Spring Cloud Eureka

首先，我们来尝试使用Spring Cloud Eureka来实现服务治理。

Spring Cloud Eureka是Spring Cloud Netflix项目下的服务治理模块。而Spring Cloud Netflix项目是Spring Cloud的子项目之一，主要内容是对Netflix公司一系列开源产品的包装，它为Spring Boot应用提供了自配置的Netflix OSS整合。通过一些简单的注解，开发者就可以快速的在应用中配置一下常用模块并构建庞大的分布式系统。它主要提供的模块包括：服务发现（Eureka），断路器（Hystrix），智能路由（Zuul），客户端负载均衡（Ribbon）等。

服务发现是基于微服务的体系结构的关键原则之一。试图手动配置每个客户端或某种形式的约定可能非常困难，而且非常脆弱。Eureka是Netflix服务发现服务器和客户端。可以将服务器配置和部署为高度可用，每个服务器将注册服务的状态复制到其他服务器。

服务注册

当客户端向Eureka注册时，它会提供关于自身的元数据，例如主机和端口、健康指示器URL、主页等。Eureka会从属于某个服务的每个实例接收心跳消息。如果心跳在可配置的时间表上失败，实例通常从注册表中删除。

下面，就来具体看看如何使用Spring Cloud Eureka实现服务治理。

创建“服务注册中心”

首先要创建一个基础的Spring Boot工程，命名为eureka-server，并在pom.xml中引入需要的依赖内容：

```
<parent>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-parent</artifactId>
    <version>2.1.1.RELEASE</version>
</parent>
<dependencies>
    <!--增加eureka-server的依赖-->
    <dependency>
        <groupId>org.springframework.cloud</groupId>
        <artifactId>spring-cloud-starter-netflix-eureka-server</artifactId>
    </dependency>
</dependencies>
<!--依赖管理，用于管理spring-cloud的依赖-->
<dependencyManagement>
    <dependencies>
        <dependency>
            <groupId>org.springframework.cloud</groupId>
            <artifactId>spring-cloud-dependencies</artifactId>
            <version>Greenwich.RELEASE</version>
            <type>pom</type>
            <scope>import</scope>
        </dependency>
    </dependencies>

</dependencyManagement>
```

