# 1.什么是微服务

**1.** 通常而言，微服务架构是一种架构模式或者说是一种架构风格，他提倡将单一应用程序划分为一组小的服务，每个服务运行在自己独立的进程中，服务之间互相协调，互相配合，为用户提供最终价值，

**2.** 服务之间采用轻量级的通信机制互相沟通通常是基于Http的RESTful Api

springcloud 和dubbo的第一个区别就是dubbo采用rpc 调用，传输的是二级制

**3.** 每个服务都采用围绕着具体业务进行构建，并且能够被独立的部署到生产环境，类生产环境等，另外，应尽量避免统一的，集中式的服务管理机制，对具体的服务而言，根据业务上下文，选择合适的语言。工具对其进行构建，可以有一个非常轻量级的集中式管理来协调这些服务，可以使用不同的语言编写服务，也可以使用不同的数据存储

## 从技术角度

微服务化的核心就是将传统的一站式应用，根据业务拆分成一个一个的服务，彻底的取出耦合，每一个微服务提供单个业务功能的服务，一个服务干一件事，从技术角度看就是一个**小而独立的处理过程** 类似进程概念，能独立的启动或者销毁，拥有自己独立的数据库

**读马丁福勒论文**

# 2.微服务和微服务架构

## 微服务

强调的是大小，他关注的是某个点，他具体解决了某一个问题/提供落地对应服务的一个服务应用，侠义的看，可以看作是Eclipse 里面的一个微服务工程，/或者是Module

## 微服务架构

**微服务架构**是一种架构模式，他体长将单一的应用程序划分成一个小的服务，服务之间互相调用，互相配合，为用户提供最终的价值，每个服务运行在单独的进程中，服务于服务之间采用轻量级的通信机制互相协作，**通常是http协议 restful api** 每个服务都围绕这具体的业务进行构建，并且能被独立的部署到生产环境，类生产环境，另外，**应当尽量避免统一的，集中式的服务机制**，对具体的一个服务而言，根据业务上下文，选择何使的语言，工具进行构建

# 3.微服务技术栈

多种技术的集合，我们再讨论一个分布式的微服务架构的话，我们需要有哪些**维度**？ ？？

服务的治理

服务的注册

服务的调用

服务的负载均衡

服务的监控

dubbo zookeeper 。。。。

springCloud 直接把这些相关的进行整合

| 微服务条目(接口) | 落地技术（实现类） |
| --- | --- |
| 服务开发 | springboot spring springmvc。。。 |
| 服务配置与管理 | Netflix 公司的Archaius 阿里的Diamond |
| 服务注册发现 | Eureka Consul zookeeper |
| 服务的调用 | Rest RPC gRPC |
| 服务熔断器 | Hystrix Envoy |
| 负载均衡 | Ribbon Nginx |
| 服务接口调用 | Feign。。。 |
| 消息队列 | Kafka RabbitMQ ActiveMQ |
| 服务配置中心管理 | SpringCloudConfig Chef |
| 服务路由 | Zuul |
| 服务监控 | Zabbix Nagios Metrics Spectator |
| 全链路追踪 | Zipkin Brave Dapper |
| 事件消息总线 | SpringCloud Bus |
| 服务部署 | Docker OpenStack Kubernetes |

