

## 1. Eureka基础知识

### 1.1. 什么是服务治理

### 1.2. 什么是服务注册

### 1.3. Eureka两组件

## 2. 单机Eureka构建步骤

### 2.1. IDEA生成EurekaServer端服务注册中心, 类似物业公司

#### 2.1.1. 建Module

#### 2.1.2. 改POM

#### 2.1.3. 写YML

#### 2.1.4. 主启动

#### 2.1.5. 测试

### 2.2. EurekaClient端cloud-provider-payment8001

#### 2.2.1. 修改POM

#### 2.2.2. 修改主配置

#### 2.2.3. 修改启动类

### 2.3. EurekaClient端cloud-consumer-order80

#### 2.3.1. 修改POM

#### 2.3.2. 修改主配置

#### 2.3.3. 修改启动类

### 2.4. 测试

### 2.5. 自我保护机制

## 3. 集群Eureka构建步骤

### 3.1. Eureka集群原理说明

### 3.2. Eureka集群环境构建步骤

#### 3.2.1. 参考cloud-eureka-server7001新建cloud-eureka-server7002模块

#### 3.2.2. 改POM

#### 3.2.3. 修改映射配置

#### 3.2.4. 写YML

##### 3.2.4.1. 7001以前的配置(单机)

##### 3.2.4.2. 7001现在的配置(集群)

##### 3.2.4.3. 7002现在的配置(集群)

#### 3.2.5. 主启动

### 3.3. 将支付服务8001微服务发布到上面2台Eureka集群配置中

### 3.4. 将订单服务80微服务发布到上面2台Eureka集群配置中

### 3.5. 测试01

### 3.6. 支付服务提供者8001集群环境搭建

3.6.1. 参考cloud-provider-payment8001新建cloud-provider-payment8002

3.6.2. 改POM

3.6.3. 改主配置

3.6.4. 主启动

3.6.5. 业务类

3.6.6. 修改8001/8002的controller

3.7. 负载均衡

3.7.1. bug => 订单服务访问地址存在硬编码,无法进行负载均衡

3.7.2. 使用@LoadBalanced注解赋予RestTemplate负载均衡的能力

3.8. 测试02

4. actuator微服务信息完善

4.1. 主机名称:服务名称修改

4.1.1. 当前问题

4.1.2. 修改cloud-provoder-payment8001/cloud-provoder-payment8002

4.1.3. 修改之后

4.2. 访问信息有IP信息提示

4.2.1. 当前问题

4.2.2. 修改cloud-provoder-payment8001/cloud-provoder-payment8002

4.2.3. 修改之后

5. 服务发现Discovery

5.1. 修改cloud-provider-payment8001/cloud-provider-payment8002模块

5.1.1. 修改controller

5.1.2. 修改启动类

5.2. 测试

6. Eureka自我保护

6.1. 故障现象

6.2. 问题原因

6.3. 怎样禁止自我保护

6.3.1. 注册中心eurekaServer端7001

6.3.1.1. 默认配置

6.3.1.2. 修改配置

6.3.1.3. 测试

6.3.2. 生产者客户端eurekaClient端8001

6.3.2.1. 默认配置

6.3.2.2. 修改配置

6.3.2.3. 测试

# 1. Eureka基础知识

## 1.1. 什么是服务治理

什么是服务治理

Spring Cloud封装了 Netflix公司开发的 Eureka模块来实现服务治理

在传统的rpc远程调用框架中，管理每个服务与服务之间依赖关系比较复杂，管理比较复杂，所以需要服务治理，管理服务与服务之间依赖关系，可以实现服务调用、负载均衡、容错等，实现服务发现与注册。

## 1.2. 什么是服务注册

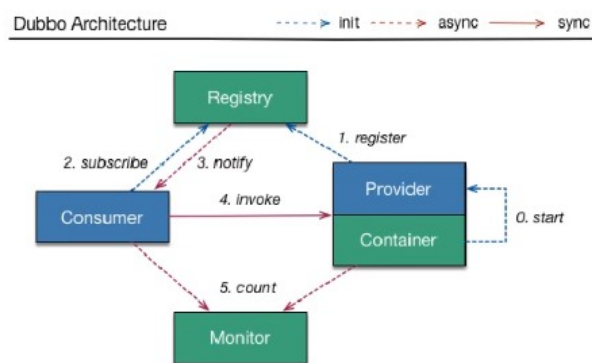
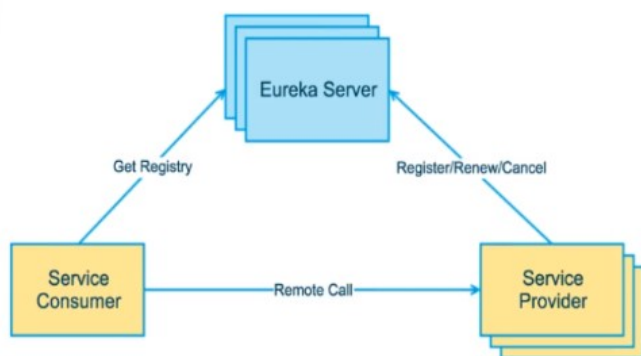
什么是服务注册与发现

Eureka采用了CS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。

在服务注册与发现中，有一个注册中心。当服务器启动的时候，会把当前自己服务器的信息比如服务地址通讯地址等以别名方式注册到注册中心上。另一方（消费者|服务提供者），以该别名的方式去注册中心上获取到实际的服务通讯地址，然后再实现本地RPC调用

RPC远程调用框架核心设计思想：在于注册中心，因为使用注册中心管理每个服务与服务之间的一个依赖关系（服务治理概念）。在任何rpc远程框架中，都会有一个注册中心存放服务地址相关信息接口地址

下左图是Eureka系统架构，右图是Dubbo的架构，请对比



## 1.3. Eureka两组件

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

Eureka Server提供服务注册服务

各个微服务节点通过配置启动后，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看到

Eureka Client通过注册中心进行访问

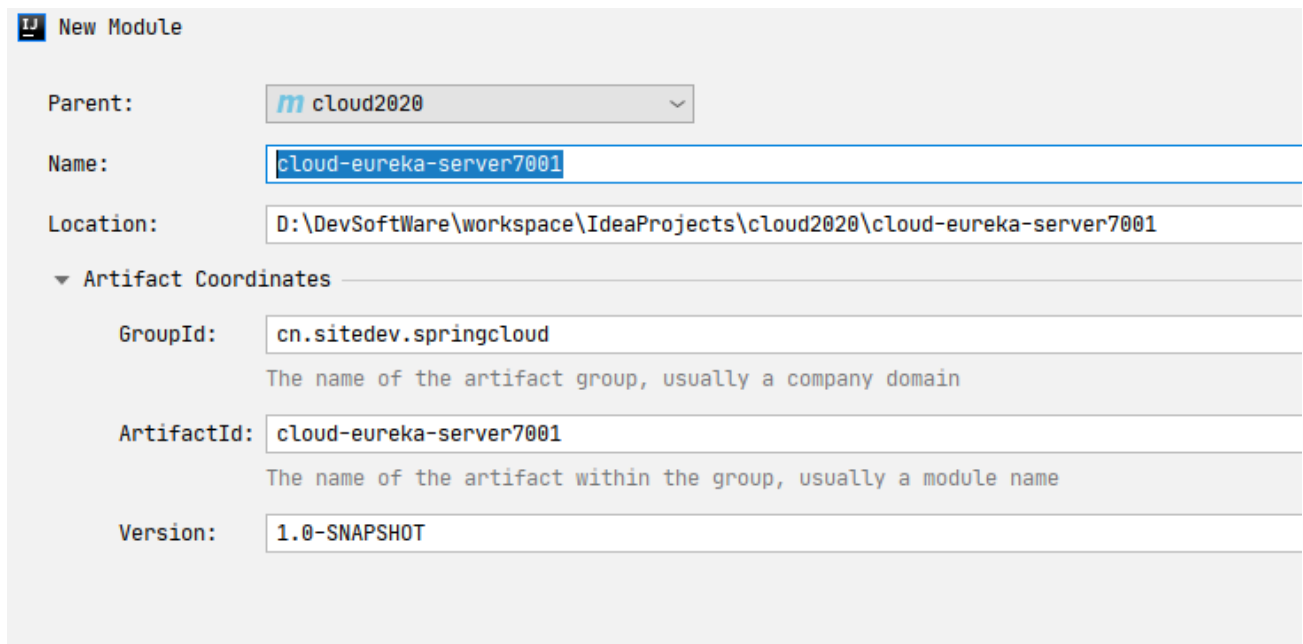
是一个Java客户端，用于简化 Eureka Server的交互，客户端同时也具备一个内置的、使用轮询（round-robin）负载算法的负载均衡器。在应用启动后，将会向 Eureka Server发送心跳（默认周期为30秒）。如果 Eureka Server在多个心跳周期内没有接收到某个节点的心跳，EurekaServer将会从服务注册表中把这个服务节点移除（默认90秒）