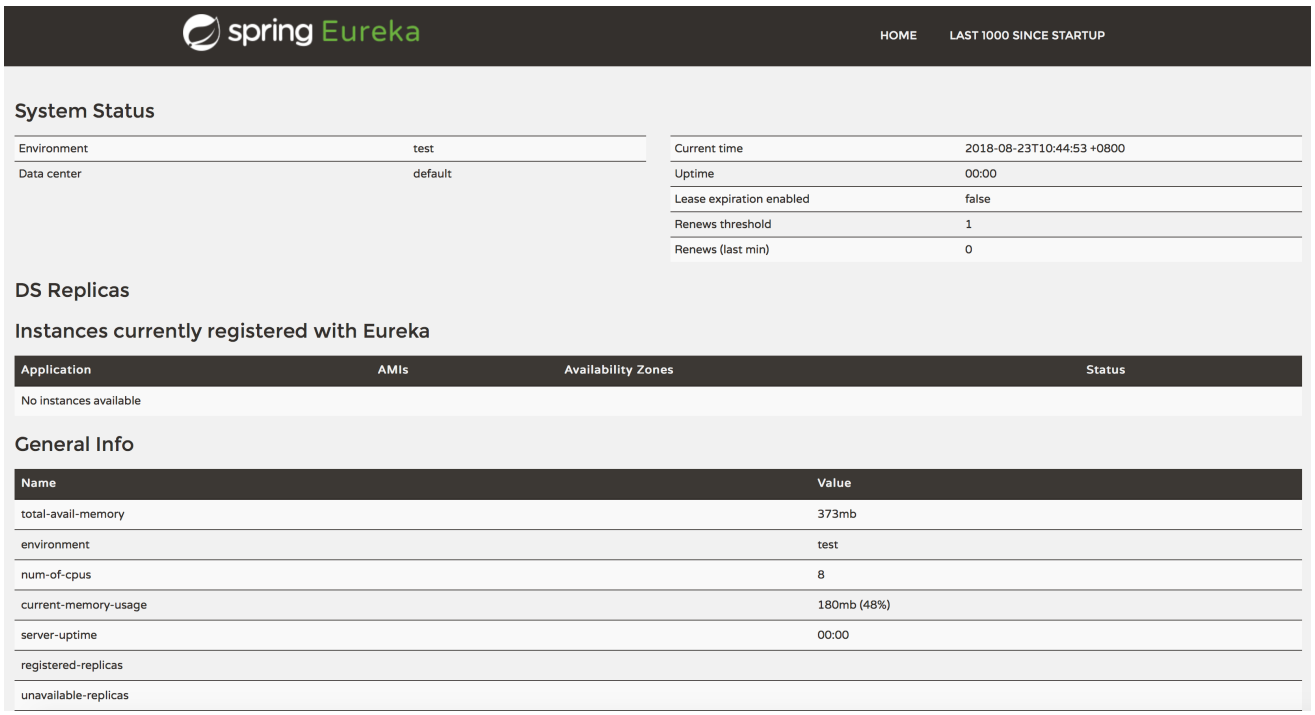
通过@EnableEurekaServer注解启动一个服务注册中心提供给其他应用进行对话。这一步非常的简单，只需要在一个普通的Spring Boot应用中添加这个注解就能开启此功能，比如下面的例子：

```
@SpringBootApplication//注解等价于以默认属性使用 @Configuration ，@EnableAutoConfiguration 和 @ComponentScan
@EnableEurekaServer//启动一个服务注册中心提供给其他应用进行对话
public class Application {
    public static void main(String[] args) {
        System.out.println("this is eureka-server for Greenwich!");
        SpringApplication app = new SpringApplication(Application.class);
        app.run(args);
    }
}
```

在默认设置下，该服务注册中心也会将自己作为客户端来尝试注册它自己，所以我们需要禁用它的客户端注册行为，只需要在application.properties配置文件中增加如下信息：

```
#指定微服务名称
spring.application.name=eureka-server
#指定端口号
server.port=9401
#服务注册中心实例的主机名
eureka.instance.hostname=localhost
#是否向服务注册中心注册自己  true将自身注册  false自身不注册
eureka.client.register-with-eureka=false
#是否检索服务
eureka.client.fetch-registry=false
```

为了与后续要进行注册的服务区分，这里将服务注册中心的端口通过server.port属性设置为9401。启动工程后，访问：http://localhost:9401/ ，可以看到下面的页面，其中还没有发现任何服务。



创建“服务提供方”（一）

下面我们创建提供服务的客户端，并向服务注册中心注册自己。本文我们主要介绍服务的注册与发现，所以我们不妨在服务提供方中尝试着提供一个接口来获取当前所有的服务信息。

首先，创建一个基本的Spring Boot应用。命名为eureka-client，在pom.xml中，加入如下配置：

```
<parent>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-parent</artifactId>
    <version>2.1.1.RELEASE</version>
    <relativePath /> <!-- lookup parent from repository -->
</parent>
<dependencies>
    <!-- Eureka 客户端依赖 -->
    <dependency>
        <groupId>org.springframework.cloud</groupId>
        <artifactId>spring-cloud-starter-netflix-eureka-server</artifactId>
    </dependency>
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-web</artifactId>
    </dependency>
</dependencies>
<!--依赖管理，用于管理spring-cloud的依赖 -->
<dependencyManagement>
    <dependencies>
        <dependency>
            <groupId>org.springframework.cloud</groupId>
            <artifactId>spring-cloud-dependencies</artifactId>
            <version>Greenwich.RELEASE</version>
            <type>pom</type>
            <scope>import</scope>
        </dependency>
    </dependencies>
</dependencyManagement>
```

