

服务注册与发现

2020年7月24日 10:31

Eureka服务注册与发现

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

为什么要使用服务治理

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，所以要使用服务治理，管理服务与服务器之间的关系，可以实现服务调用，负载均衡，容错等，实现服务发现与注册

什么是服务注册与发现

Eureka采用CS的设计架构，Eureka作为服务注册功能的服务器，他是服务注册中心。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，会把当前自己的服务器的信息，比如服务地址通讯地址等以别名的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获取到实际的服务通讯地址，然后在实现本地RPC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））

Eureka

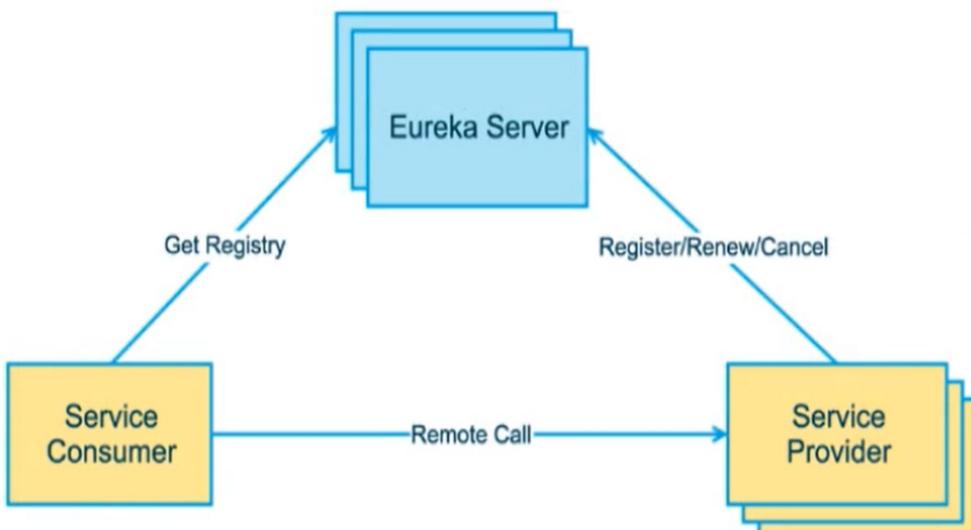
包含两个组件：Eureka Server和Eureka Client

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Eureka Server中进行注册，这样Eureka Server中的服务注册表会将存储所有可用服务节点的信息，服务节点的信息可以在页面中直观看到

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）



以前是直接80调用8001，现在80先调用Eureka Server

Eureka搭建步骤

1、idea生成Eureka Server端服务注册中心（类似物业公司）

建module

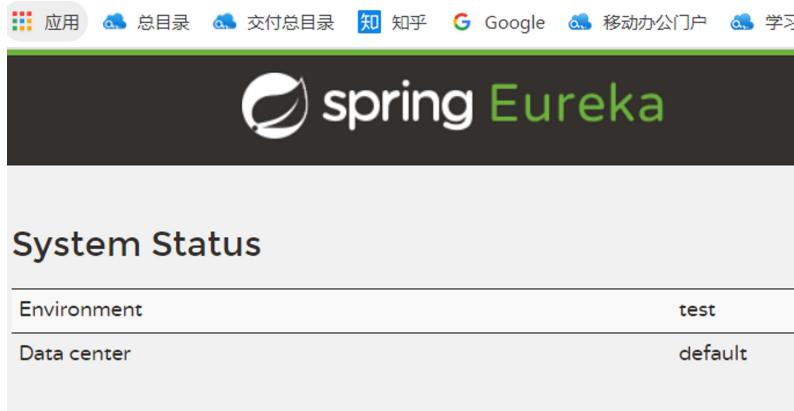
改pom

写YML

主启动

业务类

localhost:7001



Environment	test
Data center	default

支付微服务8001入驻eureka server (7001)

