**[SpringCloud之Eureka注册中心原理及其搭建](https://www.cnblogs.com/jing99/p/11576133.html)**

**一、Eureka简介**

　　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。SpringCloud将它集成在其子项目spring-cloud-netflix中，以实现SpringCloud的服务发现功能。

**1、Eureka组件**

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

**1.1 Eureka Server**

　　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 Server本身也是一个服务，默认情况下会自动注册到Eureka注册中心。  
　　**如果搭建单机版的Eureka Server注册中心，则需要配置取消Eureka Server的自动注册逻辑。**毕竟当前服务注册到当前服务代表的注册中心中是一个说不通的逻辑。  
　　Eureka Server通过**Register、Get、Renew**等接口提供服务的**注册、发现和心跳检测**等服务。

**2.1 Eureka Client**

　　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 Client分为两个角色，分别是：Application Service(Service Provider)和Application Client(Service Consumer)

**2.1.1 Application Service**

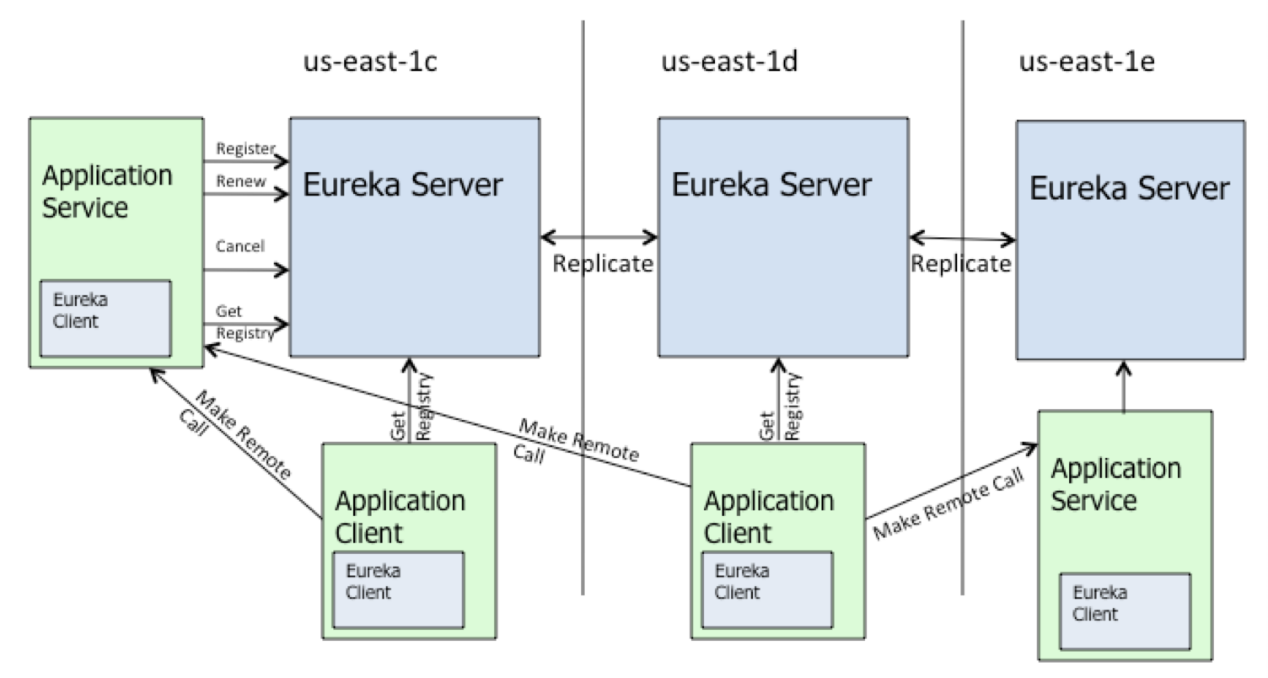
　　服务提供方，是注册到Eureka Server中的服务。

**2.1.2 Application Client**

　　服务消费方，通过Eureka Server发现服务，并消费。

　　在这里，Application Service和Application Client不是绝对上的定义，因为Provider在提供服务的同时，也可以消费其他Provider提供的服务；Consumer在消费服务的同时，也可以提供对外服务。

**2、Eureka Server架构原理简介**



Register(服务注册)：把自己的IP和端口注册给Eureka。  
Renew(服务续约)：发送心跳包，每30秒发送一次。告诉Eureka自己还活着。  
Cancel(服务下线)：当provider关闭时会向Eureka发送消息，把自己从服务列表中删除。防止consumer调用到不存在的服务。  
Get Registry(获取服务注册列表)：获取其他服务列表。  
Replicate(集群中数据同步)：eureka集群中的数据复制与同步。  
Make Remote Call(远程调用)：完成服务的远程调用。

　　Eureka Server  
　　Eureka Server既是一个注册中心，同时也是一个服务。那么搭建Eureka Server的方式和以往搭建Dubbo注册中心ZooKeeper的方式必然不同，那么首先搭建一个单机版的Eureka Server注册中心。

**二、搭建单机版Eureka Server**

　　Eureka已经被Spring Cloud继承在其子项目spring-cloud-netflix中，搭建Eureka Server的方式还是非常简单的。只需要通过一个独立的maven工程即可搭建Eureka Server。pom依赖如下：

[复制代码](javascript:void(0);)

<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 http://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>1.5.16.RELEASE</version>