其次，实现/dc请求处理接口，通过DiscoveryClient对象，在日志中打印出服务实例的相关内容。

```
@RestController
public class DcController {
    @Autowired
    DiscoveryClient discoveryClient;
    /**
     *
     * @Description 实现/dc请求处理接口，通过DiscoveryClient对象，在日志中打印出服务实例的相关内容
     * @Author ｃｈｉｘｕｅ
     * @CreatDate 2018年7月20日
     * @UpdateUser 方法更新者
     * @UpdateDate 2018年7月20日

     * @throws
```

```
     * @return
     * @throws InterruptedException
     * @return  String
     */
    //使用slf4j的Logger类记录日志，此条日志相关的追踪信息会由Sleuth自动生成并记录。
    private static Logger log=LoggerFactory.getLogger(DcController.class);
    @GetMapping("/dc")
    public String dc() throws InterruptedException {
        String services = "Services: " + discoveryClient.getServices();
        //此处进行日志追踪
        log.info(services);
        return services;
    }
}
```

最后在应用主类中通过加上@EnableDiscoveryClient注解，该注解能激活Eureka中的DiscoveryClient实现，这样才能实现Controller中对服务信息的输出。

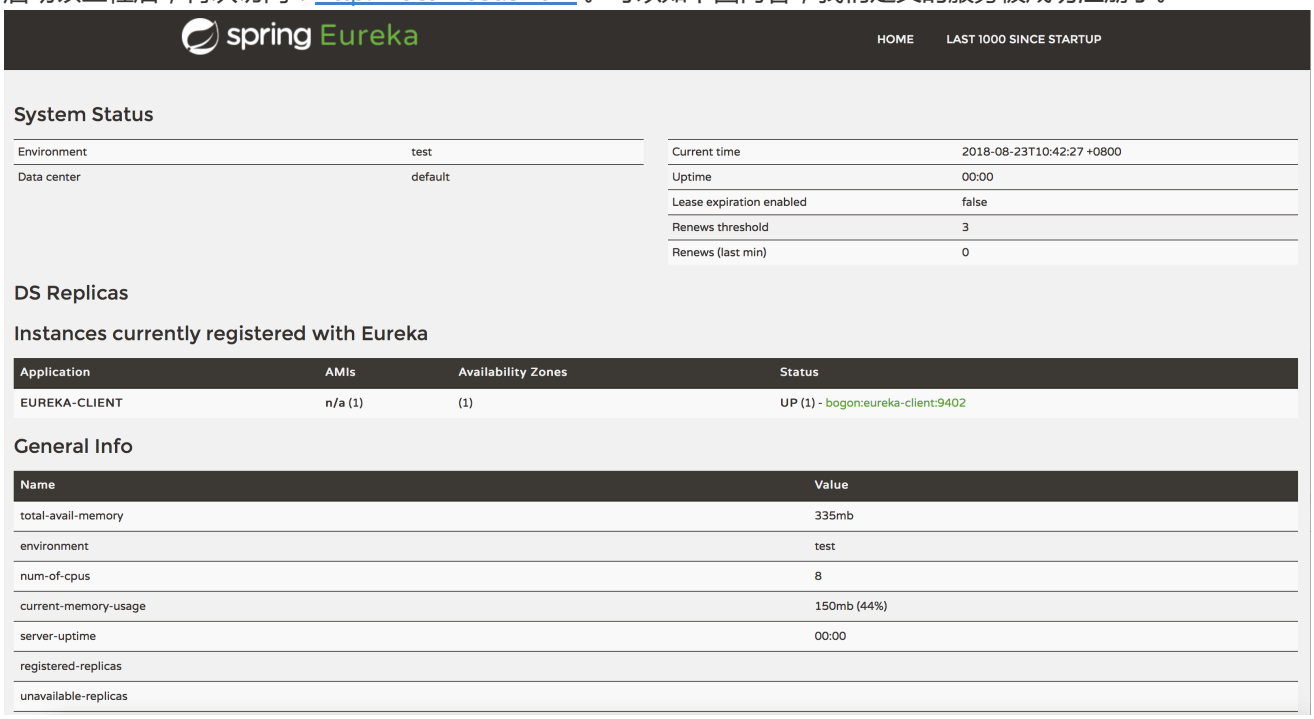
```
@SpringBootApplication//注解等价于以默认属性使用 @Configuration ，@EnableAutoConfiguration 和 @ComponentScan
//@EnableDiscoveryClient//激活eureka中的DiscoveryClient实现 该注解会根据配置文件中的地址，将服务自身注册到服务注册中心
@EnableEurekaClient //与EnableDiscoveryClient差不多，如果选用的注册中心是eureka，那么就推荐@EnableEurekaClient，如果是其他的注册中心，那么推荐使用@EnableDiscoveryClientpublic class Application {
    public static void main(String[] args) {
      System.out.println("this is eureka-client!");
    SpringApplication app = new SpringApplication(Application.class);
    app.run(args);
  }
}
```

