一.简介

传统的三层架构后来在互联网公司让几百人几千人同时开发一个项目已经变得不可行，并且会产生代码冲突的问题。基于 SOA 面向服务开发的架构，渐渐产生了微服务架构。微服务的架构的特点就是项目拆分成各个子项目，进行解耦操作，提供外部访问接口，属于敏捷开发，其实也可以视为面向接口开发。(我们海康的新体系也是采用的也是微服务的思想，把一个整体拆分成一个个小的组件来单独对外提供服务.)

一旦有了多个子项目，就回产生如何管理接口、负载均衡、高并发情况下怎么限流断路等问题。那么这就有 SpringCloud 出现了。

那么 springCloud 的组件大概有哪些呢,如下：

(1)Eureka　服务注册中心
(2)服务消费者 Rest 和 Fegin　--消费实现负载均衡 ribbon
(3)接口网关 Zuul
(4)Hystrix -服务熔断、服务降级、隔离资源

二.Eureka　服务注册中心

eureka 是一个服务注册中心,是一个高可用的组件，它没有后端缓存，每一个实例注册之后需要向注册中心发送心跳（因此可以在内存中完成），在默认情况下 erureka server 也是一个 eureka client ，通过 eureka.client.registerWithEureka：false 和 fetchRegistry：false 来表明自己是一个 eureka serve

一个 springcloud 首先需要一个 eureka,子服务在 eureka 里面注册,才能被其他自服务发现和调用.

创建 eureka 注册中心，这里建议用 idea 的工具创建 SpringBoot 项目

pom 如下

```
<?xml version="1.0" encoding="UTF-8"?>
<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
https://maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>
    <parent>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-parent</artifactId>
        <version>2.1.6.RELEASE</version>
        <relativePath/> <!-- lookup parent from repository -->
    </parent>
    <groupId>com.gua</groupId>
    <artifactId>cloud</artifactId>
    <version>0.0.1-SNAPSHOT</version>
    <name>cloud</name>
```

```
    <description>Demo project for Spring Boot</description>

    <properties>
        <java.version>1.8</java.version>
        <spring-cloud.version>Finchley.RELEASE</spring-cloud.version>
    </properties>

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

    <dependencyManagement>
        <dependencies>
            <dependency>
                <groupId>org.springframework.cloud</groupId>
                <artifactId>spring-cloud-dependencies</artifactId>
                <version>${spring-cloud.version}</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>
</project>
```

yam 如下

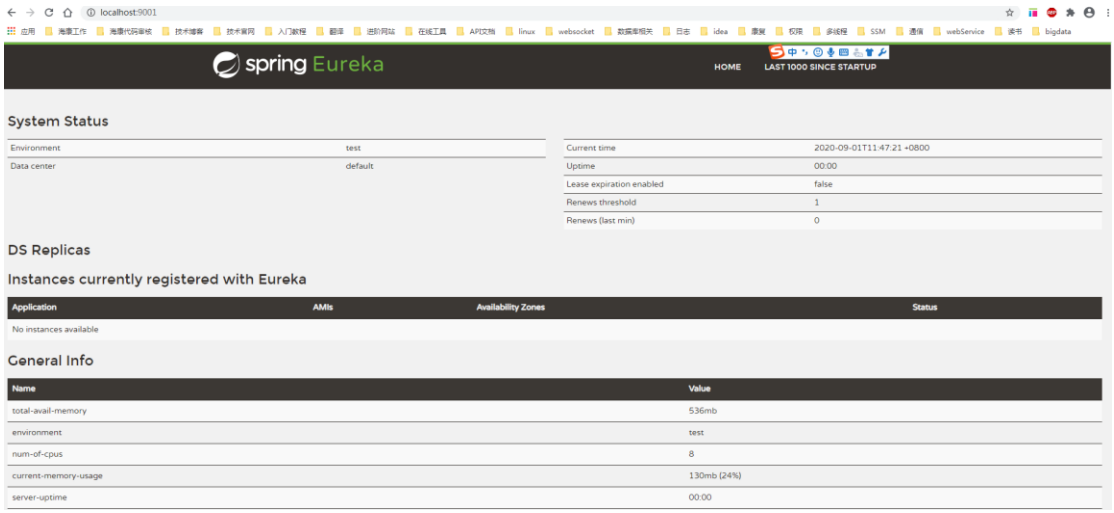
```
server:
  port: 9000   #指定服务端口号
```

```
eureka:
  instance:
    hostname: serviceCenter   #指定服务名称
  client:
    register-with-eureka: false   #是否需要将自己注册到注册中心，因为该工程自己就是服务注册中心，所以无需注册
    fetch-registry: false          #是否向注册中心定时更新自己状态
    service-url:
      defaultZone: http://${eureka.instance.hostname}:${server.port}/eureka/
```