## 2. 单机Eureka构建步骤

### 2.1. IDEA生成EurekaServer端服务注册中心, 类似物业公司

#### 2.1.1. 建Module

创建cloud-eureka-server7001模块



New Module

Parent:

Name:

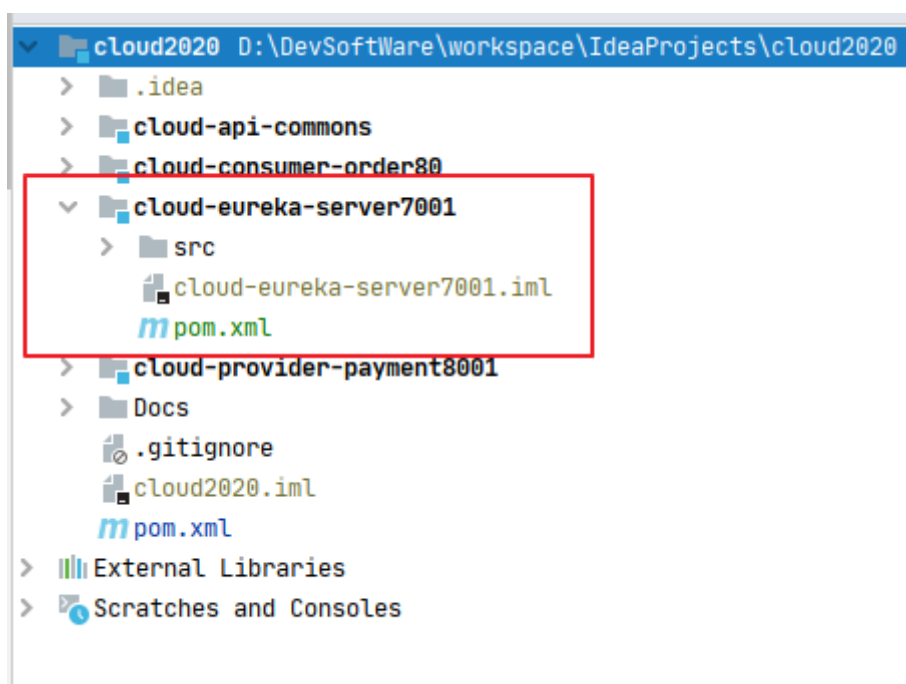
Location:

▼ Artifact Coordinates

GroupId:   
The name of the artifact group, usually a company domain

ArtifactId:   
The name of the artifact within the group, usually a module name

Version:



## 2.1.2. 改POM

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <project xmlns="http://maven.apache.org/POM/4.0.0"
3         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
4         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
5         http://maven.apache.org/xsd/maven-4.0.0.xsd">
6     <parent>
7         <artifactId>cloud2020</artifactId>
8         <groupId>cn.sitedev.springcloud</groupId>
9         <version>1.0-SNAPSHOT</version>
10    </parent>
11    <modelVersion>4.0.0</modelVersion>
12
13    <artifactId>cloud-eureka-server7001</artifactId>
14
15    <dependencies>
16        <!--eureka-server-->
17        <dependency>
18            <groupId>org.springframework.cloud</groupId>
19            <artifactId>spring-cloud-starter-netflix-eureka-server</artifactId>
20        </dependency>
21        <dependency>
22            <groupId>cn.sitedev.springcloud</groupId>
23            <artifactId>cloud-api-commons</artifactId>
24            <version>${project.version}</version>
25        </dependency>
26        <dependency>
27            <groupId>org.springframework.boot</groupId>
28            <artifactId>spring-boot-starter-web</artifactId>
29        </dependency>
30        <dependency>
31            <groupId>org.springframework.boot</groupId>
32            <artifactId>spring-boot-starter-actuator</artifactId>
33        </dependency>
34        <!--一般为通用配置-->
35        <dependency>
36            <groupId>org.springframework.boot</groupId>
37            <artifactId>spring-boot-devtools</artifactId>
38            <scope>runtime</scope>
39            <optional>true</optional>
40        </dependency>
```

```

40     <dependency>
41         <groupId>org.projectlombok</groupId>
42         <artifactId>lombok</artifactId>
43         <optional>true</optional>
44     </dependency>
45     <dependency>
46         <groupId>org.springframework.boot</groupId>
47         <artifactId>spring-boot-starter-test</artifactId>
48         <scope>test</scope>
49     </dependency>
50 </dependencies>
51
52 </project>

```

## 1.x和2.x的对比说明:

以前的老版本（当前使用2018）

```

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

```

现在新版本（当前使用2020.2）

```

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

```

## 2.1.3. 写YML

```

1 server:
2   port: 7001
3
4 eureka:
5   instance:
6     # eureka服务端的实例名称
7     hostname: localhost
8   client:
9     # false表示不向注册中心注册自己
10    register-with-eureka: false
11    # 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

```

```

12     fetch-registry: false
13     service-url:
14         # 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
15     defaultZone: http://${eureka.instance.hostname}:${server.port}/eureka/

```

## 2.1.4. 主启动

```

1 package cn.sitedev.springcloud;
2
3 import org.springframework.boot.SpringApplication;
4 import org.springframework.boot.autoconfigure.SpringBootApplication;
5 import org.springframework.cloud.netflix.eureka.server.EnableEurekaServer;
6
7 @SpringBootApplication
8 @EnableEurekaServer
9 public class EurekaMain7001 {
10     public static void main(String[] args) {
11         SpringApplication.run(EurekaMain7001.class, args);
12     }
13 }

```

## 2.1.5. 测试

启动应用, 浏览器访问<http://localhost:7001/>

The screenshot shows the Spring Eureka web interface. The header includes 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	Current time	2020-06-13T12:47:31 +0800
Data center	default	Uptime	00:00
		Lease expiration enabled	false
		Renews threshold	1
		Renews (last min)	0
- DS Replicas:** A section for distributed storage replicas.
- Instances currently registered with Eureka:** A table with columns for Application, AMIs, Availability Zones, and Status. It currently shows 'No instances available'.
- General Info:** A table showing system metrics.
 

Name	Value
total-avail-memory	369mb
environment	test

在"Instances currently registered with Eureka"这一项会提示: "No application available"(没有服务被发现), 这是因为没有注册服务进来当前不可能有服务被发现

## 2.2. EurekaClient端cloud-provider-payment8001

将注册进EurekaServer成为服务提供者provider, 类似于尚硅谷学校对外提供授课服务

### 2.2.1. 修改POM

```
1      <!--eureka-client-->
2      <dependency>
3          <groupId>org.springframework.cloud</groupId>
4          <artifactId>spring-cloud-starter-netflix-eureka-client</artifactId>
5      </dependency>
```

### 2.2.2. 修改主配置

```
1 eureka:
2   client:
3       # 表示是否将自己注册进EurekaServer, 默认为true
4       register-with-eureka: true
5       # 是否从EurekaServer抓取已有的注册信息, 默认为true
6       # 单节点无所谓, 集群必须设置为true才能配合ribbon使用负载均衡
7       fetch-registry: true
8       service-url:
9           defaultZone: http://localhost:7001/eureka
```

### 2.2.3. 修改启动类

```
1 package cn.sitedev.springcloud;
2
3 import org.springframework.boot.SpringApplication;
4 import org.springframework.boot.autoconfigure.SpringBootApplication;
5 import org.springframework.cloud.netflix.eureka.EnableEurekaClient;
6
7 @SpringBootApplication
8 @EnableEurekaClient
```



```

9 public class PaymentMain8001 {
10     public static void main(String[] args) {
11         SpringApplication.run(PaymentMain8001.class, args);
12     }
13 }

```

## 2.3. EurekaClient端cloud-consumer-order80

将注册进EurekaServer成为服务消费者consumer, 类似于尚硅谷学校上课消费的各位同学

### 2.3.1. 修改POM

```

1     <!--eureka-client-->
2     <dependency>
3         <groupId>org.springframework.cloud</groupId>
4         <artifactId>spring-cloud-starter-netflix-eureka-client</artifactId>
5     </dependency>

```

### 2.3.2. 修改主配置

```

1 spring:
2     application:
3         name: cloud-order-service
4 eureka:
5     client:
6         # 表示是否将自己注册进EurekaServer, 默认为true
7         register-with-eureka: true
8         # 是否从EurekaServer抓取已有的注册信息, 默认为true
9         # 单节点无所谓, 集群必须设置为true才能配合ribbon使用负载均衡
10        fetch-registry: true
11        service-url:
12            defaultZone: http://localhost:7001/eureka

```

### 2.3.3. 修改启动类