我们在完成了服务内容的实现之后，再继续对application.properties做一些配置工作，具体如下：

```
#指定微服务名称spring.application.name=eureka-client#指定端口号server.port=9402#指定eureka注册中心的地址eureka.client.serviceUrl.defaultZone=http://localhost:9401/eureka/
```

通过spring.application.name属性，我们可以指定微服务的名称后续在调用的时候只需要使用该名称就可以进行服务的访问。eureka.client.serviceUrl.defaultZone属性对应服务注册中心的配置内容，指定服务注册中心的位置。为了在本机上测试区分服务提供方和服务注册中心，使用server.port属性设置不同的端口。

启动该工程后，再次访问：http://localhost:9401/ 。可以如下图内容，我们定义的服务被成功注册了。



当然，我们也可以通过直接访问eureka-client服务提供的/dc接口来获取当前的服务清单，只需要访问：http://localhost:9402/dc ， 我们可以得到如下输出返回：

```
Services: [eureka-client]
```

其中，方括号中的eureka-client就是通过Spring Cloud定义的DiscoveryClient接口在eureka的实现中获取到的所有服务清单。由于Spring Cloud在服务发现这一层做了非常好的抽象，所以，对于上面的程序，我们可以无缝的从eureka的服务治理体系切换到consul的服务治理体系中区。

创建“服务提供方”（二）

与第一种创建方法相比，根据官网上的介绍，可以把spring-cloud-starter-eureka换成spring-cloud-starter-netflix-eureka-client，并且主类上也无需引用@EnableDiscoveryClient注解（By having spring-cloud-starter-netflix-eureka-client on the classpath your application will automatically register with the Eureka Server. Configuration is required to locate the Eureka server）。其他位置不变。

eureka高可用

Eureka Server 的高可用实际上就是将自己作为服务想其它服务注册中心注册自己，这样就形成了一组互相注册的服务中心，以实现服务清单 的互相同步，达到高可用的效果。