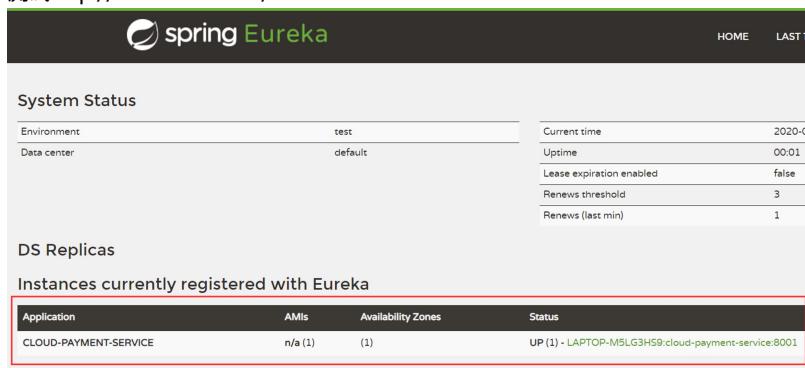
修改8001

引入eureka client依赖，改pom

改yml，添加eureka

启动类增加@EnableEurekaClient注解

测试http://localhost:7001/



Environment	test
Data center	default

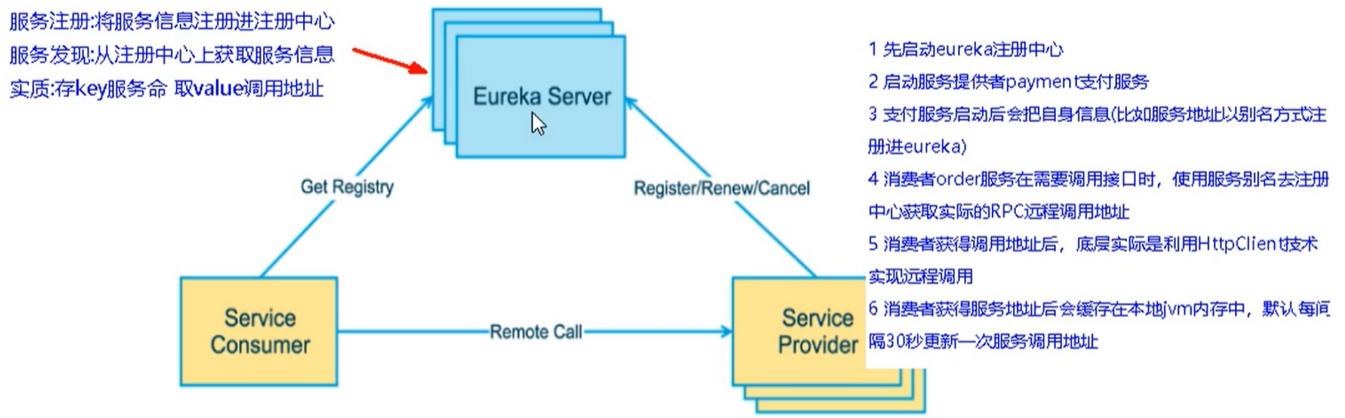
Current time	2020-07-01 00:01
Uptime	00:01
Lease expiration enabled	false
Renews threshold	3
Renews (last min)	1

Application	AMIs	Availability Zones	Status
CLOUD-PAYMENT-SERVICE	n/a (1)	(1)	UP (1) - LAPTOP-M5LG3H59:cloud-payment-service:8001

订单微服务80入驻eureka server (7001)

与上面操作相同

集群Eureka搭建步骤



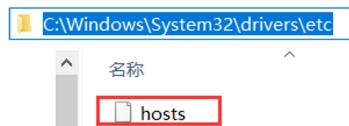
集群之间互相注册

1注册23, 2注册13...

