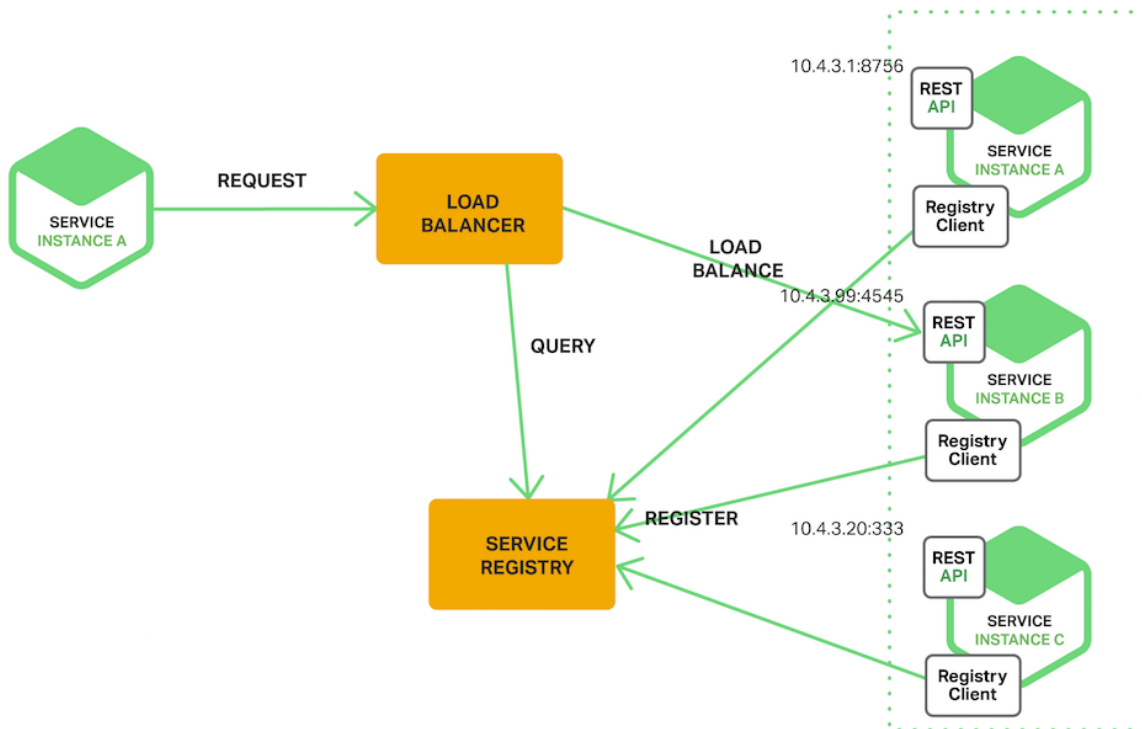
2.2 服务端发现模式

与客户端发现方式不同,服务端发现是由专门的某个或者某几个服务器来发现服务后再对外提供信息的,客户端(消费者)并不知道这个服务到底有多少个,就像我们在食堂打饭,我们只需要告诉打饭的阿姨你吃什么,阿姨她来决定从哪盘里面打给你(当然食堂里面你还能看到菜而已,我们的服务中你无法看到服务)

下图就是服务的发现的一个结构图:

可以看到我们的服务消费者并不是直接再连接注册中心查询服务并决定加载哪个了,而是连接负载均衡服务器告诉它我需要一个什么服务,负载均衡服务器会去注册中心查看有哪些服务,并通过负载均衡返回一个服务交给消费者调用,这个时候客户端是无法决定自己到底访问哪个实例的

服务端发现模式它最大的优点是客户端无需关注发现的细节,只需要简单地向负载均衡器发送请求,这减少了编程语言框架需要完成的发现逻辑。并且如上文所述,某些部署环境免费提供这一功能。这种模式也有缺点。除非负载均衡器由部署环境提供,否则会成为需要配置和管理的高可用系统组件



2.3 服务注册表

服务注册表是服务发现的核心部分,是包含服务实例的网络地址的数据库。服务注册表需要高可用而且随时更新。客户端能够缓存从服务注册表中获取的网络地址,然而,这些信息最终会过时,客户端也就无法发现服务实例。因此,服务注册表会包含若干服务端,使用复制协议保持一致性。

常见的服务注册表: Eureka,Zookeeper,Consul

3 小结

在微服务应用中，服务实例的运行环境会动态变化，实例网络地址也是如此。因此，客户端为了访问服务必须使用服务发现机制。

服务注册表是服务发现的关键部分。服务注册表是可用服务实例的数据库，提供管理 API 和查询 API。服务实例使用管理 API 来实现注册和注销，系统组件使用查询 API 来发现可用的服务实例。

服务发现有两种主要模式：客户端发现和服务端发现。在使用客户端服务发现的系统中，客户端查询服务注册表，选择可用的服务实例，然后发出请求。在使用服务端发现的系统中，客户端通过路由转发请求，路由器查询服务注册表并转发请求到可用的实例。