</parent>

<groupId>com.bjsxt</groupId>

<artifactId>spring-cloud-eureka-server-single</artifactId>

<version>1.0</version>

<properties>

<project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>

<project.reporting.outputEncoding>UTF-8</project.reporting.outputEncoding>

<java.version>1.8</java.version>

</properties>

**<dependencyManagement>**

**<dependencies>**

**<dependency>**

**<groupId>org.springframework.cloud</groupId>**

**<artifactId>spring-cloud-dependencies</artifactId>**

**<version>Edgware.SR4</version>**

**<type>pom</type>**

**<scope>import</scope>**

**</dependency>**

**</dependencies>**

**</dependencyManagement>**

<dependencies>

<dependency>

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

<artifactId>spring-boot-starter-web</artifactId>

</dependency>

<dependency>

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

<artifactId>spring-boot-starter-test</artifactId>

<scope>test</scope>

</dependency>

**<!-- spring cloud 默认配置启动器 -->**

**<dependency>**

**<groupId>org.springframework.cloud</groupId>**

**<artifactId>spring-cloud-starter-config</artifactId>**

**</dependency>**

**<!-- spring cloud Eureka Server 启动器 -->**

**<dependency>**

**<groupId>org.springframework.cloud</groupId>**

**<artifactId>spring-cloud-starter-eureka-server</artifactId>**

**</dependency>**

</dependencies>

<build>

<plugins>

<plugin>

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

<artifactId>spring-boot-maven-plugin</artifactId>

</plugin>

</plugins>

</build>

</project>

[复制代码](javascript:void(0);)

　　全局配置文件：

　　而Eureka Server本身也是一个服务，同时又是一个注册中心。在Spring Cloud中，启动的微服务会自动的搜索注册中心并注册服务，那么在单机版Eureka Server环境中，当前服务注册到当前服务中，明显是不合适的。所以搭建Eureka Server单机版时，需要提供特殊的全局配置，避免回路注册逻辑。  
　　同理，Eureka Server服务在注册中心中发现服务列表逻辑也是不必要的。毕竟注册中心是一个中立的服务管理平台，如果是单机版Eureka Server环境中，Eureka Server服务再去发现服务列表，明显也是不必要的。也需要通过全局配置，避免回路发现逻辑。

[复制代码](javascript:void(0);)

# 设置spring应用命名，可以自定义，非必要

spring.application.name=eureka-server

# 设置Eureka Server WEB控制台端口，自定义

server.port=**8761**

#是否将自己注册到Eureka-Server中，默认的为true

eureka.client.**registerWithEureka**=false

#是否从Eureka-Server中获取服务注册信息，默认为true

eureka.client.**fetchRegistry**=false

[复制代码](javascript:void(0);)

　　启动类配置：  
　　启动Eureka Server注册中心，和普通的SpringBoot应用的启动没有太大的区别。只需要在启动类上增加@EnableEurekaServer注解，来开启Eureka Server服务即可。  
　　注意：此处@SpringCloudApplication注解定义启动类。@SpringCloudApplication注解定义启动类涉及到hystrix相关内容。

[复制代码](javascript:void(0);)

@EnableEurekaServer

@SpringBootApplication

public class EurekaServerApplication {

public static void main(String[] args) {

SpringApplication.run(**EurekaServerApplication**.class, args);

}

}

[复制代码](javascript:void(0);)

　　访问Eureka Server WEB控制台：通过IP和端口，使用浏览器访问即可查看Eureka Server中的信息。本例中访问地址为：http://localhost:8761/

**三、搭建集群版Eureka Server**

　　注册中心作为微服务架构中的核心功能，其重要性不言而喻。所以单机版的Eureka Server在可靠性上并不符合现在的互联网开发环境。集群版的Eureka Server才是商业开发中的选择。

　　Eureka Server注册中心的集群和Dubbo的ZooKeeper注册中心集群在结构上有很大的不同。

　　ZooKeeper注册中心集群搭建后，集群中各节点呈现主从关系，集群中只有主节点对外提供服务的注册和发现功能，从节点相当于备份节点，只有主节点宕机时，从节点会选举出一个新的主节点，继续提供服务的注册和发现功能。

　　而Eureka Server注册中心集群中每个节点都是平等的，集群中的所有节点同时对外提供服务的发现和注册等功能。同时集群中每个Eureka Server节点又是一个微服务，也就是说，每个节点都可以在集群中的其他节点上注册当前服务。又因为每个节点都是注册中心，所以**节点之间又可以相互注册当前节点中已注册的服务，并发现其他节点中已注册的服务**。所以Eureka Server注册中心集群版在搭建过程中有很多的方式，找到一个最合适最可靠的搭建方式才能称为一个称职的程序员。

　　集群版Eureka Server可以通过Spring Boot多环境配置方式快速搭建。只要创建一个合适的Eureka Server工程，通过多个全局配置即可完成快速搭建。

　　本案例中搭建一个双节点的Eureka Server集群。

　　　　Linux版本为： CentOS 6.5  
　　　　JDK版本为： 1.8

　　POM依赖

　　　　和单机版Eureka Server相同。

　　全局配置文件

　　本例中的两个节点分别会搭建在两个Linux系统中，为这两个Linux系统分别定义域名为eurekaserver1和eurekaserver2。

　　在集群搭建过程中，全局配置文件的定义非常重要。其中euraka.client.serviceUrl.defaultZone属性是用于配置集群中其他节点的。如果有多个节点，使用逗号','分隔。