```

1 package cn.sitedev.springcloud;

```

```

2
3 import org.springframework.boot.SpringApplication;
4 import org.springframework.boot.autoconfigure.SpringBootApplication;
5 import org.springframework.cloud.netflix.eureka.EnableEurekaClient;
6
7 @SpringBootApplication
8 @EnableEurekaClient
9 public class OrderMain80 {
10     public static void main(String[] args) {
11         SpringApplication.run(OrderMain80.class, args);
12     }
13 }

```

## 2.4. 测试

先启动cloud-eureka-server7001

再启动cloud-provider-payment8001

接着启动cloud-consumer-order80

浏览器访问<http://localhost:7001/>

System Status			
Environment	test	Current time	2020-06-13T13:34:31 +0800
Data center	default	Uptime	00:02
		Lease expiration enabled	false
		Renews threshold	5
		Renews (last min)	0

DS Replicas			
Instances currently registered with Eureka			
Application	AMIs	Availability Zones	Status
CLOUD-ORDER-SERVICE	n/a (1)	(1)	UP (1) - localhost:cloud-order-service:80
CLOUD-PAYMENT-SERVICE	n/a (1)	(1)	UP (1) - localhost:cloud-payment-service:8001

对应主配置文件中的spring.application.name配置

## 2.5. 自我保护机制

此时页面会有如下提示:

**EMERGENCY! EUREKA MAY BE INCORRECTLY CLAIMING INSTANCES ARE UP WHEN THEY'RE NOT. RENEWALS ARE LESSER THAN THRESHOLD AND HENCE THE INSTANCES ARE NOT BEING EXPIRED JUST TO BE SAFE.**

## System Status

Environment	test	Current time	2020-06-13T13:37:53 +0800
Data center	default	Uptime	00:05
		Lease expiration enabled	false
		Renews threshold	5
		Renews (last min)	2

EMERGENCY! EUREKA MAY BE INCORRECTLY CLAIMING INSTANCES ARE UP WHEN THEY'RE NOT. RENEWALS ARE LESSER THAN THRESHOLD AND HENCE THE INSTANCES ARE NOT BEING EXPIRED JUST TO BE SAFE.

## DS Replicas

## Instances currently registered with Eureka

Application	AMIs	Availability Zones	Status
CLOUD-ORDER-SERVICE	n/a (1)	(1)	UP (1) - localhost:cloud-order-service:80
CLOUD-PAYMENT-SERVICE	n/a (1)	(1)	UP (1) - localhost:cloud-payment-service:8001

## 3. 集群Eureka构建步骤

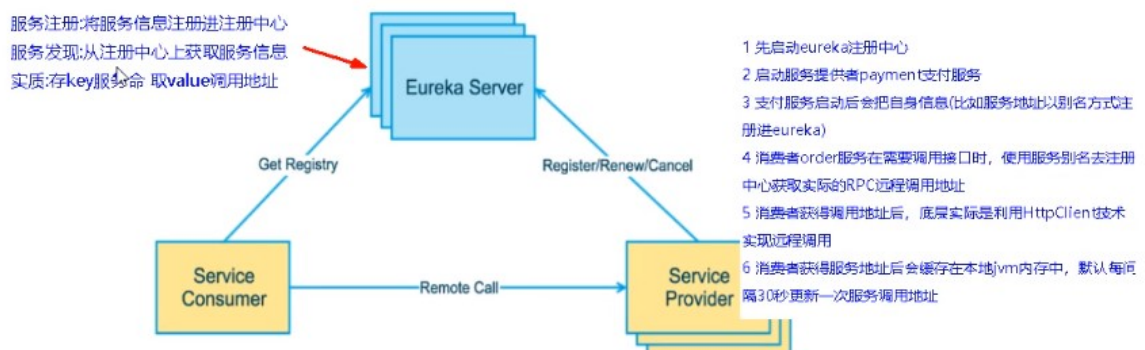
### 3.1. Eureka集群原理说明

服务注册: 服务信息注册进注册中心

服务发现: 从注册中心上获取服务信息

实质: 存key服务名, 取value调用地址

- 1 先启动 eureka注册中心
- 2 启动服务提供者 payment支付服务
- 3 支付服务启动后会把自身信息 (比如服务地址以别名方式注册进 eureka)
- 4 消费者 order服务再需要调用接口时, 使用服务别名去注册中心获取实际的RPC远程调用地址
- 5 消费者获得调用地址后, 底层实际是利用 HttpClient技术实现远程调用
- 6 消费者获得服务地址后会缓存在本地jvm内存中, 默认每间隔30秒更新一次服务调用地址



问题: 微服务RPC远程服务调用最核心的是什么

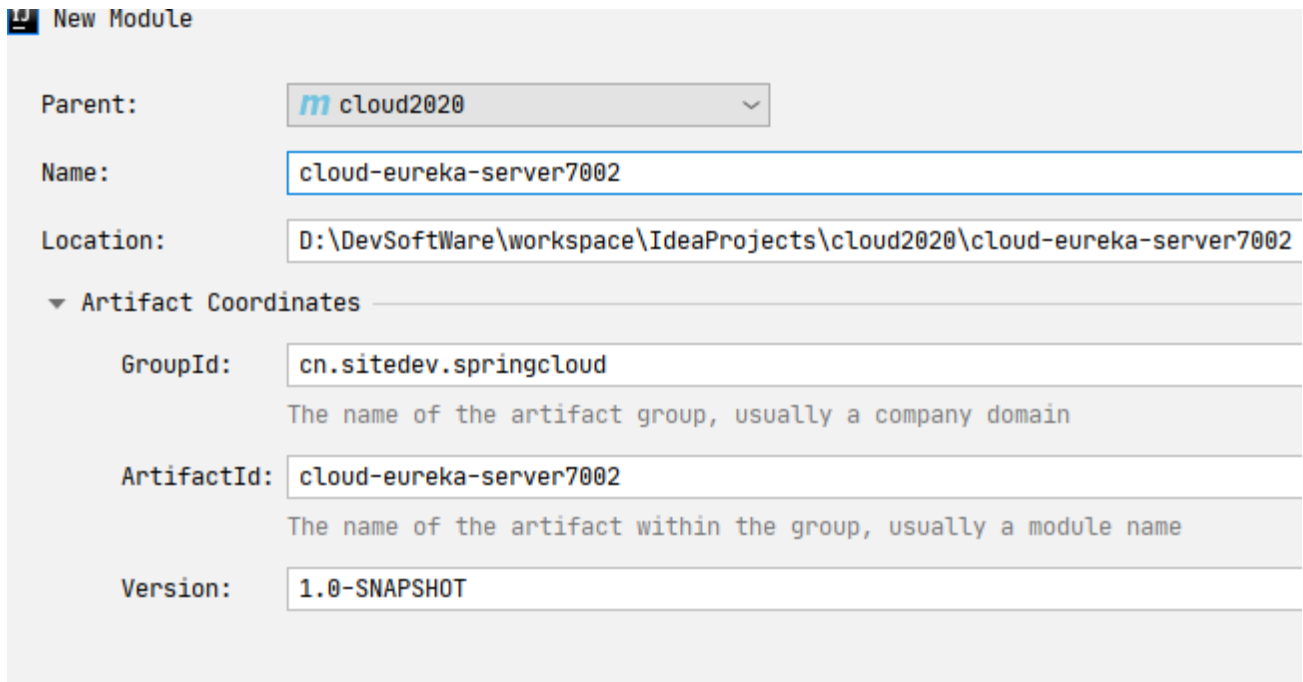
高可用, 试想你的注册中心只有一个only one, 它出故障了那就呵呵(╯▽╰)了, 会导致整个为服务环境不可用, 所以

解决办法: 搭建Eureka注册中心集群, 实现负载均衡+故障容错

## 3.2. Eureka集群环境构建步骤

### 3.2.1. 参考cloud-eureka-server7001新建cloud-eureka-server7002模块

新建cloud-eureka-server7002模块



New Module

Parent:

Name:

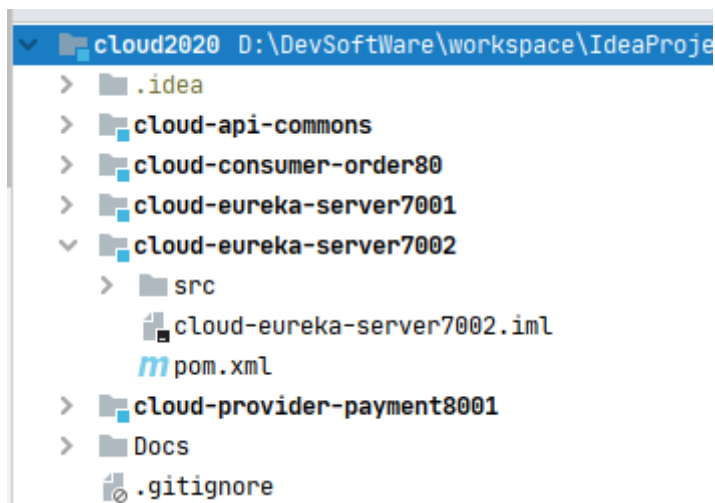
Location:

▼ Artifact Coordinates

GroupId:   
The name of the artifact group, usually a company domain

ArtifactId:   
The name of the artifact within the group, usually a module name

Version:



### 3.2.2. 改POM

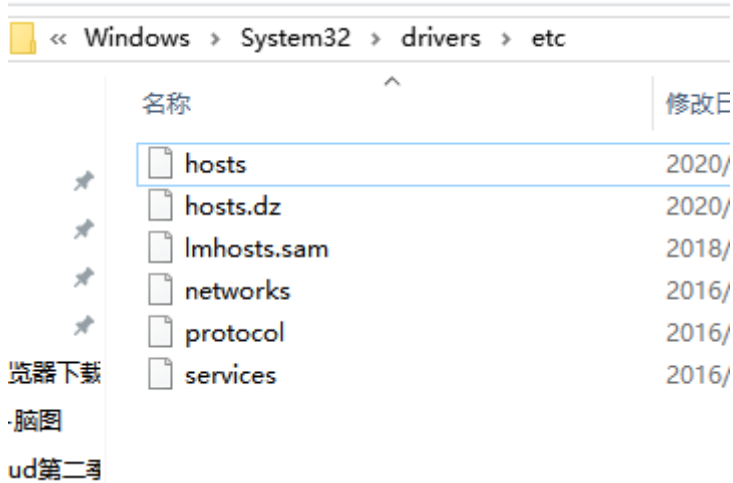
```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <project xmlns="http://maven.apache.org/POM/4.0.0"
3     xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
4     xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
5     http://maven.apache.org/xsd/maven-4.0.0.xsd">
6     <parent>
7         <artifactId>cloud2020</artifactId>
8         <groupId>cn.sitedev.springcloud</groupId>
```