然后在启动类里添加表示为 eureka 注册中心的注解

```
@EnableEurekaServer
@SpringBootApplication
public class LearningApplication {

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

}
```

通过 localhost:9000 可以进到如下页面就表示 eureka 注册中心启动成功



新建一个服务,并注册到 eureka 中

pom 如下

```
<?xml version="1.0" encoding="UTF-8"?>
<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
https://maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>
```

```
<parent>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-parent</artifactId>
    <version>2.1.6.RELEASE</version>
    <relativePath/> <!-- lookup parent from repository -->
</parent>
<groupId>com.gua</groupId>
<artifactId>cloudService1</artifactId>
<version>0.0.1-SNAPSHOT</version>
<name>cloud</name>
<description>Demo project for Spring Boot</description>

<properties>
    <java.version>1.8</java.version>
    <spring-cloud.version>Finchley.RELEASE</spring-cloud.version>
</properties>

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

<dependencyManagement>
    <dependencies>
        <dependency>
            <groupId>org.springframework.cloud</groupId>
            <artifactId>spring-cloud-dependencies</artifactId>
            <version>${spring-cloud.version}</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>
</project>
```

yam 如下

```
server:
   port: 9002                    #指定服务端口号
   registrycentery: 9001     #服务注册中心端口号

spring:
   application:
      name: cloudService1   #服务名，服务名是唯一的，不同的服务名字不能重复

eureka:
   client:
      register-with-eureka: true       #是否需要将自己注册到注册中心
      fetch-registry: true               #是否向注册中心定时更新自己状态
      service-url:
         defaultZone: http://localhost:${server.registrycentery}/eureka/        #指定去哪个服务注册中心进行注册
```

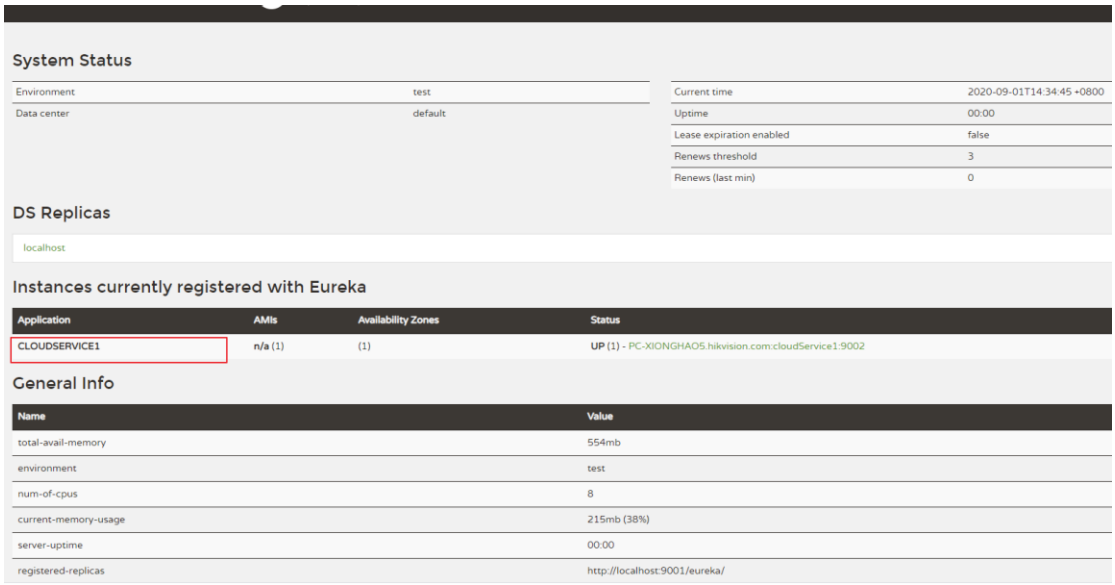
启动类如下

```
@SpringBootApplication
@EnableEurekaClient
public class CloudService1Application {

      public static void main(String[] args) {
            SpringApplication.run(CloudService1Application.class, args);
      }

}
```