　　有部分程序员只配置某一个集群节点信息，通过集群节点间的注册通讯实现节点的全面发现。这种配置形式是不推荐的。因为**Eureka Server在服务管理上，会根据连带责任来维护服务列表，如果某集群节点宕机，那么通过这个节点注册过来的服务都会连带删除**。

[复制代码](javascript:void(0);)

#eurekaserver1配置

spring.application.name=eureka-server

server.port=8761

# **设置eureka实例名称，建议与配置文件的变量相同，必须和Linux系统域名相同**

eureka.instance.hostname=eurekaserver1

# 设置服务注册中心地址，指向另一个注册中心，使用域名作为访问路径

eureka.client.serviceUrl.**defaultZone**=http://eurekaserver2:8761/eureka/

#eurekaserver2配置

spring.application.name=eureka-server

server.port=8761

eureka.instance.hostname=eurekaserver2

eureka.client.serviceUrl.defaultZone=http://eurekaserver1:8761/eureka/

[复制代码](javascript:void(0);)

　　打包工程形成jar文件  
　　使用run -> maven install即可实现打包过程。打包后的jar文件保存在工程中的target目录中，并上传打包后的jar文件到Linux系统。

　　设置Linux主机域名

　　修改/etc/hosts文件，设置主机域名。将主机域名和IP进行绑定。新增内容如下：两个Linux系统修改内容相同。

192.168.2.115 eurekaserver1

192.168.2.116 eurekaserver2

　　使用命令启动Eureka Server

　　可以在Linux终端中，通过java命令来启动Eureka Server。在启动的时候，可以通过启动参数来设置有效的配置环境。具体命令如下：

java -jar **-Dspring.profiles.active=eurekaserver1** spring-cloud-eureka-server-cluster-1.0.jar

　　其中-Dspring.profiles.active启动参数，用于定义本次启动的Eureka Server应用的有效全局配置命名，也就是全局配置文件的后缀。SpringBOOT在启动的时候，会根据启动参数来决定读取的有效全局配置文件是哪一个。

　　也可以定义一个shell文件来简化操作。具体shell内容如下：

[复制代码](javascript:void(0);)

#!/bin/bash

cd `dirname $0`

CUR\_SHELL\_DIR=`pwd`

CUR\_SHELL\_NAME=`basename ${BASH\_SOURCE}`

JAR\_NAME="项目jar包名称"

JAR\_PATH=$CUR\_SHELL\_DIR/$JAR\_NAME

#JAVA\_MEM\_OPTS=" -server -Xms1024m -Xmx1024m -XX:PermSize=128m"

JAVA\_MEM\_OPTS=""

SPRING\_PROFILES\_ACTIV="-Dspring.profiles.active=配置文件变量名称"

#SPRING\_PROFILES\_ACTIV=""

LOG\_DIR=$CUR\_SHELL\_DIR/logs

LOG\_PATH=$LOG\_DIR/${JAR\_NAME%..log

echo\_help()

{

echo -e "syntax: sh $CUR\_SHELL\_NAME start|stop"

}

if [ -z $1 ];then

echo\_help

exit 1

fi

if [ ! -d "$LOG\_DIR" ];then

mkdir "$LOG\_DIR"

fi

if [ ! -f "$LOG\_PATH" ];then

touch "$LOG\_DIR"

fi

if [ "$1" == "start" ];then

# check server

PIDS=`ps --no-heading -C java -f --width 1000 | grep $JAR\_NAME | awk '{print $2}'`

if [ -n "$PIDS" ]; then

echo -e "ERROR: The $JAR\_NAME already started and the PID is ${PIDS}."

exit 1

fi

echo "Starting the $JAR\_NAME..."

# start

nohup java $JAVA\_MEM\_OPTS -jar $SPRING\_PROFILES\_ACTIV $JAR\_PATH >> $LOG\_PATH 2>&1 &

COUNT=0

while [ $COUNT -lt 1 ]; do

sleep 1

COUNT=`ps --no-heading -C java -f --width 1000 | grep "$JAR\_NAME" | awk '{print $2}' | wc -l`

if [ $COUNT -gt 0 ]; then

break

fi

done

PIDS=`ps --no-heading -C java -f --width 1000 | grep "$JAR\_NAME" | awk '{print $2}'`

echo "${JAR\_NAME} Started and the PID is ${PIDS}."

echo "You can check the log file in ${LOG\_PATH} for details."

elif [ "$1" == "stop" ];then

PIDS=`ps --no-heading -C java -f --width 1000 | grep $JAR\_NAME | awk '{print $2}'`

if [ -z "$PIDS" ]; then

echo "ERROR:The $JAR\_NAME does not started!"

exit 1

fi

echo -e "Stopping the $JAR\_NAME..."

for PID in $PIDS; do

kill $PID > /dev/null 2>&1

done

COUNT=0

while [ $COUNT -lt 1 ]; do

sleep 1

COUNT=1

for PID in $PIDS ; do

PID\_EXIST=`ps --no-heading -p $PID`

if [ -n "$PID\_EXIST" ]; then

COUNT=0

break

fi

done

done

echo -e "${JAR\_NAME} Stopped and the PID is ${PIDS}."