我用同一个项目打算启动两个server服务，占用不同的端口，以此模拟eureka服务集群。

配置文件修改如下：

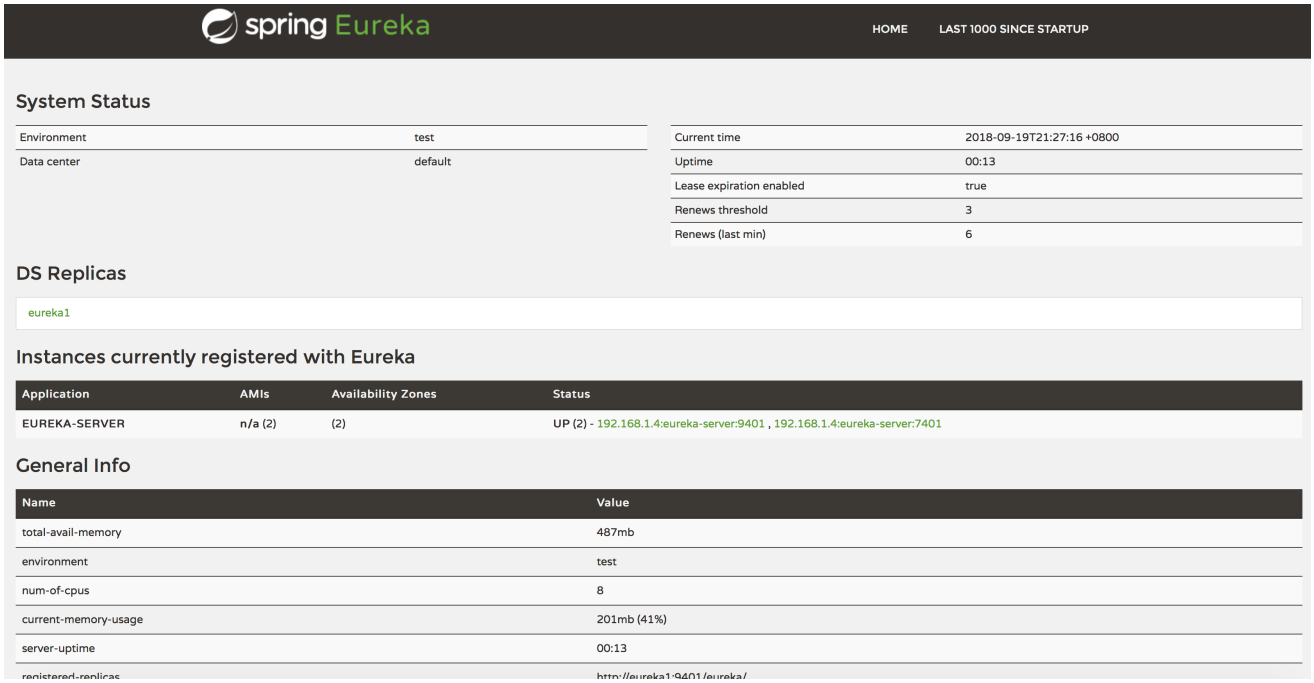
```
#指定微服务名称
spring.application.name=eureka-server
#指定端口号
server.port=7401
#服务注册中心实例的主机名
#eureka.instance.hostname=localhost
eureka.instance.hostname=eureka2
#是否向服务注册中心注册自己  true将自身注册  false自身不注册
#eureka.client.register-with-eureka=false
#是否检索服务  false是因为服务注册中心本身的职责就是维护服务实例，它也不需要去检索其他服务
#eureka.client.fetch-registry=false
# eureka高可用
eureka.client.serviceUrl.defaultZone=http://eureka1:9401/eureka/
```

启动，然后修改配置文件：

```
#指定微服务名称
spring.application.name=eureka-server
#指定端口号
server.port=9401
#服务注册中心实例的主机名
#eureka.instance.hostname=localhost
eureka.instance.hostname=eureka1
#是否向服务注册中心注册自己  true将自身注册  false自身不注册
#eureka.client.register-with-eureka=false
#是否检索服务  false是因为服务注册中心本身的职责就是维护服务实例，它也不需要去检索其他服务
#eureka.client.fetch-registry=false
# eureka高可用
eureka.client.serviceUrl.defaultZone=http://eureka2:7401/eureka/#指定微服务名称
spring.application.name=eureka-server#指定端口号server.port=9401#服务注册中心实例的主机名
#eureka.instance.hostname=localhosteureka.instance.hostname=eureka1#是否向服务注册中心注册自己
true将自身注册  false自身不注册#eureka.client.register-with-eureka=false#是否检索服务  false是因为服
务注册中心本身的职责就是维护服务实例，它也不需要去检索其他服务#eureka.client.fetch-registry=false#
eureka高可用eureka.client.serviceUrl.defaultZone=http://eureka2:7401/eureka/
```

```
指定不同的端口，并且service-url这里是重点，我将eureka1的service-url设置为eureka2，将eureka2的设置为
eureka1.以此完成两个eureka-server服务间的相互注册。
instance.hostname是唯一标识。
由于我们使用了http://server1这种写法，需要配一下host。Windows的host在/etc/host，mac的在/private/etc
增加如下内容
127.0.0.1 eureka1
127.0.0.1 eureka2
```