服务实例的注册和注销也有两种方式。一种是服务实例自己注册到服务注册表中，即自注册模式；另一种则是由其它系统组件处理注册和注销，也就是第三方注册模式。

在一些部署环境中，需要使用 Netflix Eureka、etcd、Apache Zookeeper 等服务发现来设置自己的服务发现基础设施。而另一些部署环境则内置了服务发现。例如，Kubernetes 和 Marathon 处理服务实例的注册和注销，它们也在每个集群主机上运行代理，这个代理具有服务端发现路由的功能。

HTTP 反向代理和 NGINX 这样的负载均衡器能够用做服务器端的服务发现均衡器。服务注册表能够将路由信息推送到 NGINX，激活配置更新，譬如使用 Consul Template。NGINX Plus 支持额外的动态配置机制，能够通过 DNS 从注册表中获取服务实例的信息，并为远程配置提供 API

二 Eureka

2.1 Eureka介绍

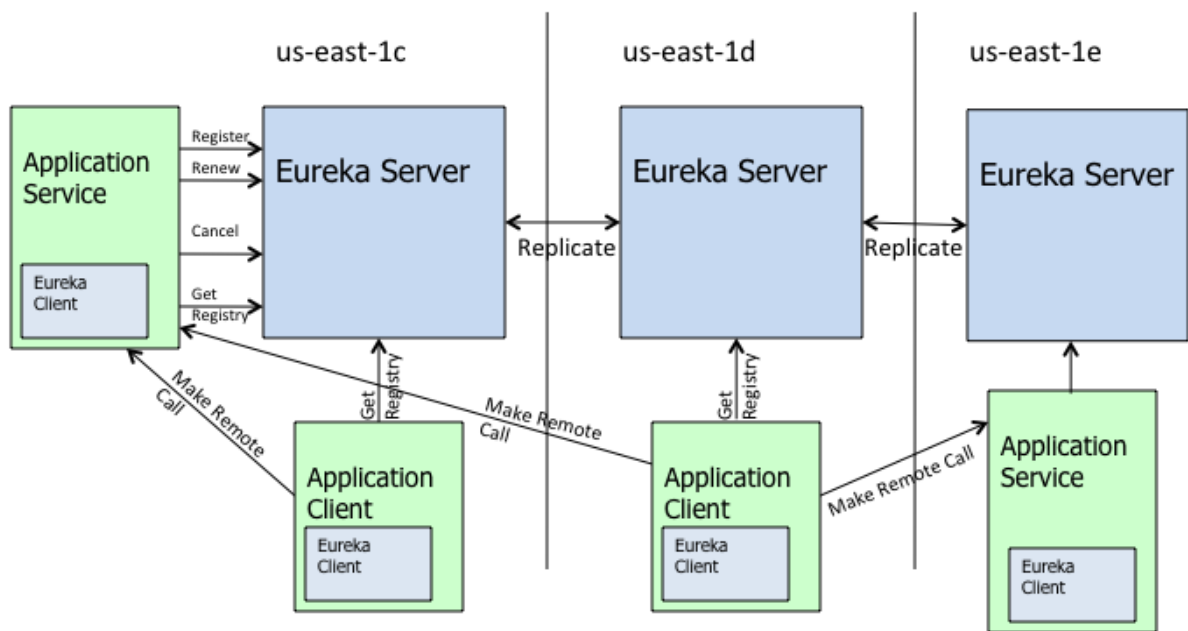
Eureka是Netflix开发的服务发现框架，本身是一个基于REST的服务，主要用于定位运行在AWS域中的中间层服务，以达到负载均衡和中间层服务故障转移的目的。Spring Cloud将它集成在其子项目spring-cloud-netflix中，以实现Spring Cloud的服务发现功能。

Github地址: <https://github.com/Netflix/eureka>

2.2 Eureka原理

wiki地址: <https://github.com/Netflix/eureka/wiki/Eureka-at-a-glance>

下图是Eureka的结构图:



上图是来自Eureka官方的架构图，大致描述了Eureka集群的工作过程

- Application Service 就相当于服务提供者，Application Client就相当于服务消费者；
- Make Remote Call，可以简单理解为调用RESTful的接口；
- us-east-1c、us-east-1d等是zone，它们都属于us-east-1这个region；类似于我们阿里云上面的华东2区下的青岛，其中青岛就是zone，华东就是region

由图可知，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Eureka Server中进行注册，这样Eureka Server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。

Eureka Client是一个Java客户端，用于简化与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，Eureka Server将会从服务注册表中把这个服务节点移除（默认90秒）。