else

echo\_help

exit 1

fi

[复制代码](javascript:void(0);)

　　设置好shell启动脚本后，需要提供可执行权限：shell脚本文件名自己修改：chmod 755 xxx.sh

　　脚本使用方式为：

#启动Eureka Server

./xxx.sh start

#关闭Eureka Server

./xxx.sh stop

**四、Eureka Server安全认证**

　　Eureka Server作为Spring Cloud中的服务注册中心，如果可以任意访问的话，那么其安全性太低。所以Spring Cloud中也有为Eureka Server提供安全认证的方式。可以使用spring-boot-starter-security组件来为Eureka Server增加安全认证。

　　POM依赖：

<!-- spring boot security安全认证启动器 -->

<dependency>

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

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

</dependency>

　　修改全局配置文件，在全局配置文件中，开启基于http basic的安全认证。

[复制代码](javascript:void(0);)

# eurekaserver1配置

spring.application.name=eureka-server

server.port=8761

eureka.instance.hostname=eurekaserver1

# 使用http basic安全认证语法，在集群通信中增加认证信息。 **http://用户名:密码@地址:端口/eureka/**

eureka.client.serviceUrl.defaultZone=http://test:123456@eurekaserver2:8761/eureka/

# 开启基于http basic的安全认证

security.basic.enabled=true

# 设置安全认证用户名

security.user.name=test

# 设置安全认证密码

security.user.password=123456

# eurekaserver2配置

spring.application.name=eureka-server

server.port=8761

eureka.instance.hostname=eurekaserver2

eureka.client.serviceUrl.defaultZone=http://test:123456@eurekaserver1:8761/eureka/

security.basic.enabled=true

security.user.name=test

security.user.password=123456

[复制代码](javascript:void(0);)

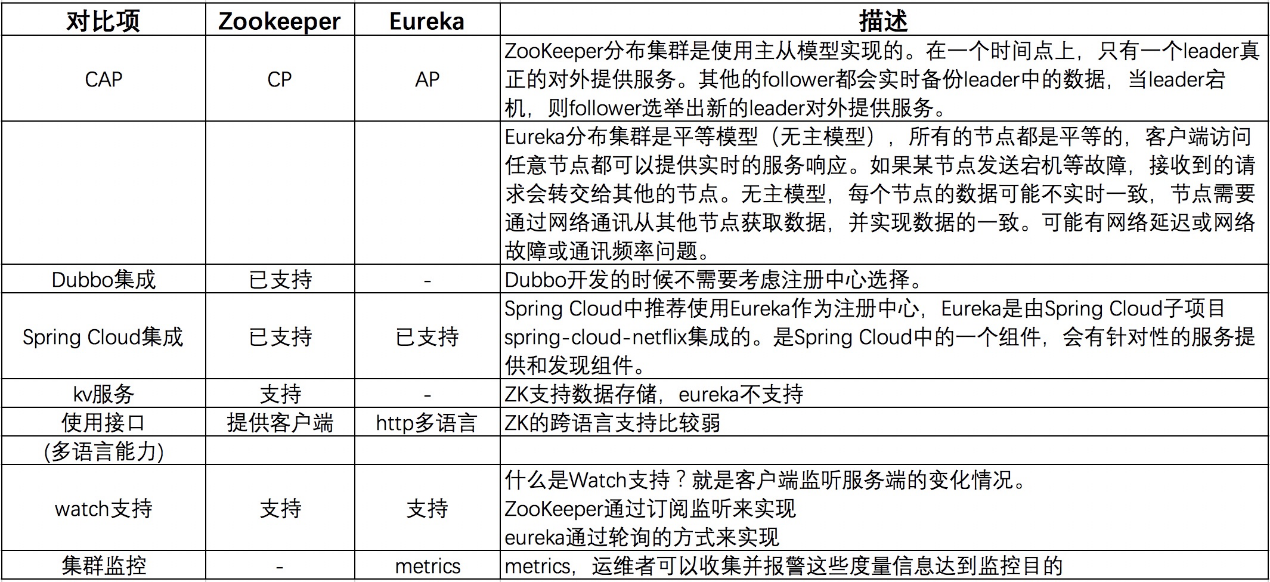
**五、CAP定理**

　　CAP原则又称CAP定理，指的是在一个分布式系统中，Consistency（数据一致性）、 Availability（服务可用性）、Partition tolerance（分区容错性），三者不可兼得。CAP由Eric Brewer在2000年PODC会议上提出。该猜想在提出两年后被证明成立，成为我们熟知的CAP定理。

|  |  |
| --- | --- |
| **分布式系统CAP定理** | |
| 数据一致性  (Consistency) | 数据一致性(Consistency)  也叫做数据原子性系统在执行某项操作后仍然处于一致的状态。在分布式系统中，更新操作执行成功后所有的用户都应该读到最新的值，这样的系统被认为是具有强一致性的。等同于所有节点访问同一份最新的数据副本。  优点： 数据一致，没有数据错误可能。  缺点： 相对效率降低。 |
| 服务可用性  (Availablity) | 每一个操作总是能够在一定的时间内返回结果，这里需要注意的是"一定时间内"和"返回结果"。一定时间内指的是，在可以容忍的范围内返回结果，结果可以是成功或者是失败。 |
| 分区容错性  (Partition-torlerance) | 在网络分区的情况下，被分隔的节点仍能正常对外提供服务(分布式集群，数据被分布存储在不同的服务器上，无论什么情况，服务器都能正常被访问) |