搭建7002, 大致相同

yml换为集群注册

找到C:\Windows\System32\drivers\etc



System Status

Environment	test
Data center	default

DS Replicas

eureka7002.com

Instances currently registered with Eureka

80, 8001改yml

#集群版

defaultZone:

<http://eureka7001.com:7001/eureka/> <http://eureka7002.com:7002/eureka/>

DS Replicas

eureka7001.com

Instances currently registered with Eureka

Application	AMIs	Availability Zones
CLOUD-ORDER-SERVICE	n/a (1)	(1)
CLOUD-PAYMENT-SERVICE	n/a (1)	(1)

```
There was an unexpected error (type=Internal Server Error, status=500).
I/O error on GET request for "http://CLOUD-PAYMENT-SERVICE/payment/
org.springframework.web.client.ResourceAccessException: I/O error on GE
java.net.UnknownHostException: CLOUD-PAYMENT-SERVICE
    at org.springframework.web.client.RestTemplate.doExecute(RestTemp
    at org.springframework.web.client.RestTemplate.execute(RestTempl
    at org.springframework.web.client.RestTemplate.getForObject(RestTe
    at com.dali.springcloud.controller.OrderController.getPayment(Order
```

出现这个原因是不知道去指定哪一个服务

需要使用一个注解赋予RestTemplate负载均衡的能力@LoadBalanced

默认方式：轮询

```
@Bean
@LoadBalanced
//使用@LoadBalanced赋予RestTemplate负载均衡的能力
public RestTemplate getRestTemplate() { return new RestTemplate(); }
```

微服务信息完善

1、服务名称修改 2、访问信息提示IP信息

```
instance:
  instance-id: payment8002
  prefer-ip-address: true #访问路径可以显示IP地址
```

服务名称修改



The screenshot shows the Eureka dashboard with two service instances listed under the 'INSTANCES' tab. The first instance is 'UP (1) - windows10.microdone.cn:cloud-order-service:80'. The second instance is 'UP (2) - payment8002, payment8001'. A red arrow points to the second instance.

显示IP地址



The screenshot shows the Eureka dashboard with the URL '192.168.11.1:8001/actuator/info' in the address bar. The page displays various metrics and configuration details for the payment service.

服务发现Discovery (80想看服务注册中心有哪些提供者 (8001, 8002那些))

对于注册进eureka的微服务，可以通过服务发现来获得该服务的信息

修改8001中的controller

启动类

自测

```
: ****element:cloud-payment-service
: ****element:cloud-order-service
: payment8001 192.168.11.1 8001 http://192.168.11.1:8001
: payment8002 192.168.11.1 8002 http://192.168.11.1:8002
```

eureka自我保护机制



The screenshot shows the Eureka dashboard with a warning message: '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.' Below the message, it shows 'DS Replicas' and 'eureka7001.com'.

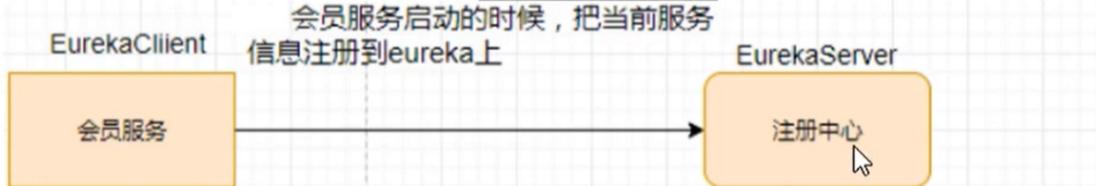
导致的原因：某时某刻一个微服务不能用了。eureka不会立即清理，依旧会对该微服务的信息进行保存

为什么会产生这种机制

为了eurekaclient可以正常运行，在eurekaserver网络不同的情况下，eurekaserver不会立刻将client剔除

什么是自我保护机制

默认情况下server (eurekaserver) 在一定时间内没有接收到某个微服务实例的心跳，server就会注销该实例（默认90秒）。但发生网络故障时。微服务与server不能正常通信。但微服务本身是健康的，此时不应该注销这个微服务。eureka通过自我保护机制来解决这个问题。---当server节点在短时间丢失过多的客户端时，那么这个节点就会进入自我保护机制