```
8      <version>1.0-SNAPSHOT</version>
9  </parent>
10 <modelVersion>4.0.0</modelVersion>
11
12 <artifactId>cloud-eureka-server7002</artifactId>
13
14 <dependencies>
15     <!--eureka-server-->
16     <dependency>
17         <groupId>org.springframework.cloud</groupId>
18         <artifactId>spring-cloud-starter-netflix-eureka-server</artifactId>
19     </dependency>
20     <dependency>
21         <groupId>cn.sitedev.springcloud</groupId>
22         <artifactId>cloud-api-commons</artifactId>
23         <version>${project.version}</version>
24     </dependency>
25     <dependency>
26         <groupId>org.springframework.boot</groupId>
27         <artifactId>spring-boot-starter-web</artifactId>
28     </dependency>
29     <dependency>
30         <groupId>org.springframework.boot</groupId>
31         <artifactId>spring-boot-starter-actuator</artifactId>
32     </dependency>
33     <!--一般为通用配置-->
34     <dependency>
35         <groupId>org.springframework.boot</groupId>
36         <artifactId>spring-boot-devtools</artifactId>
37         <scope>runtime</scope>
38         <optional>true</optional>
39     </dependency>
40     <dependency>
41         <groupId>org.projectlombok</groupId>
42         <artifactId>lombok</artifactId>
43         <optional>true</optional>
44     </dependency>
45     <dependency>
46         <groupId>org.springframework.boot</groupId>
47         <artifactId>spring-boot-starter-test</artifactId>
48         <scope>test</scope>
49     </dependency>
50 </dependencies>
```

```
51 </project>
```

### 3.2.3. 修改映射配置

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



修改映射配置添加加入hosts文件

```
1 127.0.0.1 eureka7001.com
2 127.0.0.1 eureka7002.com
```

刷新hosts文件

```
1 ipconfig /flushdns
```

### 3.2.4. 写YML

#### 3.2.4.1. 7001以前的配置(单机)

```
1 server:
2   port: 7001
3
4 eureka:
5   instance:
6     # eureka服务端的实例名称
7     hostname: localhost
```

```

8  client:
9      # false表示不向注册中心注册自己
10     register-with-eureka: false
11     # 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
12     fetch-registry: false
13     service-url:
14         # 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
15         defaultZone: http://${eureka.instance.hostname}:${server.port}/eureka/

```

### 3.2.4.2. 7001现在的配置(集群)

```

1  server:
2      port: 7001
3  spring:
4      application:
5          name: cloud-eureka-service
6  eureka:
7      instance:
8          # eureka服务端的实例名称
9          hostname: eureka7001.com
10     client:
11         # false表示不向注册中心注册自己
12         register-with-eureka: false
13         # false表示自己端就是注册中心,我的职责就是维护服务实例,并不需要检索服务
14         fetch-registry: false
15         service-url:
16             # 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
17             defaultZone: http://eureka7002.com:7002/eureka/
18

```

### 3.2.4.3. 7002现在的配置(集群)

```

1  server:
2      port: 7002
3  spring:
4      application:
5          name: cloud-eureka-service2
6  eureka:

```

```

7   instance:
8       hostname: eureka7002.com
9   client:
10      # false表示不向注册中心注册自己
11      register-with-eureka: false
12      # false表示自己端就是注册中心,我的职责就是维护服务实例,并不需要检索服务
13      fetch-registry: false
14      service-url:
15          # 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
16          defaultZone: http://eureka7001.com:7001/eureka/

```

### 3.2.5. 主启动

```

1  package cn.sitedev.springcloud;
2
3  import org.springframework.boot.SpringApplication;
4  import org.springframework.boot.autoconfigure.SpringBootApplication;
5  import org.springframework.cloud.netflix.eureka.server.EnableEurekaServer;
6
7  @SpringBootApplication
8  @EnableEurekaServer
9  public class EurekaMain7002 {
10      public static void main(String[] args) {
11          SpringApplication.run(EurekaMain7002.class, args);
12      }
13  }

```

## 3.3. 将支付服务8001微服务发布到上面2台Eureka集群配置中

修改主配置:

```

1  eureka:
2      client:
3          # 表示是否将自己注册进EurekaServer, 默认为true
4          register-with-eureka: true
5          # 是否从EurekaServer抓取已有的注册信息, 默认为true
6          # 单节点无所谓, 集群必须设置为true才能配合ribbon使用负载均衡
7          fetch-registry: true

```



```
8     service-url:
9     defaultZone:
http://eureka7001.com:7001/eureka,http://eureka7002.com:7002/eureka
```

```
1 server:
2   port: 8001
3
4   spring:
5     application:
6       name: cloud-payment-service
7     datasource:
8       type: com.alibaba.druid.pool.DruidDataSource # 当前数据源操作类型
9       driver-class-name: org.gjt.mm.mysql.Driver # mysql驱动包
10      url: jdbc:mysql://localhost:3306/db2019?useUnicode=true&characterEncoding=utf-8&useSSL=false
11      username: root
12      password: root
13
14
15   mybatis:
16     mapperLocations: classpath:mapper/*.xml
17     type-aliases-package: cn.sitedev.springcloud.entities # 所有Entity别名类所在包
18
19   eureka:
20     client:
21       # 表示是否将自己注册进EurekaServer, 默认为true
22       register-with-eureka: true
23       # 是否从EurekaServer抓取已有的注册信息, 默认为true
24       # 单节点无所谓, 集群必须设置为true才能配合ribbon使用负载均衡
25       fetch-registry: true
26       service-url:
27         defaultZone: http://eureka7001.com:7001/eureka,http://eureka7002.com:7002/eureka
```

### 3.4. 将订单服务80微服务发布到上面2台Eureka集群配置中

修改主配置:

```
1 eureka:
2   client:
3     # 表示是否将自己注册进EurekaServer, 默认为true
4     register-with-eureka: true
5     # 是否从EurekaServer抓取已有的注册信息, 默认为true
6     # 单节点无所谓, 集群必须设置为true才能配合ribbon使用负载均衡
7     fetch-registry: true
8     service-url:
9     defaultZone:
http://eureka7001.com:7001/eureka,http://eureka7002.com:7002/eureka
```

```
cloud-consumer-order80\...\application.yml
1 server:
2   port: 80
3 spring:
4   application:
5     name: cloud-order-service
6 eureka:
7   client:
8     # 表示是否将自己注册进EurekaServer, 默认为true
9     register-with-eureka: true
10    # 是否从EurekaServer抓取已有的注册信息, 默认为true
11    # 单节点无所谓, 集群必须设置为true才能配合ribbon使用负载均衡
12    fetch-registry: true
13    service-url:
14      defaultZone: http://eureka7001.com:7001/eureka,http://eureka7002.com:7002/eureka
```

### 3.5. 测试01

先要启动EurekaServer,7001/7002服务

再要启动服务提供者provider,8001

再要启动服务提供者provider,8001

浏览器访问<http://eureka7001.com:7001/>

← → ↻ ⌂ ⓘ 不安全 | eureka7001.com:7001

应用 设置 设备 工具 网络 存储 系统 手机 电视 平板 相机 打印机 扫描仪 摄像头 鼠标 键盘 耳机 音箱 显示器 投影仪 智能设备 其他设备

Data center	default	Uptime	00:08
		Lease expiration enabled	false
		Renews threshold	5
		Renews (last min)	0

EMERGENCY! EUREKA MAY BE INCORRECTLY CLAIMING INSTANCES ARE UP WHEN THEY'RE NOT. RENEWALS ARE LESSEF AND HENCE THE INSTANCES ARE NOT BEING EXPIRED JUST TO BE SAFE.

### DS Replicas

eureka7002.com

### Instances currently registered with Eureka

Application	AMIs	Availability Zones	Status
CLOUD-ORDER-SERVICE	n/a (1)	(1)	UP (1) - localhost:cloud-order-service:80
CLOUD-PAYMENT-SERVICE	n/a (1)	(1)	UP (1) - localhost:cloud-payment-service:8001

### General Info

浏览器访问<http://eureka7002.com:7002/>