|  |  |
| --- | --- |
| **定律：任何分布式系统只可同时满足二点，没法三者兼顾。** | |
| CA，放弃P | 如果想避免分区容错性问题的发生，一种做法是将所有的数据(与事务相关的)/服务都放在一台机器上。虽然无法100%保证系统不会出错，但不会碰到由分区带来的负面效果。当然这个选择会严重的影响系统的扩展性。 |
| CP，放弃A | 相对于放弃"分区容错性"来说，其反面就是放弃可用性。一旦遇到分区容错故障，那么受到影响的服务需要等待一定时间，因此在等待时间内系统无法对外提供服务。 |
| AP，放弃C | 这里所说的放弃一致性，并不是完全放弃数据一致性，而是放弃数据的强一致性，而保留数据的最终一致性。以网络购物为例，对只剩下一件库存的商品，如果同时接受了两个订单，那么较晚的订单将被告知商品告罄。 |

**Eureka和ZooKeeper的特性**

**六、Eureka Client**

　　在Spring Cloud中，开发Eureka Client组件还是非常方便的。我们在开发过程中，不需要像Dubbo那样关注服务的角色。无论是Provider还是Consumer都是一个微服务客户端，只是在编码层面上，服务消费者代码比较麻烦。

**1、Application Service服务提供者开发**

　　POM依赖：  
　　如果Eureka 开启了security安全校验机制，那么Eureka Client在开发的时候，依赖的jar包需要额外依赖一个actuator插件。那么可以修改Eureka Client工程中的启动器，把spring-cloud-starter-eureka改为spring-cloud-starter-eureka-server。Eureka-server启动器中包含HTTP Basic安全认证相关的jar包，实际上spring-cloud-starter-eureka-server启动器包含启动器spring-cloud-starter-eureka中的所有的包。

[复制代码](javascript:void(0);)

<!-- spring cloud Eureka Client 启动器，因为Eureka Server开启了安全校验，

所有需要依赖更大范围的jar包资源spring-cloud-starter-eureka-server-->

<dependency>

<groupId>org.springframework.cloud</groupId>

<artifactId>spring-cloud-starter-eureka-server</artifactId>

</dependency>

<!-- actuator 组件是Spring Boot的监控组件，actuator一旦应用，在启动的时候，会发布一系列的URL服务。包含一个shutdown服务，代表优雅关闭

当Spring Boot 应用中的actuator组件接收到shutdown请求的时候，会触发优雅关闭。

如果当前应用中有Eureka Client的集成，则会触发Eureka Client向Eureka Server发起一个shutdown优雅停服的请求 -->

<dependency>

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

<artifactId>spring-boot-actuator</artifactId>

</dependency>

[复制代码](javascript:void(0);)

　　全局配置：

　　在配置Eureka Server信息时，建议将Eureka Server集群中的所有节点依次配置，Eureka Client在注册服务的时候，会根据节点列表依次访问Eureka Server集群节点，只要注册成功，后续Eureka Server节点不再访问注册。虽然Eureka Server集群各节点可以相互发现服务，但是Eureka Server集群中每个节点对服务的管理都使用连带责任，及从某Eureka Server节点发现服务A，如果这个Eureka Server节点宕机，则A服务同时从服务列表中删除。

[复制代码](javascript:void(0);)

# 定义SpringBoot应用的名称，建议必须提供。在SpringCloud中，对服务的最大粒度的管理是使用应用命名的

# 最好是一个应用一个名称，在Consumer角色开发的时候，比较容易查找Provider

spring.application.name=eureka-application-service

server.port=8081

# 配置Eureka Server的地址信息，如果是Eureka Server集群，多个节点使用逗号','分割。

# 如果开启了安全认证，使用HTTP Bacic格式提供用户名和密码。

# 如果Eureka Server是一个集群，那么配置Eureka Server节点信息的时候，建议将所有的Eureka Server节点信息都配置上

# 实际上，只配置一个Eureka Server节点其实就可以了，但是，Eureka Server对服务的管理有连带责任。如果只配置一个Eureka Server节点，那么会导致级联删除的风险，可能导致服务不可靠

# 如果配置了多个Eureka Server节点，Eureka不会将当期的服务同时注册到所有Eureka Server节点上

# 从第一个配置的Eureka Server节点开始注册，如果注册成功，后续的Eureka Server节点不再重复注册

# 每30秒，Eureka Client发送一个心跳到Eureka Server上，如果心跳没有反馈，则从已配置的Eureka Server节点列表的下一个服务节点继续注册。

# 这样做可以保证服务的可靠性，降低服务连带责任导致的服务不可靠。

# 如果多个Eureka Client需要注册，建议Eureka Server的服务列表顺序是随机排列的。

# 如：有Eureka Server s1,s2,s3，有Eureka Client c1,c2,c3。

# 那么在c1上配置的Eureka Server列表建议是s1,s2,s3，在c2上配置的是s2,s3,s1，在c3上配置的是s3,s1,s2，这样可以更好的利用Eureka Server集群的特性。

# 因为Eureka Server和Eureka Client对心跳的监测都是3\*间隔时间的，所以会有服务列表数据的不同步可能。

# 所以在CAP原则上，Eureka Server是保证AP原则，放弃C原则的。

eureka.client.serviceUrl.defaultZone=http://eurekaserver1:111111@eurekaserver1:8761/eureka/,http://eurekaserver2:222222@eurekaserver2:8761/eureka/

security.basic.enabled=true

# 启用shutdown，优雅停服功能，配置actuator的优雅关闭

# actuator 组件监听shutdown请求地址的时候，要求请求的method必须是POST

# shutdown的请求地址是使用：@PostMapping或@RequestMapping(method=RequestMethod.POST)

endpoints.shutdown.enabled=true

# 禁用密码验证

endpoints.shutdown.sensitive=false

[复制代码](javascript:void(0);)