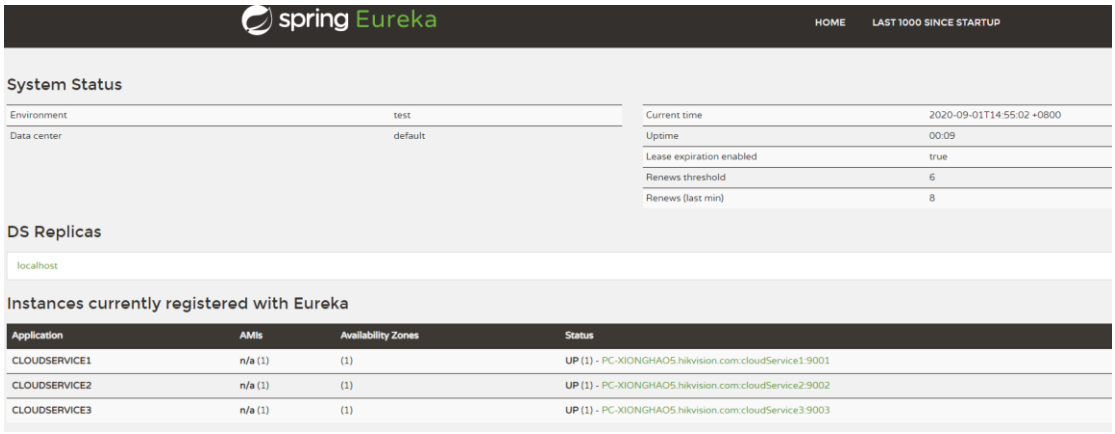
对外提供接口

```
@RestController
public class HomeController {

      @GetMapping("/service1")
      public String service1() {
            return "this is service1";
      }
}
```

在 eurake 中查看已经注册的服务



同理,再创建 2 个服务 cloudService2 和 cloudService3.这样 eurake 中就注册了 3 个服务了.



三.服务调用

rest 调用

启动类代码

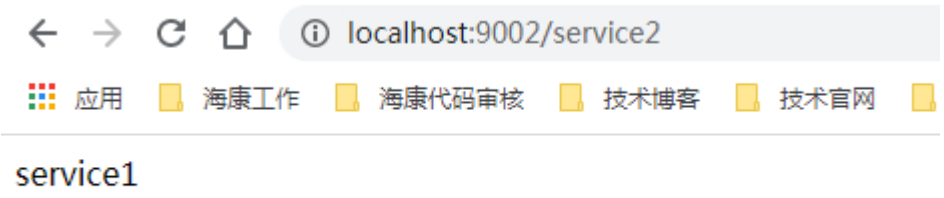
```
@SpringBootApplication
@EnableEurekaClient
public class CloudService2Application {

    public static void main(String[] args) {
        SpringApplication.run(CloudService2Application.class, args);
    }
```

```
    @Bean
    @LoadBalanced            // 开启负载均衡
    public RestTemplate restTemplate() {
        return new RestTemplate();
    }
}
```

服务调用

```
@RestController
public class HomeController {

    @Autowired
    private RestTemplate restTemplate;

    @GetMapping("/service2")
    public String service1() {
        return restTemplate.getForObject("http://cloudService1/service1",String.class);
    }
}
```