Eureka Server之间将会通过复制的方式完成数据的同步。

Eureka还提供了客户端缓存的机制，即使所有的Eureka Server都挂掉，客户端依然可以利用缓存中的信息消费其他服务的API。

综上，Eureka通过心跳检测、健康检查、客户端缓存等机制，确保了系统的高可用性、灵活性和可伸缩性。

2.3 Eureka和Zookeeper比较

著名的CAP理论指出，一个分布式系统不可能同时满足C(一致性)、A(可用性)和P(分区容错性)。由于分区容错性在是分布式系统中必须要保证的，因此我们只能在A和C之间进行权衡。在此Zookeeper保证的是CP, 而Eureka则是AP。

2.3.1 Zookeeper保证CP

当向注册中心查询服务列表时，我们可以容忍注册中心返回的是几分钟以前的注册信息，但不能接受服务直接down掉不可用。也就是说，服务注册功能对可用性的要求要高于一致性。但是zk会出现这样一种情况，当master节点因为网络故障与其他节点失去联系时，剩余节点会重新进行leader选举。问题在于，选举leader的时间太长，30 ~ 120s，且选举期间整个zk集群都是不可用的，这就导致在选举期间注册服务瘫痪。在云部署的环境下，因网络问题使得zk集群失去master节点是较大概率会发生的事，虽然服务能够最终恢复，但是漫长的选举时间导致的注册长期不可用是不能容忍的。

2.3.2 Eureka保证AP

Eureka看明白了这一点，因此在设计时就优先保证可用性。Eureka各个节点都是平等的，几个节点挂掉不会影响正常节点的工作，剩余的节点依然可以提供注册和查询服务。而Eureka的客户端在向某个Eureka注册或如果发现连接失败，则会自动切换至其它节点，只要有一台Eureka还在，就能保证注册服务可用(保证可用性)，只不过查到的信息可能不是最新的(不保证强一致性)。除此之外，Eureka还有一种自我保护机制，如果在15分钟内超过85%的节点都没有正常的心跳，那么Eureka就认为客户端与注册中心出现了网络故障，此时会出现以下几种情况：

1. Eureka不再从注册列表中移除因为长时间没收到心跳而应该过期的服务
2. Eureka仍然能够接受新服务的注册和查询请求，但是不会被同步到其它节点上(即保证当前节点依然可用)
3. 当网络稳定时，当前实例新的注册信息会被同步到其它节点中

因此，Eureka可以很好的应对因网络故障导致部分节点失去联系的情况，而不会像zookeeper那样使整个注册服务瘫痪。

Eureka作为单纯的服务注册中心来说要比zookeeper更加“专业”，因为注册服务更重要的是可用性，我们可以接受短期内达不到一致性的状况。不过Eureka目前1.X版本的实现是基于servlet的 **Java** web应用，它的极限性能肯定会受到影响。期待正在开发之中的2.X版本能够从servlet中独立出来成为单独可部署执行的服务。

2.4 Eureka 基本使用

2.4.1 搭建Eureka服务端

因为作为注册中心一定是独立于其他程序之外的,所以我们需要搭建一个Eureka的服务端,与Zookeeper不同的是,Eureka的服务端是一个JavaWeb程序

参考代码中 03eurekaserver

2.4.1.1 Pom.xml

此pom中只包含了依赖,

```
<dependencies>
  <!--Eureka server 的依赖-->
  <dependency>
    <groupId>org.springframework.cloud</groupId>
    <artifactId>spring-cloud-starter-netflix-eureka-
server</artifactId>
  </dependency>

</dependencies>
```

2.4.1.2 application.yml

在resources目录下创建application.yml

```
server:
  port: 10000 #程序通过springboot启动后tomcat的端口
eureka:
  client:
    register-with-eureka: false #eureka单机版的配置
    fetchRegistry: false #表示当前是个服务端
    service-url:
      defaultZone: http://localhost:10000/eureka #Eureka对外提供服务的注册地
      址,也就是eureka最终运行的地址,此处我们使用的是localhost,springboot模式启动,所以是
      localhost:10000/eureka
  spring:
    application:
      name: eureka-server #程序的名字
```

2.4.1.3 启动主程序

Eureka默认情况下不需要其他的额外代码,只需要一个主程序即可

```
import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import
org.springframework.cloud.netflix.eureka.server.EnableEurekaServer;

/**
 * Created by jackiechan on 19-6-8 上午2:50
 *
 * @Author jackiechan
 */
@SpringBootApplication
```



```
@EnableEurekaServer//开启eurekaServer,会自动帮我们配置
public class EurekaStartApp {

    public static void main (String[] args){
        SpringApplication.run(EurekaStartApp.class,args);
    }
}
```

2.4.1.4 效果

启动后访问 <http://localhost:10000/>

The screenshot shows the Spring Eureka web interface in a browser. The page has a dark header with the 'spring Eureka' logo and navigation links for 'HOME' and 'LAST 1000 SINCE STARTUP'. The main content area is divided into several sections:

- System Status:** A table showing environment details (Environment: test, Data center: default) and system metrics (Current time: 2019-07-02T10:00:59 +0800, Uptime: 00:00, Lease expiration enabled: false, Renew threshold: 1, Renew last min: 0).
- DS Replicas:** A section titled 'Eureka集群中的机器,当前是单机的' (Machines in the Eureka cluster, currently a single machine) showing a single instance 'localhost'.
- Instances currently registered with Eureka:** A section titled '用于显示当前注册到Eureka上面的服务示例' (Used to display service examples currently registered on Eureka) showing a table with columns 'Application', 'AMIs', 'Availability Zones', and 'Status'. It currently shows 'No instances available'.
- General Info:** A section titled '服务器信息' (Server information) showing a table of system metrics:

Name	Value
total-avail-memory	435mb
environment	test
num-of-cpus	8
current-memory-usage	164mb (37%)
server-uptime	00:00
registered-replicas	http://localhost:10000/eureka/
unavailable-replicas	http://localhost:10000/eureka/
available-replicas	

2.4.2 服务提供者修改

参考 04provider-eureka,消费者配置同提供者,但是调用的时候需要修改一些东西,所以消费者单独写

2.4.2.1 POM.xml中添加依赖

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

2.4.2.2 application.yml

```
server:
  port: 8001
spring:
  datasource:
    username: root
    password: qishimeiyoumima
    driver-class-name: com.mysql.jdbc.Driver
    url: jdbc:mysql:///db_shopping?useUnicode=true&characterEncoding=utf8
    type: com.alibaba.druid.pool.DruidDataSource
#添加程序的名字
  application:
    name: 04provider-eureka #注意这个名字,eureka server 区分服务的方式就是通过
    这个名字,相同名字的服务会被归为集群
#配置eureka服务端地址
  eureka: #注册中心的地址
    client:
      serviceUrl:
        defaultZone: http://localhost:10000/eureka #这里说eureka server运行起
        来后的地址
```

2.4.2.3 主程序修改

在主程序中ProviderStarter添加注解,注册到eureka

```
import org.mybatis.spring.annotation.MapperScan;
import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.cloud.netflix.eureka.EnableEurekaClient;

/**
 * Created by jackiechan on 19-6-8 上午12:09
 *
 * @Author jackiechan
```

```

*/
@SpringBootApplication
@MapperScan("com.qianfeng.microservice.provider.mapper")
@EnableEurekaClient //开启服务注册和发现, 只要添加此注解
public class ProviderStarter {

    public static void main (String[] args){
        SpringApplication.run(ProviderStarter.class,args);
    }
}

```

2.4.2.4 启动提供者

启动后再刷新eureka网页,可以看到有一个服务已经注册上来了

The screenshot shows the Spring Eureka web interface at localhost:10000. The interface includes a header with the Spring Eureka logo and navigation links for HOME and LAST 1000 SINCE STARTUP. The main content area is divided into several sections:

- System Status:** A table showing environment (test), data center (default), current time (2019-07-02T11:25:44 +0800), uptime (00:00), lease expiration enabled (false), renews threshold (3), and renews (last min) (0).
- DS Replicas:** A section titled "Eureka集群列表,当前是单机" (Eureka cluster list, currently single machine) showing a single replica at localhost.
- Instances currently registered with Eureka:** A table titled "当前注册到eureka上面的实例列表" (Instances currently registered on Eureka). It shows one instance:

Application	AMIs	Availability Zones	Status
04PROVIDER-EUREKA <small>应用的名字,相同应用的会被认为是集群 机器信息会显示在一起</small>	n/a (1)	(1)	UP (1) - bogon:04provider-eureka:8001 <small>状态显示,当前是UP, Down为失效</small>
- General Info:** A table showing various system metrics:

Name	Value
total-avail-memory	618mb
environment	test
num-of-cpus	8
current-memory-usage	273mb (44%)
server-uptime	00:00
registered-replicas	http://localhost:10000/eureka/
unavailable-replicas	http://localhost:10000/eureka/

Red arrows point from the "UP (1) - bogon:04provider-eureka:8001" status and the "registered-replicas" URL to the "bogon:8001/actuator/info" link at the bottom left.

2.5 Eureka 密码保护

2.5.1 Eureka Server修改

2.5.1.1 POM.xml添加依赖

SpringCloud通过Spring security来进行安全管理,所以需要导入这个依赖包

```
<!--
```

```
    安全组件,用于设置访问的时候需要密码
```

```
-->
```

```
<dependency>
```

```
    <groupId>org.springframework.boot</groupId>
```

```
    <artifactId>spring-boot-starter-security</artifactId>
```

```
</dependency>
```

2.5.1.2 设置密码

导入security后如果不设置用户名密码会有默认的用户名密码,用户名是user,密码是每次启动的时候会在控制台打印一个随机密码,所以我们需要在application.yml中设置密码

```
server:
  port: 10000 #程序通过springboot启动后tomcat的端口
eureka:
  client:
    register-with-eureka: false #eureka单机版的配置
    fetchRegistry: false
    serviceUrl:
      defaultZone: http://localhost:10000/eureka #Eureka对外提供服务的注册地址,也就是eureka最终运行的地址,此处我们使用的是localhost,springboot模式启动,所以是localhost:10000/eureka
spring:
  application:
    name: eureka-server #程序的名字
  security: #设置密码,注意是在spring层级下面
    user:
      password: abc #密码
      name: zhangsan #用户名
```