　　建议：如果有多个服务功能需要注册，那么在设置Eureka Server信息的时候，推荐异序排列。如：现在有3个工程A、B、C需要注册服务到Eureka Server集群中，集群节点有三个，分别是e1、e2、e3，那么在工程中推荐配置为，A工程配置-e1,e2,e3，B工程配置e2,e3,e1，C工程配置e3,e1,e2。这样可以更好的利用Eureka Server集群的特性。

　　启动类：  
　　需要在启动类上新增注解@EnableEurekaClient，代表当前应用开启Eureka客户端，应用启动后，会自动将服务注册到Eureka Server中。

[复制代码](javascript:void(0);)

/\*\*

\* Eureka Client启动类

\* 是使用SpringBoot启动类实现启动的

\* @EnableEurekaClient 注解，就是用于通知SpringBoot应用，当前应用是一个Irk客户端

\* 需要做一下服务的注册。

\* 使用全局配置文件application.properties配置Eureka Server的相关信息

\* 如：Eureka Server的地址，个数，是否需要安全认证等。

\*/

@EnableEurekaClient

@SpringBootApplication

public class EurekaApplicationServiceApplication {

public static void main(String[] args) {

SpringApplication.run(EurekaApplicationServiceApplication.class, args);

}

}

[复制代码](javascript:void(0);)

　　对外接口：

　　Eureka Client的Application Server对外需要暴露接口方法，接口定义如Rest请求定义方式：

[复制代码](javascript:void(0);)

@Controller

public class TestApplicationServiceController {

@RequestMapping(value="/test")

@ResponseBody

public List<Map<String, Object>> test(){

List<Map<String, Object>> result = new ArrayList<>();

for(int i = 0; i < 3; i++){

Map<String, Object> data = new HashMap<>();

data.put("id", i+1);

data.put("name", "test name " + i);

data.put("age", 20+i);

result.add(data);

}

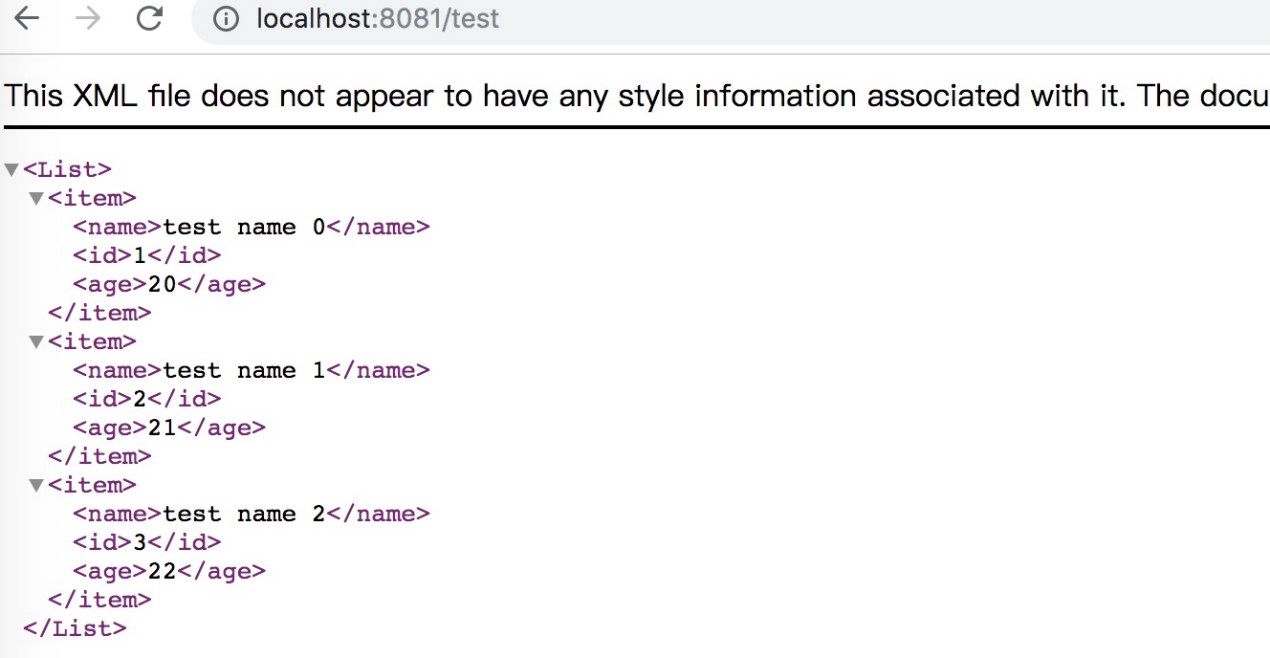
return result;

}

}

[复制代码](javascript:void(0);)