服务流操作开发包 SpringCloud Stream 封装与redis rabbit kafka 等发送接收消息

# 为什么选择SpringCloud 作为微服务架构

微服务架构不仅仅有springCloud 这一种实现，

# 选型依据

1. 整体解决方案和框架成熟度
2. 社区热度
3. 可维护性
4. 学习曲线

   it公司使用的微服务架构有哪些？

   阿里**Dubbo /HSF**

dubbo停止更新了， 2012 年不在维护，梁非团队对被打散，

2016年 阿里对Dubbo进行重新维护 正是这5年springCloud 发展时机

这个期间阿里团队使用HSF相当于dubbo 的二代吧****

**京东JSF服务治理工具**

**新浪微博Motan**

**当当网DubboX**

但是当当网被卖了

# 各微服务框架对比

| 功能点 | **NetFlix/Spring Cloud** | **Motan** | **Google/gRPC** | **Thrift** | **Dubbo/DubboX** |
|---|---|---|---|---|---|
| 功能定位 | 完整的微服务框架 | RPC框架，但是整合了ZK或Consul 实现集群环境的服务注册发现 | RPC框架 | Rpc框架 | 服务框架 |
| 支持Rest | 是 Ribbon支持多种可插拔的序列化选择 | 否 | 否 | 否 | 否 |
| 支持RPC多语言 | 否 | 是（Hession2） | 是 | 是 | 是 |
| 支持多语言 | 是（Rest形式）？ | 否 | 是 | 是 | 否 |
| 服务器注册发现 | 是（Eureka）Eureka服务注册表，Karyon服务端框架支持服务自注册和健康检查 | 是（zookeeper/consul） | 否 | 否 | 是 |

| 功能点 | NetFlix/Spring Cloud | Motan | Google/gRPC | Thrift | Dubbo/DubboX |
|---|---|---|---|---|---|
| 负载均衡 | 是（服务器zuul+客户端Ribbon）Zuul-服务，动态路由，云端负载均衡Eureka 针对中间层服务器 | 是 | 否 | 否 | 是客户端 |
| 配置服务 | Netflix Archaius Spring 是cloud Config server 集中配置 | 是zookeeper 提供 | 否 | 否 | 否 |
| 服务调用链监控 | 是zuul zuul提供边缘服务API网管 | 否 | 否 | 否 | 否 |
| 高可用/容错 | 是（服务端Hystrix+客户端Ribbon） | 是（客户端） | 否 | 否 | 是（客户端） |
| 典型应用案例 | Netflix | Sina | Google | Facebook | |
| 社区活跃度 | 高 | 一般 | 高 | 一般 | 2012不维护<br>2017 维护 |
| 学习难度 | 中等 | 低 | 高 | 高 | 低 |

| 功能点 | NetFlix/Spring Cloud | Motan | Google/gRPC | Thrift | Dubbo/DubboX |
|---|---|---|---|---|---|
| 文档丰富度 | 高 | 一般 | 一般 | 一般 | 高 |
| 其他 | spring Cloud Bus为我们应用程序带来更多的管理端点 | 支持降级 | NetFlix内部在GRPC | IDL定义 | 实践公司多 |

为什么**springcloud** 流行了？

dubbo 睡觉睡了5 年

老系统用dubbo多， 新的用springcloud 多

# 4.SpringCloud 入门概述

## 1.官网说明

spring.io

1. SpringCloud ,是基于springboot 提供的一套微服务解决方案，包括服务注册发现，配置中心，全链路监控，服务网关，负载均衡，熔断器等组件，除了基于NetFlix的开源组件做了高亮抽象封装之外，哈有一些选型中立的开源组件
2. springCloud利用SpringBoot的开发遍历性轻巧的简化了分布式系统的基础设施开发，SpringCloud为开发人员提供了快速构建分布式系统的一些工具，包括配置管理，服务发现，断路器，路由，微代理，事件总线，全局锁，决策竞选，分布式回话，都可以使用springboot的开发风格做到一键启动部署
3. SpringBoot并没有重复制造轮子，他只是将目前各家公司开发的比较成熟的，经得起考验的服务架构组合起来，通过Springboot风格在进行封装屏蔽了复杂的配置实现原理，**最终给开发这留下了一套简单易懂，易部署和易维护的分布式系统开发包**

SpringCloud=分布式微服务架构下的一站式解决方案，是各个微服务架构的落地的集合体，俗称微服务全家桶

## SpringCloud 和SpringBoot是什么关系？

Boot关注的是维观，Cloud是分布式微服务架构下的一站式解决方案，他关注的是宏观

boot 是一个一个科室 Cloud 是一个医院

Boot 不用以来Cloud Cloud 要用boot

1. **springboot** 关注于快速开发单个微服务

2. **springCloud**是关注全局微服务协调整理治理框架，他将**Springboot**开发的一个个单体微服务整合并管理起来，为各个微服务之间提供，配置管理，服务发现，断路器，路由，微代理，事件总线，全局锁，决策竞争，分布式会话等等集成服务
3. SpringBoot可以离开SpringCloud独立使用开发项目，但是**SpirngCloud**离不开**SpringBoot** 属于依赖关系，**SpringBoot**专注于快速，方便的开发单个微服务个体，**SpringCloud**关注全局的服务治理架构