```
13
14 <dependencies>
15     <!--eureka-client-->
16     <dependency>
17         <groupId>org.springframework.cloud</groupId>
18         <artifactId>spring-cloud-starter-netflix-eureka-client</artifactId>
19     </dependency>
20     <!--引入自己定义的api通用包，可以使用Payment支付Entity-->
21     <dependency>
22         <groupId>cn.sitedev.springcloud</groupId>
23         <artifactId>cloud-api-commons</artifactId>
24         <version>${project.version}</version>
25     </dependency>
26     <dependency>
27         <groupId>org.springframework.boot</groupId>
28         <artifactId>spring-boot-starter-web</artifactId>
29     </dependency>
30
31     <dependency>
32         <groupId>org.springframework.boot</groupId>
33         <artifactId>spring-boot-starter-actuator</artifactId>
34     </dependency>
35
36     <dependency>
37         <groupId>org.mybatis.spring.boot</groupId>
38         <artifactId>mybatis-spring-boot-starter</artifactId>
39     </dependency>
40
41     <dependency>
42         <groupId>com.alibaba</groupId>
43         <artifactId>druid-spring-boot-starter</artifactId>
44         <version>1.1.10</version>
45     </dependency>
46     <dependency>
47         <groupId>mysql</groupId>
48         <artifactId>mysql-connector-java</artifactId>
49     </dependency>
50
51     <dependency>
52         <groupId>org.springframework.boot</groupId>
53         <artifactId>spring-boot-starter-jdbc</artifactId>
54     </dependency>
55
```

```

56     <dependency>
57         <groupId>org.springframework.boot</groupId>
58         <artifactId>spring-boot-devtools</artifactId>
59         <scope>runtime</scope>
60         <optional>true</optional>
61     </dependency>
62     <dependency>
63         <groupId>org.projectlombok</groupId>
64         <artifactId>lombok</artifactId>
65         <optional>true</optional>
66     </dependency>
67     <dependency>
68         <groupId>org.springframework.boot</groupId>
69         <artifactId>spring-boot-starter-test</artifactId>
70         <scope>test</scope>
71     </dependency>
72
73 </dependencies>
74
75 </project>

```

### 3.6.3. 改主配置

```

1 server:
2   port: 8002
3
4 spring:
5   application:
6     name: cloud-payment-service
7   datasource:
8     type: com.alibaba.druid.pool.DruidDataSource # 当前数据源操作类型
9     driver-class-name: org.gjt.mm.mysql.Driver # mysql驱动包
10    url: jdbc:mysql://localhost:3306/db2019?
11    useUnicode=true&characterEncoding=utf-8&useSSL=false
12    username: root
13    password: root
14
15 mybatis:
16   mapperLocations: classpath:mapper/*.xml
17   type-aliases-package: cn.sitedev.springcloud.entities # 所有Entity别名类所在包

```

```

18
19 eureka:
20   client:
21     # 表示是否将自己注册进EurekaServer, 默认为true
22     register-with-eureka: true
23     # 是否从EurekaServer抓取已有的注册信息, 默认为true
24     # 单节点无所谓, 集群必须设置为true才能配合ribbon使用负载均衡
25     fetch-registry: true
26     service-url:
27       defaultZone:
http://eureka7001.com:7001/eureka,http://eureka7002.com:7002/eureka

```

### 3.6.4. 主启动

```

1 package cn.sitedev.springcloud;
2
3 import org.springframework.boot.SpringApplication;
4 import org.springframework.boot.autoconfigure.SpringBootApplication;
5 import org.springframework.cloud.netflix.eureka.EnableEurekaClient;
6
7 @SpringBootApplication
8 @EnableEurekaClient
9 public class PaymentMain8002 {
10     public static void main(String[] args) {
11         SpringApplication.run(PaymentMain8002.class, args);
12     }
13 }

```

### 3.6.5. 业务类

controller:

```

1 package cn.sitedev.springcloud.controller;
2
3 import cn.sitedev.springcloud.entities.CommonResult;
4 import cn.sitedev.springcloud.entities.Payment;
5 import cn.sitedev.springcloud.service.PaymentService;
6 import lombok.extern.slf4j.Slf4j;
7 import org.springframework.web.bind.annotation.*;

```

```

8
9 import javax.annotation.Resource;
10
11 @RestController
12 @Slf4j
13 @RequestMapping("/payment")
14 public class PaymentController {
15     @Resource
16     private PaymentService paymentService;
17
18     @PostMapping(value = "/create")
19     public CommonResult create(@RequestBody Payment payment) {
20         int result = paymentService.create(payment);
21         log.info("*****插入结果: " + result);
22         if (result > 0) {
23             return new CommonResult(200, "插入数据库成功", result);
24         } else {
25             return new CommonResult(444, "插入数据库失败", null);
26         }
27     }
28
29     @GetMapping(value = "/get/{id}")
30     public CommonResult getPaymentById(@PathVariable("id") Long id) {
31         Payment payment = paymentService.getPaymentById(id);
32         log.info("*****查询结果: " + payment);
33         if (payment != null) {
34             return new CommonResult(200, "查询成功", payment);
35         } else {
36             return new CommonResult(444, "没有对应记录, 查询id: " + id, null);
37         }
38     }
39 }

```

service:

```

1 package cn.sitedev.springcloud.service;
2
3 import cn.sitedev.springcloud.entities.Payment;
4
5 public interface PaymentService {
6     int create(Payment payment);

```

```

7
8     Payment getPaymentById(Long id);
9 }
10 //////////////////////////////////////////////////
11 package cn.sitedev.springcloud.service;
12
13 import cn.sitedev.springcloud.dao.PaymentDao;
14 import cn.sitedev.springcloud.entities.Payment;
15 import org.springframework.stereotype.Service;
16
17 import javax.annotation.Resource;
18
19 @Service
20 public class PaymentServiceImpl implements PaymentService {
21     @Resource
22     private PaymentDao paymentDao;
23
24     @Override
25     public int create(Payment payment) {
26         return paymentDao.create(payment);
27     }
28
29     @Override
30     public Payment getPaymentById(Long id) {
31         return paymentDao.getPaymentById(id);
32     }
33 }

```

dao:

```

1 package cn.sitedev.springcloud.dao;
2
3 import cn.sitedev.springcloud.entities.Payment;
4 import org.apache.ibatis.annotations.Mapper;
5 import org.apache.ibatis.annotations.Param;
6
7 @Mapper
8 public interface PaymentDao {
9     int create(Payment payment);
10
11     Payment getPaymentById(@Param("id") Long id);

```



```
12 }
```

### 3.6.6. 修改8001/8002的controller

8001/8002:

```
1 package cn.sitedev.springcloud.controller;
2
3 import cn.sitedev.springcloud.entities.CommonResult;
4 import cn.sitedev.springcloud.entities.Payment;
5 import cn.sitedev.springcloud.service.PaymentService;
6 import lombok.extern.slf4j.Slf4j;
7 import org.springframework.beans.factory.annotation.Value;
8 import org.springframework.web.bind.annotation.*;
9
10 import javax.annotation.Resource;
11
12 @RestController
13 @Slf4j
14 @RequestMapping("/payment")
15 public class PaymentController {
16     @Resource
17     private PaymentService paymentService;
18
19     @Value("${server.port}")
20     private String serverPort;
21
22     @PostMapping(value = "/create")
23     public CommonResult create(@RequestBody Payment payment) {
24         int result = paymentService.create(payment);
25         log.info("*****插入结果: " + result);
26         if (result > 0) {
27             return new CommonResult(200, "插入数据库成功, serverPort: " +
serverPort, result);
28         } else {
29             return new CommonResult(444, "插入数据库失败", null);
30         }
31     }
32
33     @GetMapping(value = "/get/{id}")
34     public CommonResult getPaymentById(@PathVariable("id") Long id) {
```

```

35     Payment payment = paymentService.getPaymentById(id);
36     log.info("*****查询结果: " + payment);
37     if (payment != null) {
38         return new CommonResult(200, "查询成功, serverPort: " + serverPort,
payment);
39     } else {
40         return new CommonResult(444, "没有对应记录, 查询id: " + id, null);
41     }
42 }
43 }

```

## 3.7. 负载均衡

### 3.7.1. bug => 订单服务访问地址存在硬编码,无法进行负载均衡

修改订单服务cloud-consumer-order80模块的controller中的订单服务访问地址