调用结果



四.路由网关

cloudService1 和 cloudService2 已经注册进服务中心，两者之间是通过网址直接访问。但是如果在浏览器里,会因为域名不同而导致跨域问题。跨域问题的解决方案可以使用 http client 设置、设置请求头、nginx 转发解决，那么在 SpringCloud 里面当然提供了一套解决方案，那就是网关 ZUUL。

pom 如下

```
<?xml version="1.0" encoding="UTF-8"?>
<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
https://maven.apache.org/xsd/maven-4.0.0.xsd">
        <modelVersion>4.0.0</modelVersion>
        <parent>
```

```
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-parent</artifactId>
        <version>2.3.3.RELEASE</version>
        <relativePath/> <!-- lookup parent from repository -->
    </parent>
    <groupId>com.gua</groupId>
    <artifactId>cloudService3</artifactId>
    <version>0.0.1-SNAPSHOT</version>
    <name>cloud</name>
    <description>Demo project for Spring Boot</description>

    <properties>
        <java.version>1.8</java.version>
        <spring-cloud.version>Hoxton.SR8</spring-cloud.version>
    </properties>

    <dependencies>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-web</artifactId>
        </dependency>
        <dependency>
            <groupId>org.springframework.cloud</groupId>
            <artifactId>spring-cloud-starter-netflix-eureka-client</artifactId>
        </dependency>
        <dependency>
            <groupId>org.springframework.cloud</groupId>
            <artifactId>spring-cloud-starter-netflix-zuul</artifactId>
        </dependency>

    </dependencies>

    <dependencyManagement>
        <dependencies>
            <dependency>
                <groupId>org.springframework.cloud</groupId>
                <artifactId>spring-cloud-dependencies</artifactId>
                <version>${spring-cloud.version}</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>
    </project>
```

yam 如下

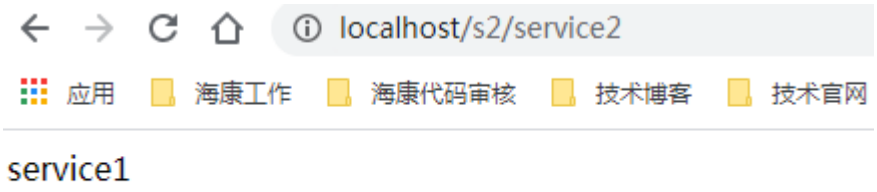
```
server:
  port: 80                    #指定服务端口号
  registrycentery: 9000     #服务注册中心端口号

spring:
  application:
    name: zuulService   #服务名，服务名是唯一的，不同的服务名字不能重复

eureka:
  client:
    register-with-eureka: true       #是否需要将自己注册到注册中心
    fetch-registry: true             #是否向注册中心定时更新自己状态
    service-url:
      defaultZone: http://localhost:${server.registrycentery}/eureka/        # 指 定 去 哪
个服务注册中心进行注册

#配置网关转发详情
zuul:
  routes:
    api-a:
      path: /s1/**
      service-id: cloudService1
    api-b:
      path: /s2/**
      service-id: cloudService2
```

启动代码如下

```
@SpringBootApplication
@EnableEurekaClient
@EnableZuulProxy
public class CloudService3Application {
```

```
    public static void main(String[] args) {
        SpringApplication.run(CloudService3Application.class, args);
    }

}
```

调用如下
转发已经启动



五断路器（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 的作用

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.

给 cloudService2 的 pom 加入如下依赖

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

代码

```
@RestController
public class HomeController {

    @Autowired
    private RestTemplate restTemplate;

    @GetMapping("/service2")
    @HystrixCommand(fallbackMethod = "testError")
    public String service1() {
        return restTemplate.getForObject("http://cloudService1/service1",String.class);
    }

    public String testError(){
        return "服务维护中...";
    }
}
```

启动类中加入熔断注解

```
@SpringBootApplication
@EnableEurekaClient
@EnableHystrix
@EnableCircuitBreaker
public class CloudService2Application {

    public static void main(String[] args) {
        SpringApplication.run(CloudService2Application.class, args);
    }
```

```
    @Bean
    @LoadBalanced              // 开启负载均衡
    public RestTemplate restTemplate() {
        return new RestTemplate();
    }
}
```

把 cloudService1 停掉,调用 cloudService2 的接口,如下,熔断已经发生.