# SpringCloud VS Dubbo

活跃度

通过gith

| | **Dubbo** | **SpringCloud** |
|---|---|---|
| 服务注册中心 | Zookeeper | Spring Cloud Netflix Eureka |
| 服务调用方式 | RPC | REST API |
| 服务监控 | Dubbo-monitor | Spring Boot Admin |
| 断路器 | 不完善 | Spring Cloud Netflix Hystrix |
| 服务网关 | 无 | Spring Cloud Netflix Zuul |
| 分布式配置 | 无 | Spring Cloud Config |
| 服务跟踪 | 无 | Spring Cloud Sleuth |
| 消息总线 | 无 | Spring Cloud Bus |
| 数据流 | 无 | Spring Cloud Stream |

批量任务 无 SpringCloud Task

ub 的活跃度分析SpringCloud 的活跃度比较高

**最大区别：** springCloud 排起了Dubbo的RPC 通信，采用的是基于HTTP 的REST方式

严格来说，两种方式各有优略，但是从一定程度上来说，后者牺牲了服务调用性能，但也避免了上面提到的远程RPC带来的问题，而且REST相比RPC更为灵活，服务提供方的调用和调用方的以来只能依靠一纸契约，不存在代码级别的强的依赖，这强调快速演变的微服务环境下，显得更加合适

**品牌和组装机区别**

很明显，SpringCloud的功能相比Dubbo更加强大，涵盖面更加广泛，而且作为Spring的拳头项目，也能够与 Spring Framework，SpringBoot ，Spirngdata ，SpringBatch 等其他的spring项目完美融合，这些对微服务 而言是至关重要的，使用Dubbo构建微服务就像是组装电脑，各环节我们的选择自由度很高，但是最终结果就很有可能因为一条内存质量不行就不亮了，总是让人不怎么放心，如果你是高手，呢这些都不是问题，而 SpringCloud 更像是品牌机，在SpringCloud 的整合下，做了很多的兼容测试，保证了机器拥有更高的稳定性，如果要在使用非原装组件外的东西，就要对其基础有足够的了解

**刘军，Dubbo重启维护开发的刘军说的话**

dubbo的定位始终是一款RPC框架，而SpringCloud 的目标是微服务架构的一战式解决方案，如果非要对比的话，Dubbo可以类比到SpringCloud中的NetflixOSS 技术栈而SpirngCloud集成了Nettflix OSS 最为分布式服务治理解决方案，但除此之外SpringCloud还提供了config stream security，sleuth 等分布式问题解决方案，当前由于RPC 协议，注册中心不匹配等问题，在面临微服务基础架构选型 时候经常Dubbo和SpringCloud是只能二选一，这一也是为什么大家总拿Dubbo和SpringCloud做对比的原因之一，Dubbo之后会积极的寻求适配到SpringCloud 生态，比如为SpringCloud的二进制方式来发挥Dubbo的性能优劣，或者Dubbo通过模块化对http的支持的适配到SpirngCloud