启动，分别访问http://localhost:9401 和http://localhost:7401 ，如图：



这时修改eureka-client的配置文件，指定其中一个server地址，如：

```
eureka.client.serviceUrl.defaultZone=http://eureka1:9401/eureka/
```

启动eureka-client，分别访问分别访问http://localhost:9401 和http://localhost:7401 ，会发现eureka-client在两个server上都注册了。虽然我们在client的yml里default_zone只配置了eureka1的server。这是因为eureka是通过在各个节点进行复制来达到高可用的目的。

这时停掉一个eureka-server的服务，然后分别访问eureka-server的地址，查看下是否实现了eureka-server的高可用。

目前公司已有eureka注册与发现

已有环境

开发环境:http://eureka-server-fnw-dev.topaas.enncloud.cn

测试环境:http://eureka-server-fnw-test.topaas.enncloud.cn

预生产环境:http://eureka-01-fnw-prod.topaas.enncloud.cn

生产环境:http://eureka-01-fnw-release.topaas.enncloud.cn/eureka,http://eureka-02-fnw-release.topaas.enncloud.cn/eureka,http://eureka-03-fnw-release.topaas.enncloud.cn/eureka

客户端注册与发现：

添加pom 依赖

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

开启注解

启动类添加@EnableDiscoveryClient或者@EnableEurekaClient

修改配置

服务提供者：

```
# server
server.port=8080
# 客户端在注册时就会使用自己的ip地址而不是主机名(客户端自身加)
eureka.instance.preferIpAddress=true
eureka.client.registerWithEureka=true
#是否需要去检索寻找服务，默认是true
eureka.client.fetchRegistry=true
#renew频率，向Eureka服务发送renew信息，默认30秒
eureka.instance.leaseRenewalIntervalInSeconds=10

# 心跳时间，即服务续约间隔时间（缺省为30s）
#eureka.instance.lease-renewal-interval-in-seconds: 5
# 发呆时间，即服务续约到期时间（缺省为90s）
#eureka.instance.lease-expiration-duration-in-seconds: 10
# 开启健康检查（依赖spring-boot-starter-actuator）
#eureka.client.healthcheck.enabled: true

# 提供者服务名
spring.application.name=service-test
# eureka注册地址 配置对应环境的注册地址
eureka.client.serviceUrl.defaultZone=http://eureka-01-fnw-
release.topaas.enncloud.cn/eureka,http://eureka-02-fnw-
release.topaas.enncloud.cn/eureka,http://eureka-03-fnw-release.topaas.enncloud.cn/eureka
```

服务消费者（一般需要连接其他微服务的服务或者gateway/zuul）

```
# server
server.port=8080
# 消费者服务名
spring.application.name=zuul
#client隔多久去拉取服务注册信息，默认为30秒，zuul需要将该时间缩短以便迅速获取其他服务信息
eureka.client.registry-fetch-interval-seconds=5
# eureka注册地址 配置对应环境的注册地址
eureka.client.serviceUrl.defaultZone=http://eureka-01-fnw-
release.topaas.enncloud.cn/eureka,http://eureka-02-fnw-
release.topaas.enncloud.cn/eureka,http://eureka-03-fnw-release.topaas.enncloud.cn/eureka
```