2.5.1.3 修改安全配置

当我们开启安全配置后,我们发现我们的服务会无法注册到eureka上面,即便是客户端也指定了用户名和密码仍旧不可以,原因是spring security 2开始,默认对csrf进行了拦截,导致我们的服务注册也会被拦截,而我们的springcloud F版本使用的就是springboot2,那么导入的security依赖也是2的版本,所以我们需要修改安全配置来允许

服务注册

在项目中添加一个类

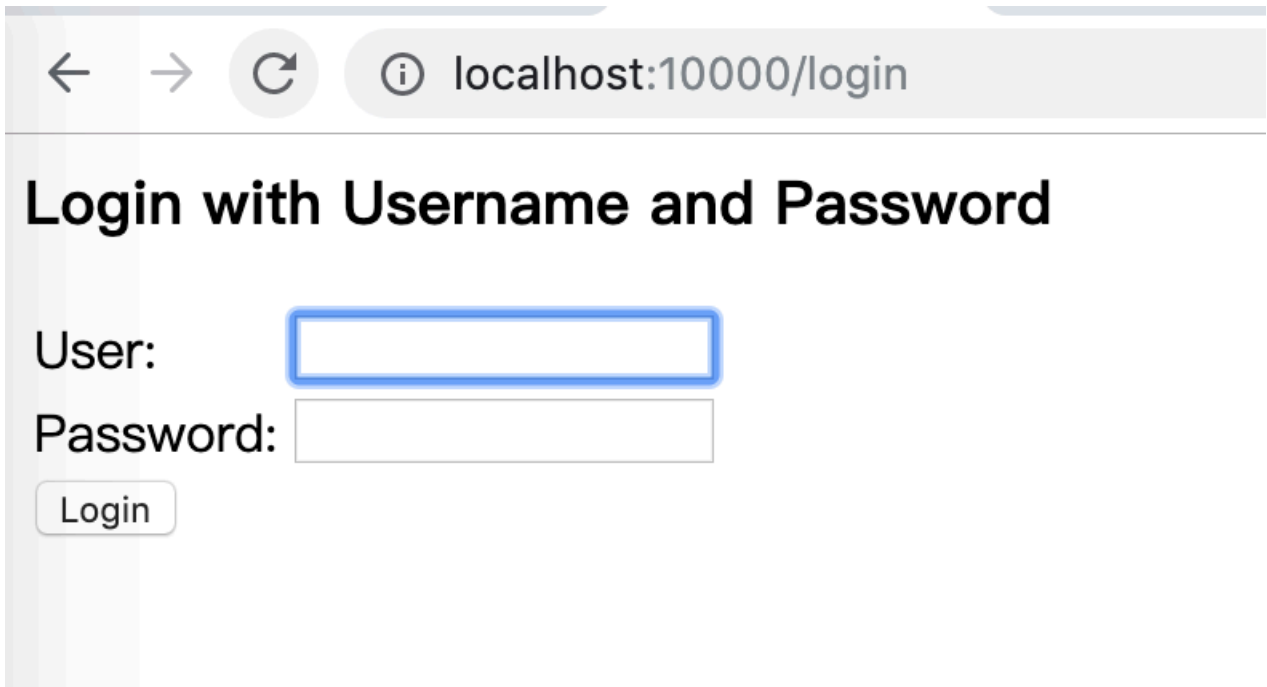
```
import org.springframework.context.annotation.Configuration;
import
org.springframework.security.config.annotation.web.builders.HttpSecurity;
import
org.springframework.security.config.annotation.web.configuration.EnableWe
bSecurity;
import
org.springframework.security.config.annotation.web.configuration.WebSecur
ityConfigurerAdapter;

/**
 * Created by jackiechan on 19-4-20/上午11:14
 * 解决springcloud F版本开始的问题
 * @Author jackiechan
 */
@Configuration //注意要加这个注解
@EnableWebSecurity//注意要加这个注解,开启安全配置
public class WebSecurityConfig extends WebSecurityConfigurerAdapter {
    @Override
    protected void configure(HttpSecurity http) throws Exception {
        super.configure(http);
        //http.csrf().disable().httpBasic(); //方式1,禁用csrf并使用http的普
        通方式,但是使用这个后,我们访问eureka会发现配置的用户密码是不需要写的,失效了

        http.csrf().ignoringAntMatchers("/eureka/**");//我们的服务注册地址是
        在/eureka下面,所以我们可以通过姜/eureka/**进行忽略安全过滤来达到允许服务注册到目的,
        而且其他地址的访问会仍旧需要密码
    }
}
```