## 能干什么

springCloud 拿英国伦敦地铁站的地铁名来作为版本号，不是以数字来的

http://springcloud.cc/spring-cloud-dalston.html

SpringCloud API 直接看中文官网

SpringCloud 中文社区

SpringCloud 中文网

## SpringCloud 国内使用情况

国内公司

1.中国联通

2.阿里，阿里云 因为阿里的淘宝电商不能进行大的改动，所以没有使用springCloud，

# 5.REST微服务构建案例工程模块

**项目自动化编译部署**

清理-->编译-->测试-->打包-->部署

mvn clean -->mvn compile --> mvn test--> mvn package -->mvn install

规划版本

springboot 1.5

springcloud dalstonSR1

## 构建步骤

### 新建pom模块

```
<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>
```

```
    <groupId>com.atguigu.springcloud</groupId>
    <artifactId>microservicecloud</artifactId>
    <version>0.0.1-SNAPSHOT</version>
    <packaging>pom</packaging>


    <properties>
        <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
        <maven.compiler.source>1.8</maven.compiler.source>
        <maven.compiler.target>1.8</maven.compiler.target>
        <junit.version>4.12</junit.version>
        <log4j.version>1.2.17</log4j.version>
        <lombok.version>1.16.18</lombok.version>
    </properties>

    <dependencyManagement>
        <dependencies>
            <dependency>
                <groupId>org.springframework.cloud</groupId>
                <artifactId>spring-cloud-dependencies</artifactId>
                <version>Dalston.SR1</version>
                <type>pom</type>
                <scope>import</scope>
            </dependency>
            <dependency>
                <groupId>org.springframework.boot</groupId>
                <artifactId>spring-boot-dependencies</artifactId>
                <version>1.5.9.RELEASE</version>
                <type>pom</type>
                <scope>import</scope>
            </dependency>
            <dependency>
                <groupId>mysql</groupId>
                <artifactId>mysql-connector-java</artifactId>
                <version>5.0.4</version>
            </dependency>
            <dependency>
                <groupId>com.alibaba</groupId>
                <artifactId>druid</artifactId>
                <version>1.0.31</version>
            </dependency>
            <dependency>
                <groupId>org.mybatis.spring.boot</groupId>
                <artifactId>mybatis-spring-boot-
starter</artifactId>
                <version>1.3.0</version>
            </dependency>
            <dependency>
                <groupId>ch.qos.logback</groupId>
                <artifactId>logback-core</artifactId>
                <version>1.2.3</version>
            </dependency>
            <dependency>
```

```
                                <groupId>junit</groupId>
                                <artifactId>junit</artifactId>
                                <version>${junit.version}</version>
                                <scope>test</scope>
                        </dependency>
                        <dependency>
                                <groupId>log4j</groupId>
                                <artifactId>log4j</artifactId>
                                <version>${log4j.version}</version>
                        </dependency>
                </dependencies>
        </dependencyManagement>

        <build>
                <finalName>microservicecloud</finalName>
                <resources>
                        <resource>
                                <directory>src/main/resources</directory>
                                <filtering>true</filtering>
                        </resource>
                </resources>
                <plugins>
                        <plugin>
                                <groupId>org.apache.maven.plugins</groupId>
                                <artifactId>maven-resources-plugin</artifactId>
                                <configuration>
                                        <delimiters>
                                                <delimit>$</delimit>
                                        </delimiters>
                                </configuration>
                        </plugin>
                </plugins>
        </build>


        <modules>

        <module>microservicecloud-api</module>
        </modules>

</project>
```

## 创建api公共模块和部门Entity

新建工程microsevicecloud-api

```
<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><!-- 子类里面显示声明才能有明确的继承表现，无意外就是父类的默认版本否则
自己定义 -->
                <groupId>com.atguigu.springcloud</groupId>
                <artifactId>microservicecloud</artifactId>
                <version>0.0.1-SNAPSHOT</version>
        </parent>

        <artifactId>microservicecloud-api</artifactId><!-- 当前Module我自己叫什么名
字 -->

        <dependencies><!-- 当前Module需要用到的jar包，按自己需求添加，如果父类已经包含
了，可以不用写版本号 -->
                <dependency>
                        <groupId>org.projectlombok</groupId>
                        <artifactId>lombok</artifactId>
                </dependency>
                <dependency>
                        <groupId>org.springframework.cloud</groupId>
                        <artifactId>spring-cloud-starter-feign</artifactId>
                </dependency>
        </dependencies>



</project>
```