　　因此服务也可以作为url请求直接调用（Security Basic安全认证导致返回结果为XML类型，而不是JSON类型）：



**2、Application Client服务消费者开发**

　　在Spring Cloud中，服务消费方代码的开发确实比较麻烦，并不像Dubbo那么直接注入服务接口代理对象，通过代理对象方法直接访问远程服务。在Spring Cloud中，微服务的提供是通过REST风格提供的，也就是服务的调用是基于HTTP协议的，所以在服务调用上比较麻烦，具体详见案例代码。

　　POM依赖：

　　同Application Service工程。

　　全局配置：

　　因为都是本地启动，需要修改服务端口。推荐修改spring应用命名。在Eureka Server中，对服务的管理是基于spring应用名称的，所以不同的服务推荐使用不同的应用名称。

[复制代码](javascript:void(0);)

spring.application.name=eureka-application-client

server.port=8080

# 点对点直连是不发现服务，不是不注册服务。

# 任何Eureka Client都必须注册。如果没有配置Eureka Server节点列表，则注册失败。Eureka client无法正常启动。

eureka.client.serviceUrl.defaultZone=http://eurekaserver1:111111@eurekaserver1:8761/eureka/,http://eurekaserver2:222222@eurekaserver2:8761/eureka/

# 设置负载均衡策略 eureka-application-service为调用的服务的名称

# 没有配置全部服务的负载均衡策略的方式。因为不是每个服务都可以使用相同负载均衡策略的。

# 如：搜索服务和注册服务就不能使用相同的负载均衡策略。

eureka-application-service.ribbon.NFLoadBalancerRuleClassName=com.netflix.loadbalancer.RandomRule

# 关闭ribbon访问注册中心Eureka Server发现服务

ribbon.eureka.enabled=false

# 配置服务列表，其中eureka-application-service代表要访问的服务的应用名，如果有多个服务结点组成集群，多个节点的配置信息使用逗号','分隔。

# 配置服务列表，需要配置要调用的服务的名字和服务所在的位置。

# 服务的名字，就是Application Service中配置的spring.application.name。

# 服务的位置，就是服务的所在ip和端口。

# 如果服务位置有多个，也就是服务集群，那么使用逗号','分割多个服务列表信息。

eureka-application-service.ribbon.listOfServers=localhost:8083

[复制代码](javascript:void(0);)

　　启动类：

　　同Applicaiton Service一致。

　　请求调用类：

[复制代码](javascript:void(0);)

/\*\*

\* 在这里开发Eureka Client中的Application Client角色。就是consumer服务的消费者。

\* **服务消费者需要在注册中心中发现服务列表的。且同时将自己注册到注册中心的服务列表中。（参考如下截图）**

\*

\* consumer在消费provider的时候，是通过LoadBalancer来实现的。

\* **LoadBalancer简介 ： 是Eureka client内置的一个负载均衡器。复杂在发现的服务列表中选择服务应用，获取服务的IP和端口。**

\* 实现服务的远程调用。

\*

\* application client代码开发相比较dubbo的consumer开发麻烦很多。

\*

\*/

@RestController

public class TestApplicationClientController {

/\*\*

\* ribbon负载均衡器，其中记录了从Eureka Server中获取的所有服务信息。

\* 这些服务的信息是IP和端口等。应用名称，域名，主机名等信息。

\*/

@Autowired

private LoadBalancerClient loadBalancerClient;

/\*\*

\* 通过HTTP协议，发起远程服务调用，实现一个远程的服务消费。

\* @return

\*/

@GetMapping

public List<Map<String, Object>> test() {

// 通过spring应用命名，获取服务实例ServiceInstance对象

// ServiceInstance 封装了服务的基本信息，如 IP，端口

/\*

\* 在Eureka中，对所有注册到Eureka Server中的服务都称为一个service instance服务实例。

\* 一个服务实例，就是一个有效的，可用的，provider单体实例或集群实例。

\* 每个service instance都和spring application name对应。

\* 可以通过spring application name查询service instance

\*/

ServiceInstance si =

this.loadBalancerClient.choose("eureka-application-service");

// 拼接访问服务的URL

StringBuilder sb = new StringBuilder();

// http://localhost:8081/test

sb.append("http://").append(si.getHost())

.append(":").append(si.getPort()).append("/test");

System.out.println("本次访问的service是： " + sb.toString());

// SpringMVC RestTemplate，用于快速发起REST请求的模板对象。

/\*

\* RestTemplate是SpringMVC提供的一个用于发起REST请求的模板对象。

\* 基于HTTP协议发起请求的。

\* 发起请求的方式是exchange。需要的参数是： URL, 请求方式， 请求头， 响应类型，【URL rest参数】。

\*/

RestTemplate rt = new RestTemplate();

/\*

\* 创建一个响应类型模板。

\* 就是REST请求的响应体中的数据类型。

\* ParameterizedTypeReference - 代表REST请求的响应体中的数据类型。

\*/

ParameterizedTypeReference<List<Map<String, Object>>> type =

new ParameterizedTypeReference<List<Map<String, Object>>>() {

};

/\*

\* ResponseEntity:封装了返回值信息，相当于是HTTP Response中的响应体。

\* 发起REST请求。

\*/

ResponseEntity<List<Map<String, Object>>> response =

rt.exchange(sb.toString(), HttpMethod.GET, null, type);