```

1  package cn.sitedev.springcloud.controller;
2
3  import cn.sitedev.springcloud.entities.CommonResult;
4  import cn.sitedev.springcloud.entities.Payment;
5  import lombok.extern.slf4j.Slf4j;
6  import org.springframework.web.bind.annotation.*;
7  import org.springframework.web.client.RestTemplate;
8
9  import javax.annotation.Resource;
10
11 @RestController
12 @RequestMapping("/consumer/payment")
13 @Slf4j
14 public class OrderController {
15
16     //    public static final String PAYMENT_URL = "http://localhost:8001";
17
18     // 通过在 eureka上注册过的微服务名称调用
19     public static final String PAYMENT_URL = "http://CLOUD-PAYMENT-SERVICE";
20
21     @Resource
22     private RestTemplate restTemplate;
23
24     @PostMapping(value = "/create")

```

```

25     public CommonResult<Payment> create(Payment payment) {
26         return restTemplate.postForObject(PAYMENT_URL + "/payment/create",
payment, CommonResult.class);
27     }
28
29     @GetMapping(value = "/get/{id}")
30     public CommonResult<Payment> getPayment(@PathVariable("id") Long id) {
31         return restTemplate.getForObject(PAYMENT_URL + "/payment/get/" + id,
CommonResult.class);
32     }
33 }

```

← → ↻ ⌂ ① 不安全 | eureka7001.com:7001

Lease expiration enabled	false
Renews threshold	6
Renews (last min)	4

**EMERGENCY! EUREKA MAY BE INCORRECTLY CLAIMING INSTANCES ARE UP WHEN THEY'RE NOT. RENEWALS ARE LESSER TH AND HENCE THE INSTANCES ARE NOT BEING EXPIRED JUST TO BE SAFE.**

### DS Replicas

eureka7002.com

### Instances currently registered with Eureka

Application	AMIs	Availability Zones	Status
CLOUD-ORDER-SERVICE	n/a (1)	(1)	UP (1) - localhost:cloud-order-service:80
CLOUD-PAYMENT-SERVICE	n/a (2)	(2)	UP (2) - localhost:cloud-payment-service:8002 , localhost:cloud-payment-service:8001

### 3.7.2. 使用@LoadBalanced注解赋予RestTemplate负载均衡的能力

我们通过浏览器访问<http://localhost/consumer/payment/get/1>

← → ↻ ⌂ ① localhost/consumer/payment/get/1

## Whitelabel Error Page

This application has no explicit mapping for /error, so you are seeing this as a fallback.

Sat Jun 13 15:20:30 CST 2020  
There was an unexpected error (type=Internal Server Error, status=500).  
I/O error on GET request for "http://CLOUD-PAYMENT-SERVICE/payment/get/1": CLOUD-PAYMENT-SERVICE; nested exception is java.net.UnknownHostException: CLOUD-PAYMENT-SERVICE  
org.springframework.web.client.ResourceAccessException: I/O error on GET request for "http://CLOUD-PAYMENT-SERVICE/payment/get/1": CLOUD-PAYMENT-SERVICE; nested exception is java.net.UnknownHostException: CLOUD-PAYMENT-SERVICE  
at org.springframework.web.client.RestTemplate.doExecute(RestTemplate.java:751)  
at org.springframework.web.client.RestTemplate.execute(RestTemplate.java:677)  
at org.springframework.web.client.RestTemplate.getForObject(RestTemplate.java:318)  
at cn.sitedev.springcloud.controller.OrderController.getPayment(OrderController.java:31)  
at sun.reflect.NativeMethodAccessorImpl.invoke0(Native Method)  
at sun.reflect.NativeMethodAccessorImpl.invoke(NativeMethodAccessorImpl.java:62)  
at sun.reflect.DelegatingMethodAccessorImpl.invoke(DelegatingMethodAccessorImpl.java:43)  
at java.lang.reflect.Method.invoke(Method.java:498)  
at org.springframework.web.method.support.InvocableHandlerMethod.doInvoke(InvocableHandlerMethod.java:190)  
at org.springframework.web.method.support.InvocableHandlerMethod.invokeForRequest(InvocableHandlerMethod.java:138)  
at org.springframework.web.servlet.mvc.method.annotation.ServletInvocableHandlerMethod.invokeAndHandle(ServletInvocableHandlerMethod.java:106)

发现产生了异常, 因此进行一些修改

修改订单服务cloud-consumer-order80模块的配置类

```

1 package cn.sitedev.springcloud.config;
2
3 import org.springframework.cloud.client.loadbalancer.LoadBalanced;
4 import org.springframework.context.annotation.Bean;
5 import org.springframework.context.annotation.Configuration;
6 import org.springframework.web.client.RestTemplate;
7
8 @Configuration
9 public class ApplicationContextConfig {
10
11     @LoadBalanced
12     @Bean
13     public RestTemplate restTemplate() {
14         return new RestTemplate();
15     }
16 }

```

## 3.8. 测试02

先要启动EurekaServer,7001/7002服务

再要启动服务提供者provider,8001/8002服务

浏览器访问<http://localhost/consumer/payment/get/1>





结果: 负载均衡效果达到,8001/8002端口交替出现

结论: Ribbon和Eureka整合后Consumer可以直接调用服务而不用再关心地址和端口号,且该服务还有负载均衡了

## 4. actuator微服务信息完善

### 4.1. 主机名称:服务名称修改

#### 4.1.1. 当前问题

Instances currently registered with Eureka			
Application	AMIs	Availability Zones	Status
CLOUD-ORDER-SERVICE	n/a (1)	(1)	UP (1) - localhost:cloud-order-service:80
CLOUD-PAYMENT-SERVICE	n/a (2)	(2)	UP (2) - localhost:cloud-payment-service:8002 , localhost:cloud-payment-service:8001

含有主机名称

#### 4.1.2. 修改cloud-provoder-payment8001/cloud-provoder-payment8002

修改cloud-provoder-payment8001的主配置

```
1 eureka:
2   client:
3     # 表示是否将自己注册进EurekaServer, 默认为true
4     register-with-eureka: true
5     # 是否从EurekaServer抓取已有的注册信息, 默认为true
6     # 单节点无所谓, 集群必须设置为true才能配合ribbon使用负载均衡
7     fetch-registry: true
```

```

8     service-url:
9     defaultZone:
http://eureka7001.com:7001/eureka,http://eureka7002.com:7002/eureka
10    instance:
11    instance-id: payment8001

```

修改cloud-provoder-payment8002的主配置

```

1 eureka:
2   client:
3     # 表示是否将自己注册进EurekaServer, 默认为true
4     register-with-eureka: true
5     # 是否从EurekaServer抓取已有的注册信息, 默认为true
6     # 单节点无所谓, 集群必须设置为true才能配合ribbon使用负载均衡
7     fetch-registry: true
8     service-url:
9     defaultZone:
http://eureka7001.com:7001/eureka,http://eureka7002.com:7002/eureka
10    instance:
11    instance-id: payment8002

```

### 4.1.3. 修改之后

← → ↻ ⌂ ⓘ 不安全 | eureka7001.com:7001

EMERGENCY! EUREKA MAY BE INCORRECTLY CLAIMING INSTANCES ARE UP WHEN THEY'RE NOT. RENEWALS ARE LESSER AND HENCE THE INSTANCES ARE NOT BEING EXPIRED JUST TO BE SAFE.

DS Replicas

eureka7002.com

Instances currently registered with Eureka

Application	AMIs	Availability Zones	Status
CLOUD-ORDER-SERVICE	n/a (1)	(1)	UP (1) - localhost:cloud-order-service:80
CLOUD-PAYMENT-SERVICE	n/a (2)	(2)	UP (2) - payment8002 , payment8001

General Info

## 4.2. 访问信息有IP信息提示

### 4.2.1. 当前问题

EMERGENCY! EUREKA MAY BE INCORRECTLY CLAIMING INSTANCES ARE UP WHEN THEY'RE NOT. RENEWALS ARE LESSER 1 AND HENCE THE INSTANCES ARE NOT BEING EXPIRED JUST TO BE SAFE.

## DS Replicas

eureka7002.com

### Instances currently registered with Eureka

Application	AMIs	Availability Zones	Status
CLOUD-ORDER-SERVICE	n/a (1)	(1)	UP (1) - localhost:cloud-order-service:80
CLOUD-PAYMENT-SERVICE	n/a (2)	(2)	UP (2) - payment8002, payment8001

### General Info

Name	Value
total-avail-memory	484mb
environment	test
num-of-cpus	8
current-memory-usage	288mb (59%)
server-uptime	00:29
localhost:8002/actuator/info	http://eureka7002.com:7002/eureka/

没有IP提示

## 4.2.2. 修改cloud-provoder-payment8001/cloud-provoder-payment8002

修改cloud-provoder-payment8001的主配置