使用lombok 我写成了博客这就不说了

然后写完emp 之后

mvn clean

mvn install

形成最新的jar包

方便别的模块引用

## Rest微服务案例-部门提供者

**约定>配置>编码**

1. microservicecloud-provider-dept-8001
2. pom

```
<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>com.atguigu.springcloud</groupId>
                <artifactId>microservicecloud</artifactId>
                <version>0.0.1-SNAPSHOT</version>
        </parent>

        <artifactId>microservicecloud-provider-dept-8001</artifactId>

        <dependencies>
                <!-- 引入自己定义的api通用包，可以使用Dept部门Entity -->
                <dependency>
                        <groupId>com.atguigu.springcloud</groupId>
                        <artifactId>microservicecloud-api</artifactId>
                        <version>${project.version}</version>
                </dependency>
                <!-- actuator监控信息完善 -->
                <dependency>
                        <groupId>org.springframework.boot</groupId>
                        <artifactId>spring-boot-starter-actuator</artifactId>
                </dependency>
                <!-- 将微服务provider侧注册进eureka -->
                <dependency>
                        <groupId>org.springframework.cloud</groupId>
                        <artifactId>spring-cloud-starter-eureka</artifactId>
                </dependency>
                <dependency>
                        <groupId>org.springframework.cloud</groupId>
                        <artifactId>spring-cloud-starter-config</artifactId>
                </dependency>
                <dependency>
                        <groupId>junit</groupId>
                        <artifactId>junit</artifactId>
                </dependency>
                <dependency>
                        <groupId>mysql</groupId>
                        <artifactId>mysql-connector-java</artifactId>
                </dependency>
                <dependency>
                        <groupId>com.alibaba</groupId>
                        <artifactId>druid</artifactId>
                </dependency>
                <dependency>
                        <groupId>ch.qos.logback</groupId>
                        <artifactId>logback-core</artifactId>
                </dependency>
                <dependency>
                        <groupId>org.mybatis.spring.boot</groupId>
                        <artifactId>mybatis-spring-boot-starter</artifactId>
                </dependency>
                <dependency>
```

```
                        <groupId>org.springframework.boot</groupId>
                        <artifactId>spring-boot-starter-jetty</artifactId>
                </dependency>
                <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>
                </dependency>
                <!-- 修改后立即生效，热部署 -->
                <dependency>
                        <groupId>org.springframework</groupId>
                        <artifactId>springloaded</artifactId>
                </dependency>
                <dependency>
                        <groupId>org.springframework.boot</groupId>
                        <artifactId>spring-boot-devtools</artifactId>
                </dependency>
        </dependencies>

</project>
```

yml文件

```
server:
  port: 8001

mybatis:
  config-location: classpath:mybatis/mybatis.cfg.xml        # mybatis配置文件所在路
径
  type-aliases-package: com.atguigu.springcloud.entities    # 所有Entity别名类所在
包
  mapper-locations:
  - classpath:mybatis/mapper/**/*.xml                       # mapper映射文件

spring:
   application:
    name: microservicecloud-dept
   datasource:
    type: com.alibaba.druid.pool.DruidDataSource            # 当前数据源操作类型
    driver-class-name: org.gjt.mm.mysql.Driver              # mysql驱动包
    url: jdbc:mysql://localhost:3306/cloudDB01              # 数据库名称
    username: root
    password: 123456
    dbcp2:
      min-idle: 5                                           # 数据库连接池的最小维持
连接数
      initial-size: 5                                       # 初始化连接数
```

```
      max-total: 5                                                    # 最大连接数
      max-wait-millis: 200                                            # 等待连接获取的最大超时
时间

eureka:
  client: #客户端注册进eureka服务列表内
    service-url:
      #defaultZone: http://localhost:7001/eureka
       defaultZone:
http://eureka7001.com:7001/eureka/,http://eureka7002.com:7002/eureka/,http://eurek
a7003.com:7003/eureka/
  instance:
    instance-id: microservicecloud-dept8001
    prefer-ip-address: true     #访问路径可以显示IP地址

info:
  app.name: atguigu-microservicecloud
  company.name: www.atguigu.com
  build.artifactId: $project.artifactId$
  build.version: $project.version$
```

```
spring:
   application:
    name: microservicecloud-dept                                  #很重要
```

mybatis.cfg.xml

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE configuration
  PUBLIC "-//mybatis.org//DTD Config 3.0//EN"
  "http://mybatis.org/dtd/mybatis-3-config.dtd">

<configuration>

        <settings>
                <setting name="cacheEnabled" value="true" /><!-- 二级缓存开启 -->
        </settings>

</configuration>
```

dao

```
package com.atguigu.springcloud.dao;

import java.util.List;

import org.apache.ibatis.annotations.Mapper;

import com.atguigu.springcloud.Dept;

@Mapper
public interface DeptDao {

        public boolean addDept(Dept dept);

        public Dept findById(Long id);

        public List<Dept> findAll();