/\*

\* ResponseEntity.getBody() - 就是获取响应体中的java对象或返回数据结果。

\*/

List<Map<String, Object>> result = response.getBody();

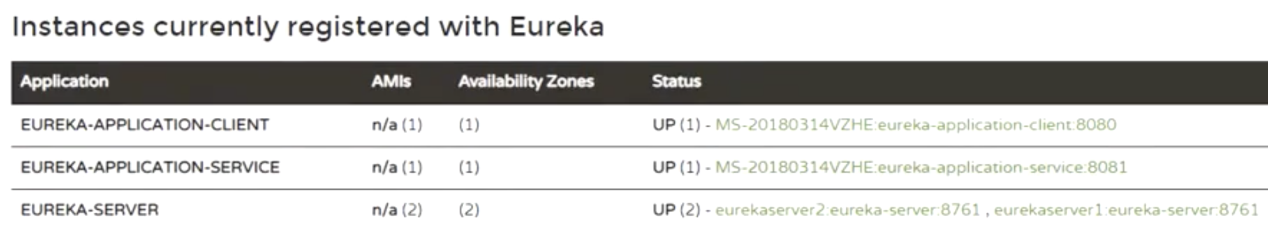
return result;

}

}

[复制代码](javascript:void(0);)

　　LoadBanlancerClient中包含了所有的服务注册信息，如下图示例：



**七、服务保护**

**1 服务保护模式**

　　服务保护模式（自我保护模式）：一般情况下，微服务在Eureka上注册后，会每30秒发送心跳包，Eureka通过心跳来判断服务时候健康，同时会定期删除超过90秒没有发送心跳服务。

　　导致Eureka Server接收不到心跳包的可能：一是微服务自身的原因，二是微服务与Eureka之间的网络故障。通常微服务的自身的故障只会导致个别服务出现故障，一般不会出现大面积故障，而网络故障通常会导致Eureka Server在短时间内无法收到大批心跳。虑到这个区别，Eureka设置了一个阀值，当判断挂掉的服务的数量超过阀值时，Eureka Server认为很大程度上出现了网络故障，将不再删除心跳过期的服务。

　　那么这个阀值是多少呢？Eureka Server在运行期间，会统计心跳失败的比例在15分钟内是否低于85%，如果低于85%，Eureka Server则任务是网络故障，不会删除心跳过期服务。

　　这种服务保护算法叫做Eureka Server的服务保护模式。

　　这种不删除的，90秒没有心跳的服务，称为无效服务，但是还是保存在服务列表中。如果Consumer到注册中心发现服务，则Eureka Server会将所有好的数据（有效服务数据）和坏的数据（无效服务数据）都返回给Consumer。

**2 服务保护模式的存在必要性**

　　因为同时保留"好数据"与"坏数据"总比丢掉任何数据要更好，当网络故障恢复后，Eureka Server会退出"自我保护模式"。

　　Eureka还有客户端缓存功能(也就是微服务的缓存功能)。即便Eureka Server集群中所有节点都宕机失效，微服务的Provider和Consumer都能正常通信。

　　微服务的负载均衡策略会自动剔除死亡的微服务节点（**Robbin**）。

　　只要Consumer不关闭，缓存始终有效，直到一个应用下的所有Provider访问都无效的时候，才会访问Eureka Server重新获取服务列表。

**3 关闭服务保护模式**

　　可以通过全局配置文件来关闭服务保护模式，商业项目中不推荐关闭服务保护，因为网络不可靠很容易造成网络波动、延迟、断线的可能。如果关闭了服务保护，可能导致大量的服务反复注册、删除、再注册。导致效率降低。在商业项目中，服务的数量一般都是几十个，大型的商业项目中服务的数量可能上百、数百，甚至上千：

# 关闭自我保护:true为开启自我保护，false为关闭自我保护

eureka.server.enableSelfPreservation=false

# 清理间隔(单位:毫秒，默认是60\*1000)，当服务心跳失效后多久，删除服务。

eureka.server.eviction.interval-timer-in-ms=60000

**4 优雅关闭服务（优雅停服）**

　　在Spring Cloud中，可以通过HTTP请求的方式，通知Eureka Client优雅停服，这个请求一旦发送到Eureka Client，那么Eureka Client会发送一个shutdown请求到Eureka Server，Eureka Server接收到这个shutdown请求后，会在服务列表中标记这个服务的状态为down，同时Eureka Client应用自动关闭。这个过程就是优雅停服。

　　如果使用了优雅停服，则不需要再关闭Eureka Server的服务保护模式。

　　POM依赖：  
　　优雅停服是通过Eureka Client发起的，所以需要在Eureka Client中增加新的依赖，这个依赖是autuator组件，添加下述依赖即可。

<dependency>

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

<artifactId>spring-boot-actuator</artifactId>

</dependency>

　　修改全局配置文件：

　　Eureka Client默认不开启优雅停服功能，需要在全局配置文件中新增如下内容：

# 启用shutdown，优雅停服功能

endpoints.shutdown.enabled=true

# 禁用密码验证

endpoints.shutdown.sensitive=false

　　发起shutdown请求：

　　必须通过POST请求向Eureka Client发起一个shutdown请求。请求路径为：http://ip:port/shutdown。可以通过任意技术实现，如：HTTPClient、form表单，AJAX等。

　　建议使用优雅停服方式来关闭Application Service/Application Client服务。