自我保护机制：默认情况下EurekaClient定时向EurekaServer端发送心跳包

如果Eureka在server端在一定时间内(默认90秒)没有收到EurekaClient发送心跳包，便会直接从服务注册列表中剔除该服务，但是在短时间（90秒中）内丢失了大量的服务实例心跳，这时候EurekaServer会开启自我保护机制，不会剔除该服务（该现象可能出现在如果网络不通但是EurekaClient为出现宕机，此时如果换做别的注册中心如果一定时间内没有收到心跳会将剔除该服务，这样就出现了严重失误，因为客户端还能正常发送心跳，只是网络延迟问题，而保护机制是为了解决此问题而产生的）

spring cloud整合ZK替代Eureka

pom引入

```
<!--SpringBoot整合Zookeeper客户端-->
<dependency>
    <groupId>org.springframework.cloud</groupId>
    <artifactId>spring-cloud-starter-zookeeper-discovery</artifactId>
    <exclusions>
        <!--先排除自带的zookeeper3.5.3-->
        <exclusion>
            <groupId>org.apache.zookeeper</groupId>
            <artifactId>zookeeper</artifactId>
        </exclusion>
    </exclusions>
</dependency>
```

yml修改

```
#服务别名 --- 注册zookeeper到注册中心名称
spring:
  application:
    name: cloud-provider-payment
  cloud:
    zookeeper:
      connect-string: 192.168.11.130:2181
```

controller

```
@RequestMapping(value = "/payment/zk")
public String paymentzk() {
    return "springcloud with zookeeper:" + serverPort + "\t" + UUID.randomUUID().toString();
}
```

localhost:8004/payment/zk

应用 总目录 交付总目录 知识 Google 移动办公门户 学习计划 bili

springcloud with zookeeper:8004 6efa93d0-69bb-4ce7-9f29-0cbce04a85b4

如果出现jar包冲突，首先排除zk自带版本

```
<exclusions>
    <!--先排除自带的zookeeper3.5.3-->
    <exclusion>
        <groupId>org.apache.zookeeper</groupId>
        <artifactId>zookeeper</artifactId>
    </exclusion>
</exclusions>
```

添加你服务器的版本3.4.7。总之这里的版本要小于你的服务器版本号否则报错

```
<!--添加zookeeper3.4.7版本-->
<dependency>
    <groupId>org.apache.zookeeper</groupId>
    <artifactId>zookeeper</artifactId>
    <version>3.4.7</version>
</dependency>
```

下图说明我们已经入驻zk服务

```
[zk: localhost:2181(CONNECTED) 2] ls /services
[cloud-provider-payment]
```

如果不规定名称那么就会默认application，建议写上

```
#服务别名 --- 注册zookeeper到注册中心名称
spring:
  application:
    name: cloud-provider-payment
```

```
[zk: localhost:2181(CONNECTED) 3] ls /services
[cloud-provider-payment, application]
```

这里的json串代表入驻(8004)的服务信息

```
[zk: localhost:2181(CONNECTED) 7] get /services/cloud-provider-payment/6e95d1d9-6bbf-4341-b608-826793e499c5
{"name":"cloud-provider-payment","id":"6e95d1d9-6bbf-4341-b608-826793e499c5","address":"windows10.microdone.cn","port":8004,"sslPort":null,"payload":{"@class":"org.springframework.cloud.zookeeper.discovery.ZookeeperInstance","id":"application-1","name":"cloud-provider-payment","metadata":{},"registrationTimeUTC":1595668479319,"serviceType":"DYNAMIC","uriSpec":{"parts":[{"value":"scheme","variable":true},{"value":"://","variable":false},{"value":"address","variable":true},{"value":":","variable":false},{"value":"port","variable":true}]}}
```