```
1 eureka:
2   client:
3     # 表示是否将自己注册进EurekaServer, 默认为true
4     register-with-eureka: true
5     # 是否从EurekaServer抓取已有的注册信息, 默认为true
6     # 单节点无所谓, 集群必须设置为true才能配合ribbon使用负载均衡
7     fetch-registry: true
8     service-url:
9       defaultZone:
10      http://eureka7001.com:7001/eureka,http://eureka7002.com:7002/eureka
11   instance:
12     instance-id: payment8001
13     prefer-ip-address: true # 访问路径可以显示IP地址
```

修改cloud-provoder-payment8002的主配置

```
1 server:
```

```

2   port: 8002
3
4  spring:
5    application:
6      name: cloud-payment-service
7    datasource:
8      type: com.alibaba.druid.pool.DruidDataSource # 当前数据源操作类型
9      driver-class-name: org.gjt.mm.mysql.Driver # mysql驱动包
10     url: jdbc:mysql://localhost:3306/db2019?
        useUnicode=true&characterEncoding=utf-8&useSSL=false
11     username: root
12     password: root
13
14
15  mybatis:
16    mapperLocations: classpath:mapper/*.xml
17    type-aliases-package: cn.sitedev.springcloud.entities # 所有Entity别名类所在包
18
19  eureka:
20    client:
21      # 表示是否将自己注册进EurekaServer, 默认为true
22      register-with-eureka: true
23      # 是否从EurekaServer抓取已有的注册信息, 默认为true
24      # 单节点无所谓, 集群必须设置为true才能配合ribbon使用负载均衡
25      fetch-registry: true
26      service-url:
27        defaultZone:
28          http://eureka7001.com:7001/eureka,http://eureka7002.com:7002/eureka
29    instance:
30      instance-id: payment8002
31      prefer-ip-address: true # 访问路径可以显示IP地址

```

### 4.2.3. 修改之后



## Instances currently registered with Eureka

Application	AMIs	Availability Zones	Status
CLOUD-ORDER-SERVICE	n/a (1)	(1)	UP (1) - localhost:cloud-order-service:80
CLOUD-PAYMENT-SERVICE	n/a (2)	(2)	UP (2) - payment8002, payment8001

## General Info

Name	Value
total-avail-memory	484mb
environment	test
num-of-cpus	8
current-memory-usage	303mb (62%)
server-uptime	00:33
registered-replicas	http://eureka7002.com:7002/eureka/
unavailable-replicas	http://eureka7002.com:7002/eureka/
available-replicas	

192.168.5.1:8002/actuator/info

## 5. 服务发现Discovery

对于注册eureka里面的微服务,可以通过服务发现来获得该服务的信息

### 5.1. 修改cloud-provider-payment8001/cloud-provider-payment8002模块

#### 5.1.1. 修改controller

cloud-provider-payment8001/cloud-provider-payment8002

```

1 package cn.sitedev.springcloud.controller;
2
3 import cn.sitedev.springcloud.entities.CommonResult;
4 import cn.sitedev.springcloud.entities.Payment;
5 import cn.sitedev.springcloud.service.PaymentService;
6 import lombok.extern.slf4j.Slf4j;
7 import org.springframework.beans.factory.annotation.Value;
8 import org.springframework.cloud.client.ServiceInstance;
9 import org.springframework.cloud.client.discovery.DiscoveryClient;
10 import org.springframework.web.bind.annotation.*;
11
12 import javax.annotation.Resource;
13 import java.util.List;

```

```
14
15 @RestController
16 @Slf4j
17 @RequestMapping("/payment")
18 public class PaymentController {
19     @Resource
20     private PaymentService paymentService;
21
22     /**
23      * 端口号
24      * 查看负载均衡效果
25      */
26     @Value("${server.port}")
27     private String serverPort;
28
29     /**
30      * 服务发现，获取服务信息
31      */
32     @Resource
33     private DiscoveryClient discoveryClient;
34
35     @PostMapping(value = "/create")
36     public CommonResult create(@RequestBody Payment payment) {
37         int result = paymentService.create(payment);
38         log.info("*****插入结果: " + result);
39         if (result > 0) {
40             return new CommonResult(200, "插入数据库成功, serverPort: " +
serverPort, result);
41         } else {
42             return new CommonResult(444, "插入数据库失败", null);
43         }
44     }
45
46     @GetMapping(value = "/get/{id}")
47     public CommonResult getPaymentById(@PathVariable("id") Long id) {
48         Payment payment = paymentService.getPaymentById(id);
49         log.info("*****查询结果: " + payment);
50         if (payment != null) {
51             return new CommonResult(200, "查询成功, serverPort: " + serverPort,
payment);
52         } else {
53             return new CommonResult(444, "没有对应记录, 查询id: " + id, null);
54         }
55     }
}
```

```

56
57     /**
58     * 服务发现
59     *
60     * @return
61     */
62     @GetMapping("/discovery")
63     public Object discovery() {
64         List<String> services = discoveryClient.getServices();
65         for (String element : services) {
66             log.info("*****element: " + element);
67         }
68         // 一个微服务下的全部实例
69         List<ServiceInstance> instances = discoveryClient.getInstances("CLOUD-
PAYMENT-SERVICE");
70         for (ServiceInstance instance : instances) {
71             log.info(instance.getServiceId() + "\t" + instance.getHost() + "\t" +
instance.getPort() + "\t" + instance.getUri());
72         }
73         return discoveryClient;
74     }
75
76 }

```

### 5.1.2. 修改启动类

cloud-provider-payment8001模块:

```

1 package cn.sitedev.springcloud;
2
3 import org.springframework.boot.SpringApplication;
4 import org.springframework.boot.autoconfigure.SpringBootApplication;
5 import org.springframework.cloud.client.discovery.EnableDiscoveryClient;
6 import org.springframework.cloud.netflix.eureka.EnableEurekaClient;
7
8 @SpringBootApplication
9 @EnableEurekaClient
10 @EnableDiscoveryClient
11 public class PaymentMain8001 {
12     public static void main(String[] args) {
13         SpringApplication.run(PaymentMain8001.class, args);

```