2.5.1.4 启动Eureka Server

启动后访问 <http://localhost:10000/> 发现需要用户名和密码



← → ↻ ⓘ localhost:10000/login

Login with Username and Password

User:

Password:

Login

2.5.2 服务提供者修改

2.5.2.1 修改application.yml

我们的服务只需要修改eureka的地址风格即可

```
server:
  port: 8001
spring:
  datasource:
    username: root
    password: qishimeiyounima
    driver-class-name: com.mysql.jdbc.Driver
    url: jdbc:mysql:///db_shopping?useUnicode=true&characterEncoding=utf8
    type: com.alibaba.druid.pool.DruidDataSource

  application:
    name: 04provider-eureka

eureka: #注册中心的地址
  instance:
    prefer-ip-address: true #在 eureka 后台中访问当前项目的时候使用ip 地址访问, 默认是机器名的方式
  client:
    serviceUrl:
```

`defaultZone: http://zhangsan:abc@localhost:10000/eureka` #curl风格 意思是用户名是zhangsan密码是abc 访问的主机是localhost:10000

2.5.2.2 启动测试

启动后发现服务仍旧可以注册到eureka上, 并且访问时的信息显示为ip

The screenshot shows the Spring Eureka web interface at localhost:10000. The page has a dark header with the 'spring Eureka' logo and navigation links for 'HOME' and 'LAST 1000 SINCE STARTUP'. The main content area is divided into several sections:

- System Status:** A table showing environment details (test, default) and system metrics (Current time: 2019-07-02T11:32:36 +0800, Uptime: 00:00, Lease expiration enabled: false, Renew threshold: 3, Renew (last min): 0).
- DS Replicas:** A section showing 'localhost' as the data center.
- Instances currently registered with Eureka:** A table with columns 'Application', 'AMIs', 'Availability Zones', and 'Status'. It shows one instance: '04PROVIDER-EUREKA' with status 'UP (1) - bogon:04provider-eureka:8001'.
- General Info:** A table with columns 'Name' and 'Value'. It lists various system metrics like 'total-avail-memory' (434mb), 'environment' (test), 'num-of-cpus' (8), 'current-memory-usage' (216mb (49%)), 'server-up-time' (00:00), 'registered-replicas' (http://localhost:10000/eureka/), and 'unavailable-replicas' (http://localhost:10000/eureka/).

A red arrow points from the 'UP (1) - bogon:04provider-eureka:8001' status in the 'Instances currently registered with Eureka' table to the 'unavailable-replicas' row in the 'General Info' table.

2.6 服务消费者改进

前面的操作都是服务提供者的,现在我们开始修改服务消费者

注册到Eureka上面来说我们的服务提供者和消费者是一致的,我们需要解决的是在没有使用eureka的情况下,我们消费者访问提供者时是直接写的网址,现在我们需要通过eureka来获取提供者地址然后发起访问

除了修改controller外其他地方和服务提供者修改没区别

参考 05consumer-eureka

2.6.1 POM.xml

添加eureka的依赖用于注册服务并从eureka上面获取服务用

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

2.6.2 修改application.yml

```
server:
  port: 9000
eureka: #注册中心的地址
  instance:
    prefer-ip-address: true    #在 eureka 后台中访问当前项目的时候使用ip 地址访
问,默认是机器名的方式
  client:
    service-url:
      defaultZone: http://zhangsan:abc@localhost:10000/eureka #curl风格
spring:
  application:
    name: 05consumer-eureka
```

2.6.3 修改Controller

因为我们现在要动态获取服务提供者地址,所以我们需要通过代码获取服务提供者的最新地址

```
import com.netflix.appinfo.InstanceInfo;
import com.netflix.discovery.EurekaClient;
import com.qianfeng.microservice.consumer.pojo.User;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.web.bind.annotation.*;
import org.springframework.web.client.RestTemplate;

/**
 * Created by jackiechan on 19-6-8 下午12:00
 *
 * @Author jackiechan
 */
@RestController
@RequestMapping("/userconsumer")
public class UserController {
    @Autowired
```



```
private RestTemplate template;

@Autowired
private EurekaClient eurekaClient; //这个对象spring会自动创建不需要我们额外创建,用于帮我们连接eureka

@GetMapping("/info/{id}")
public User getUserById(@PathVariable Long id) throws Exception {

    InstanceInfo instanceInfo =
eurekaClient.getNextServerFromEureka("04PROVIDER-EUREKA", false); //从
euerka上获取指定名字的服务的实例,注意大小写
    String url = instanceInfo.getHomePageUrl(); //获取服务实例的地址 也就是 类似于 http://localhost:8000
    System.out.println("服务提供者的地址:=====>" + url);
    User user = template.getForObject(url + "/user/info/" + id,
User.class); //请求指定地址并将返回的json字符串转换为User对象
    return user;
}

@PostMapping("/save")
public String addUser(@RequestBody User user) {
    //此方法没有修改
    String result =
template.postForObject("http://localhost:8000/user/save/", user,
String.class); //post方式请求这个地址,并将user作为参数传递过去(json格式),并将返回
结果转换为String类型
    return result;
}
}
```