ZK没有自我保护机制

80入驻ZK

和80入驻eureka类似

下图表示入驻成功

```
[zk: localhost:2181(CONNECTED) 11] ls /services
[cloud-provider-payment, cloud-consumer-order]
```

consul

是什么

Consul 是一套开源的分布式服务发现和配置管理系统，由 HashiCorp 公司用 Go 语言开发。

提供了微服务系统中的服务治理、配置中心、控制总线等功能。这些功能中的每一个都可以根据需要单独使用，也可以一起使用以构建全方位的服务网格，总之Consul提供了一种完整的服务网格解决方案。

它有很多优点。包括：基于 raft 协议，比较简洁；支持健康检查，同时支持 HTTP 和 DNS 协议 支持跨数据中心的 WAN 集群 提供图形界面 跨平台，支持 Linux、Mac、Windows

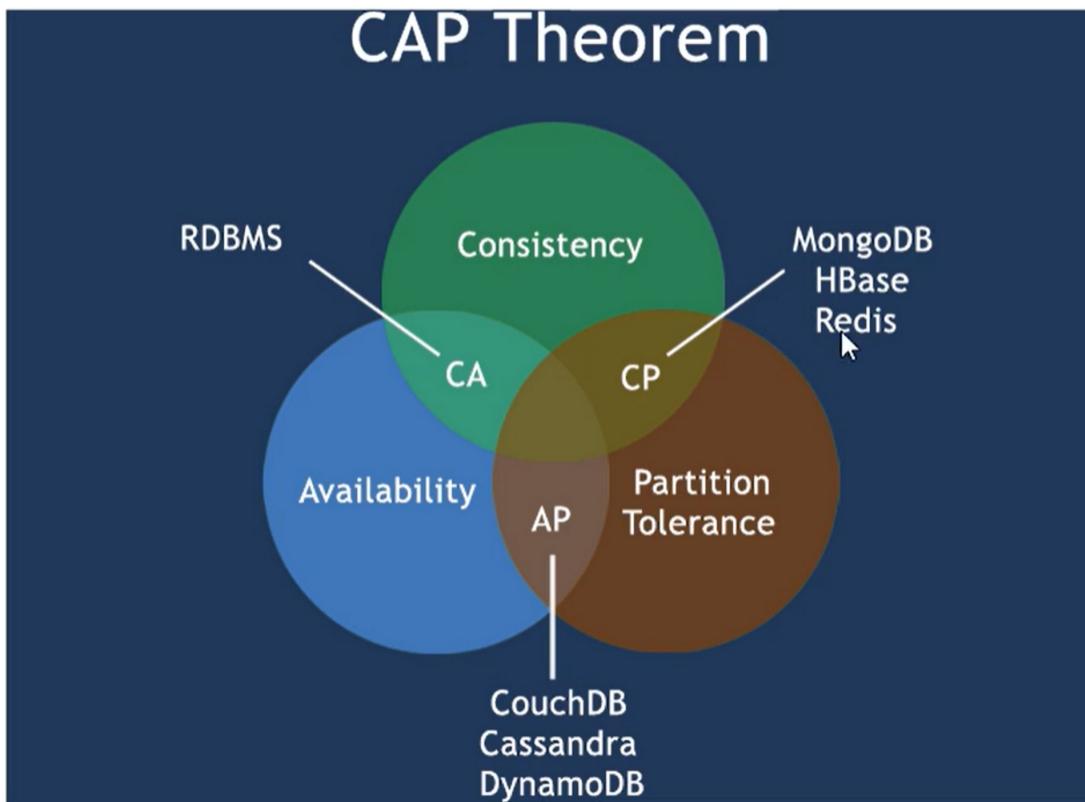
能干什么



和eureka和zk一样

下图为三者的区别

组件名	语言	CAP	服务健康检查	对外暴露接口	Spring Cloud集成
Eureka	Java	AP	可配支持	HTTP	已集成
Consul	Go	CP	支持	HTTP/DNS	已集成
Zookeeper	Java	CP	支持	客户端	已集成



c: 强一致性

a: 高可用性

p: 分区容错性 (分布式)

AP: eureka。没心跳不会杀死，有自我保护机制

CP: zk, consul。没心跳立刻杀死