```
14     }
15 }
```

cloud-provider-payment8002模块:

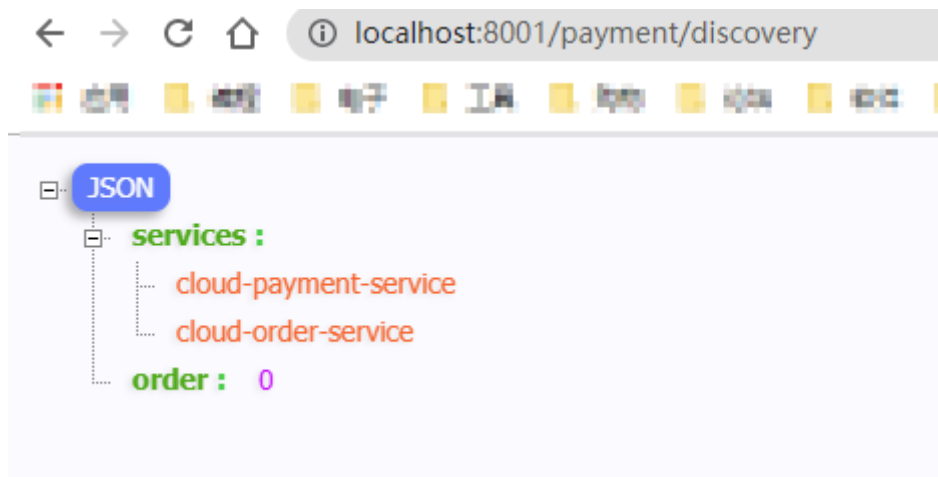
```
1 package cn.sitedev.springcloud;
2
3 import org.springframework.boot.SpringApplication;
4 import org.springframework.boot.autoconfigure.SpringBootApplication;
5 import org.springframework.cloud.client.discovery.EnableDiscoveryClient;
6 import org.springframework.cloud.netflix.eureka.EnableEurekaClient;
7
8 @SpringBootApplication
9 @EnableEurekaClient
10 @EnableDiscoveryClient
11 public class PaymentMain8002 {
12     public static void main(String[] args) {
13         SpringApplication.run(PaymentMain8002.class, args);
14     }
15 }
```

## 5.2. 测试

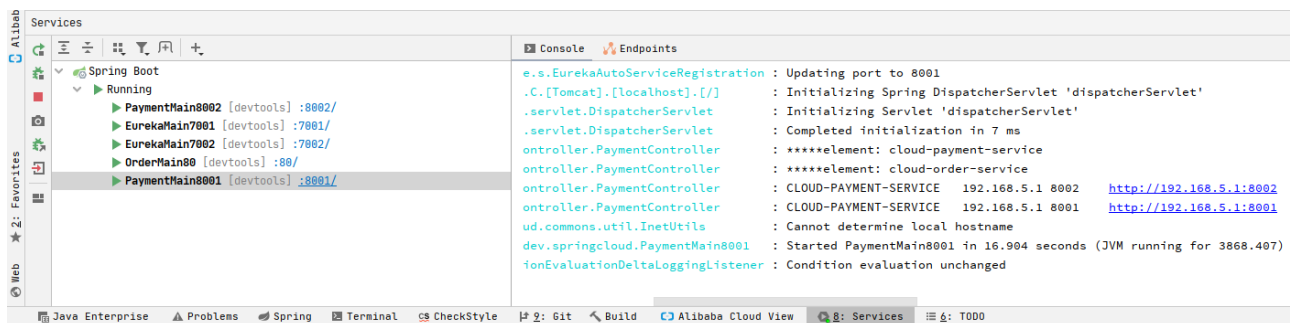
先要启动EurekaServer

再启动8001主启动类,需要稍等一会

浏览器访问<http://localhost:8001/payment/discovery>



查看控制台输出



## 6. Eureka自我保护

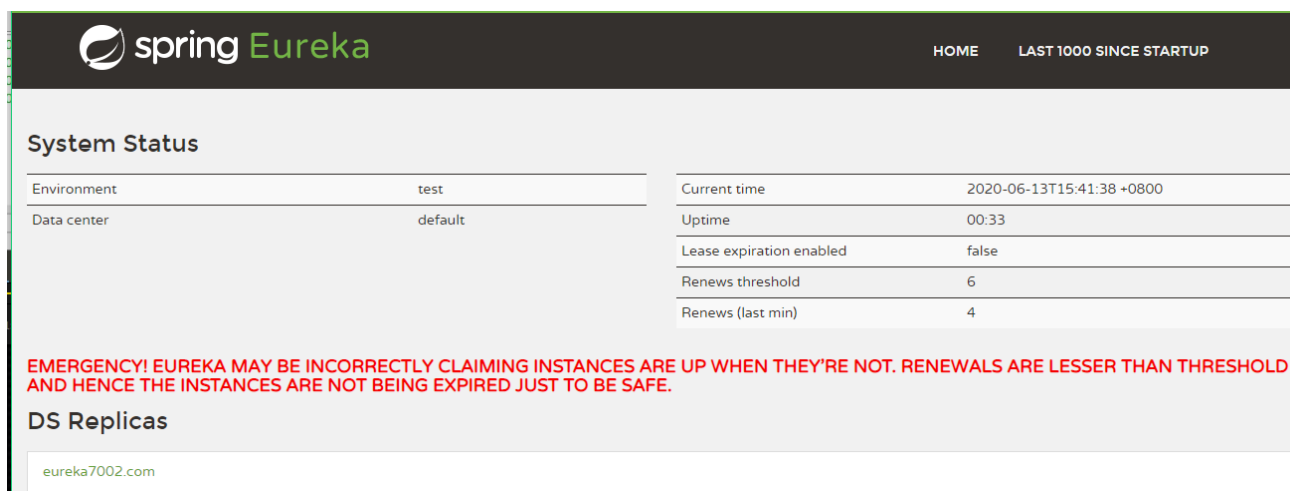
### 6.1. 故障现象

概述:

保护模式主要用于一组客户端和 Eureka Server之间存在网络分区场景下的保护。一旦进入保护模式, Eureka Server将会尝试保护其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。

如果在 Eureka Server的首页看到以下这段提示, 则说明 Eureka进入了保护模式:

**EMERGENCY! EUREKA MAY BE INCORRECTLY CLAIMING INSTANCES ARE UP WHEN THEY'RE NOT. RENEWALS ARE LESSER THAN THRESHOLD AND HENCE THE INSTANCES ARE NOT BEING EXPIRED JUST TO BE SAFE.**



### 6.2. 问题原因

为什么会产生 Eureka自我保护机制

为了防止 EurekaClient可以正常运行, 但是与 EurekaServer网络不通情况下, EurekaServer不会立刻将 EurekaClient服务剔除

什么是白我保护模式?

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，Eureka Server将会注销该实例（默认90秒）。但是当网络分区故障发生(延时、卡顿、拥挤) 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



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。

一句话:某时刻一个微服务不可用了,Eureka不会立刻清理,依旧会对该服务的信息进行保存  
属于CAP里面的AP分支

## 6.3. 怎样禁止自我保护

### 6.3.1. 注册中心eurekaServer端7001

#### 6.3.1.1. 默认配置

自我保护机制默认开启

```
1 eureka.server.enable-self-preservation=true
```

#### 6.3.1.2. 修改配置

修改主配置

```
1 eureka:
2   instance:
3     # eureka服务端的实例名称
4     hostname: eureka7001.com
```


```

5  client:
6      # false表示不向注册中心注册自己
7      register-with-eureka: false
8      # false表示自己端就是注册中心,我的职责就是维护服务实例,并不需要检索服务
9      fetch-registry: false
10     service-url:
11         # 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
12         defaultZone: http://eureka7002.com:7002/eureka/
13  server:
14      # 是否启用自我保护机制,为false时关闭,保证不可用服务能被及时剔除
15      enable-self-preservation: false
16      eviction-interval-timer-in-ms: 2000

```

### 6.3.1.3. 测试

浏览器访问<http://eureka7001.com:7001/>


HOME
LAST 1000 SINCE STARTUP

#### System Status

Environment	test	Current time	2020-06-13T22:08:25 +0800
Data center	default	Uptime	07:00
		Lease expiration enabled	true
		Renews threshold	6
		Renews (last min)	0

THE SELF PRESERVATION MODE IS TURNED OFF. THIS MAY NOT PROTECT INSTANCE EXPIRY IN CASE OF NETWORK/OTHER PROBLEMS.

#### DS Replicas

eureka7002.com

#### Instances currently registered with Eureka

Application	AMIs	Availability Zones	Status
CLOUD-ORDER-SERVICE	n/a (1)	(1)	UP (1) - localhost:cloud-order-service:80
CLOUD-PAYMENT-SERVICE	n/a (2)	(2)	UP (2) - payment8002, payment8001

可以看到如下文字提示:

**THE SELF PRESERVATION MODE IS TURNED OFF. THIS MAY NOT PROTECT INSTANCE EXPIRY IN CASE OF NETWORK/OTHER PROBLEMS.**

### 6.3.2. 生产者客户端eurekaClient端8001

#### 6.3.2.1. 默认配置

Eureka客户端向服务端发送心跳的时间间隔,单位为秒(默认是30秒)

```
1 eureka.instance.lease-renewal-interval-in-seconds=30
```

Eureka服务端在收到最后一次心跳后等待时间上限,单位为秒(默认是90秒),超时剔除服务

```
1 eureka.instance.lease-expiration-duration-in-seconds=90
```

### 6.3.2.2. 修改配置

修改主配置:

```
1 eureka:
2   client:
3     # 表示是否将自己注册进EurekaServer, 默认为true
4     register-with-eureka: true
5     # 是否从EurekaServer抓取已有的注册信息, 默认为true
6     # 单节点无所谓, 集群必须设置为true才能配合ribbon使用负载均衡
7     fetch-registry: true
8     service-url:
9       defaultZone:
10    http://eureka7001.com:7001/eureka,http://eureka7002.com:7002/eureka
11  instance:
12    instance-id: payment8001
13    prefer-ip-address: true # 访问路径可以显示IP地址
14    # Eureka客户端向服务端发送心跳的时间间隔, 单位:s.默认30s
15    lease-renewal-interval-in-seconds: 1
16    # Eureka服务端在收到最后一次心跳后等待时间上限.单位:s.默认90s.超时剔除服务
17    lease-expiration-duration-in-seconds: 2
```

### 6.3.2.3. 测试

关闭生产者客户端eurekaClient端8001

浏览器访问<http://eureka7001.com:7001/>



### System Status

Environment	test	Current time	2020-06-13T22:15:10 +0800
Data center	default	Uptime	07:07
		Lease expiration enabled	true
		Renews threshold	6
		Renews (last min)	6

RENEWALS ARE LESSER THAN THE THRESHOLD. THE SELF PRESERVATION MODE IS TURNED OFF. THIS MAY NOT PROTECT INSTANCE EXPIRY IN CASE OF NETWORK/OTHER PROBLEMS.

### DS Replicas

eureka7002.com

### Instances currently registered with Eureka

Application	AMIs	Availability Zones	Status
CLOUD-ORDER-SERVICE	n/a (1)	(1)	UP (1) - localhost:cloud-order-service:80
CLOUD-PAYMENT-SERVICE	n/a (1)	(1)	UP (1) - payment8002

可以看到如下文字提示:

**RENEWALS ARE LESSER THAN THE THRESHOLD. THE SELF PRESERVATION MODE IS TURNED OFF. THIS MAY NOT PROTECT INSTANCE EXPIRY IN CASE OF NETWORK/OTHER PROBLEMS.**

并且页面上展示的实例中没有了payment8001