2.6.4 修改主程序并启动测试

启动后我们访问这个服务的接口地址,发现可以继续获取数据,说明我们的消费者从eureka上面获取到了数据并返回了

```
import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.cloud.netflix.eureka.EnableEurekaClient;
```

```
import org.springframework.context.annotation.Bean;
import org.springframework.web.client.RestTemplate;

/**
 * Created by jackiechan on 19-6-8 下午12:04
 *
 * @Author jackiechan
 */
@SpringBootApplication
@EnableEurekaClient //开启服务发现
public class ConsumerStarter {

    public static void main (String[] args){
        SpringApplication.run(ConsumerStarter.class,args);
    }

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

三 常见错误

1. Eureka Server没有启动或者是微服务中配置的Eureka注册中心地址错误

我们无法注册服务的时候基本都会报错as unable to refresh its cache! status = Cannot execute request on any known server ,因为错误会比较长,要详细看,比如当前里面还有 java.net.ConnectException: Connection refused (Connection refused)这个错误,代表我们的微服务连接的eureka server地址不对或者是没启动

```
com.sun.jersey.api.client.ClientHandlerException:
java.net.ConnectException: Connection refused (Connection refused)
    at
com.sun.jersey.client.apache4.ApacheHttpClient4Handler.handle(ApacheHttpC
lient4Handler.java:187) ~[jersey-apache-client4-1.19.1.jar:1.19.1]
    .....

2019-07-02 11:45:39.019 WARN 62122 --- [           main]
c.n.d.s.t.d.RetryableEurekaHttpClient : Request execution failed with
message: java.net.ConnectException: Connection refused (Connection
refused)
2019-07-02 11:45:39.021 ERROR 62122 --- [           main]
com.netflix.discovery.DiscoveryClient : DiscoveryClient_04PROVIDER-
EUREKA/bogon:04provider-eureka:8001 - was unable to refresh its cache!
status = Cannot execute request on any known server
    .....

com.netflix.discovery.shared.transport.TransportException: Cannot execute
request on any known server
    at
com.netflix.discovery.shared.transport.decorator.RetryableEurekaHttpClie
nt.execute(RetryableEurekaHttpClient.java:112) ~[eureka-client-
1.9.3.jar:1.9.3]
```

2. 密码问题

当我们的Eureka服务端设置了密码 但是没有将csrf关闭或者是忽略eureka的注册地址的时候会出现无法注册,但是错误提示和上面基本一样, 都是 Cannot execute request on any known server 但是可能会出现一个Eureka javax.ws.rs.WebApplicationException: null ,这个时候基本上就是我们没有关闭csrf,通过上面的密码保护部分来关闭即可

```
2019-07-02 11:52:53.655 WARN 62155 --- [nfoReplicator-0]
c.n.discovery.InstanceInfoReplicator : There was a problem with the
instance info replicator

com.netflix.discovery.shared.transport.TransportException: Cannot execute
request on any known server
    at
com.netflix.discovery.shared.transport.decorator.RetryableEurekaHttpClie
nt.execute(RetryableEurekaHttpClient.java:112) ~[eureka-client-
1.9.3.jar:1.9.3]
    at
com.netflix.discovery.shared.transport.decorator.EurekaHttpClientDecorato
r.register(EurekaHttpClientDecorator.java:56) ~[eureka-client-
1.9.3.jar:1.9.3]
    at
com.netflix.discovery.shared.transport.decorator.EurekaHttpClientDecorato
r$1.execute(EurekaHttpClientDecorator.java:59) ~[eureka-client-
1.9.3.jar:1.9.3]
    at
com.netflix.discovery.shared.transport.decorator.SessionedEurekaHttpClie
nt.execute(SessionedEurekaHttpClient.java:77) ~[eureka-client-
1.9.3.jar:1.9.3]
    at
com.netflix.discovery.shared.transport.decorator.EurekaHttpClientDecorato
r.register(EurekaHttpClientDecorator.java:56) ~[eureka-client-
1.9.3.jar:1.9.3]
```

四 Eureka几处坑(了解)

4.1 Eureka的几处缓存

Eureka的wiki上有一句话，大意是一个服务启动后最长可能需要2分钟时间才能被其它服务感知到，但是文档并没有解释为什么会有这2分钟。其实这是由三处缓存 + 一处延迟造成的。

首先，Eureka对HTTP响应做了缓存。在Eureka的“控制器”类 `ApplicationResource` 的 109行可以看到有一行

```
String payLoad = responseCache.get(cacheKey);11
```

的调用，该代码所在的 `getApplication()` 方法的功能是响应客户端查询某个服务信息的HTTP请求：

```
String payLoad = responseCache.get(cacheKey); // 从cache中拿响应数据
if (payLoad != null) {
    logger.debug("Found: {}", appName);
    return Response.ok(payLoad).build();
} else {
    logger.debug("Not Found: {}", appName);
    return Response.status(Status.NOT_FOUND).build();
}
```

上面的代码中，`responseCache`引用的是`ResponseCache`类型，该类型是一个接口，其`get()`方法首先会去缓存中查询数据，如果没有则生成数据返回（即真正去查询注册列表），且缓存的有效时间为30s。也就是说，客户端拿到Eureka的响应并不一定是即时的，大部分时候只是缓存信息。

其次，Eureka Client对已经获取到的注册信息也做了30s缓存。即服务通过eureka客户端第一次查询到可用服务地址后会将结果缓存，下次再调用时就不会真正向Eureka发起HTTP请求了。

再次，负载均衡组件Ribbon也有30s缓存。Ribbon会从上面提到的Eureka Client获取服务列表，然后将结果缓存30s。

最后，如果你并不是在Spring Cloud环境下使用这些组件(Eureka, Ribbon)，你的服务启动后并不会马上向Eureka注册，而是需要等到第一次发送心跳请求时才会注册。心跳请求的发送间隔也是30s。（Spring Cloud对此做了修改，服务启动后会马上注册）

以上这四个30秒正是官方wiki上写服务注册最长需要2分钟的原因。

4.2 服务注册信息不会被二次传播

如果在集群环境下Eureka A的peer指向了B, B的peer指向了C，那么当服务向Eureka A注册时，Eureka B中会有该服务的注册信息，但是Eureka C中没有。也就是说，如果你希望只要向一台Eureka注册然后其它所有Eureka实例都能得到注册信息，那么就必须把其它所有节点都配置到当前Eureka的 `peer` 属性中。这一逻辑是

在 `PeerAwareInstanceRegistryImpl#replicateToPeers()` 方法中实现的：

```
private void replicateToPeers(Action action, String appName, String id,
                              InstanceInfo info /* optional */,
                              InstanceStatus newStatus /* optional
*/, boolean isReplication) {
    Stopwatch tracer = action.getTimer().start();
    try {
```

```
        if (isReplication) {
            numberOfReplicationsLastMin.increment();
        }
        // 如果这条注册信息是其它Eureka同步过的则不会继续传播给自己的peer节点
        if (peerEurekaNodes == Collections.EMPTY_LIST ||
isReplication) {
            return;
        }
        for (final PeerEurekaNode node :
peerEurekaNodes.getPeerEurekaNodes()) {
            // 不要向自己发同步请求
            if (peerEurekaNodes.isThisMyUrl(node.getServiceUrl())) {
                continue;
            }
            replicateInstanceActionsToPeers(action, appName, id,
info, newStatus, node);
        }
    } finally {
        tracer.stop();
    }
}
```

4.3多网卡环境下的IP选择问题

如果机器上(客户端 服务端)安装了多块网卡，它们分别对应IP地址A, B, C，此时：Eureka 会选择IP合法(标准ipv4地址)、索引值最小(eth0, eth1中eth0优先)且不在忽略列表中(可在 `application.properties` 中配置忽略哪些网卡)的网卡地址作为服务IP。

我们的服务在注册到Eureka的时候,在Eureka管理页面可能会发现,服务的ip并不是我们真正的外部ip,这种情况一般发生在电脑上面装了虚拟机的时候产生了多个网卡

这个坑的详细分析见：<http://blog.csdn.net/neosmith/article/details/53126924>

三 Eureka 集群

Eureka为了保证高可用,需要搭建集群,Eureka的集群搭建很简单,只要eureka服务端之间相互注册就行了,每个Eureka作为其他的Eureka的客户端,配置方式和我们配置普通服务没区别

#第一个Eureka的配置文件

```
spring:
  application:
    name: EUREKA-HA
server:
  port: 8761
eureka:
  instance:
    hostname: localhost
  client:
    serviceUrl:
      defaultZone:
http://localhost:8762/eureka/,http://localhost:8763/eureka/ ##配置另外两个
的eureka的地址,主要我们之前单机的时候eureka指定为单机版,此处需要删除
```

#第二个的配置文件

```
server:
  port: 8762
spring:
  application:
    name: EUREKA-HA
eureka:
  instance:
    hostname: localhost
  client:
    serviceUrl:
      defaultZone:
http://localhost:8761/eureka/,http://localhost:8763/eureka/ ##配置另外两个
的eureka的地址,主要我们之前单机的时候eureka指定为单机版,此处需要删除
```

#第三个eureka的配置文件

```
server:
  port: 8763
spring:
  application:
    name: EUREKA-HA
eureka:
  instance:
    hostname: localhost
  client:
    serviceUrl:
```

defaultZone:

http://localhost:8761/eureka/, http://localhost:8762/eureka/ ##配置另外两个
的eureka的地址, 主要我们之前单机的时候eureka指定为单机版, 此处需要删除