}
```

mapper

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE mapper PUBLIC "-//mybatis.org//DTD Mapper 3.0//EN"
"http://mybatis.org/dtd/mybatis-3-mapper.dtd">

<mapper namespace="com.atguigu.springcloud.dao.DeptDao">

        <select id="findById" resultType="Dept" parameterType="Long">
                select deptno,dname,db_source from dept where deptno=#{deptno};
        </select>
        <select id="findAll" resultType="Dept">
                select deptno,dname,db_source from dept;
        </select>
        <insert id="addDept" parameterType="Dept">
                INSERT INTO dept(dname,db_source) VALUES(#{dname},DATABASE());
        </insert>

</mapper>
```

```
package com.atguigu.springcloud.service;

import java.util.List;

import com.atguigu.springcloud.entities.Dept;
```

```
public interface DeptService
{
        public boolean add(Dept dept);

        public Dept get(Long id);

        public List<Dept> list();
}
```

service.impl

```
package com.atguigu.springcloud.service.impl;

import java.util.List;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;

import com.atguigu.springcloud.dao.DeptDao;
import com.atguigu.springcloud.entities.Dept;
import com.atguigu.springcloud.service.DeptService;

@Service
public class DeptServiceImpl implements DeptService {
        @Autowired
        private DeptDao dao;

        @Override
        public boolean add(Dept dept) {
                return dao.addDept(dept);
        }

        @Override
        public Dept get(Long id) {
                return dao.findById(id);
        }

        @Override
        public List<Dept> list() {
                return dao.findAll();
        }

}
```

controller

```
package com.atguigu.springcloud.controller;

import java.util.List;

import javax.ws.rs.GET;
import javax.ws.rs.POST;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.PathVariable;
import org.springframework.web.bind.annotation.PostMapping;
import org.springframework.web.bind.annotation.RestController;

import com.atguigu.springcloud.entities.Dept;
import com.atguigu.springcloud.service.DeptService;

/**
 * 前后端分离
 * @author 27660
 *
 */
@RestController
public class DeptController {

	@Autowired
	private DeptService deptService ;

	@PostMapping("/dept/add")
	public boolean add(Dept dept){
		return deptService.add(dept);
	}

	@GetMapping("/dept/add/{id}")
	public Dept get(@PathVariable("id") Long id){
		return deptService.get(id);
	}

	@GetMapping("/dept/list")
	public List<Dept> list(){
		return deptService.list();
	}



}
```

## 构建消费者

创建microservicecloud-consumer-dept-80 模块

pom

```
<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>com.atguigu.springcloud</groupId>
                <artifactId>microservicecloud</artifactId>
                <version>0.0.1-SNAPSHOT</version>
        </parent>

        <artifactId>microservicecloud-consumer-dept-80</artifactId>
        <description>部门微服务消费者</description>

        <dependencies>
                <dependency><!-- 自己定义的api -->
                        <groupId>com.atguigu.springcloud</groupId>
                        <artifactId>microservicecloud-api</artifactId>
                        <version>${project.version}</version>
                </dependency>
                <!-- Ribbon相关 -->
                <dependency>
                        <groupId>org.springframework.cloud</groupId>
                        <artifactId>spring-cloud-starter-eureka</artifactId>
                </dependency>
                <dependency>
                        <groupId>org.springframework.cloud</groupId>
                        <artifactId>spring-cloud-starter-ribbon</artifactId>
                </dependency>
                <dependency>
                        <groupId>org.springframework.cloud</groupId>
                        <artifactId>spring-cloud-starter-config</artifactId>
                </dependency>
                <dependency>
                        <groupId>org.springframework.boot</groupId>
                        <artifactId>spring-boot-starter-web</artifactId>
                </dependency>
                <!-- 修改后立即生效，热部署 -->
                <dependency>
                        <groupId>org.springframework</groupId>
                        <artifactId>springloaded</artifactId>
                </dependency>
                <dependency>
                        <groupId>org.springframework.boot</groupId>
                        <artifactId>spring-boot-devtools</artifactId>
                </dependency>
        </dependencies>


</project>
```

```

```

yml

```
server:
  port: 80
```

configbean

```
package com.atguigu.springcloud.cfgbeans;

import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.web.client.RestTemplate;

/**
 * 配置类
 * @author 27660
 *
 */
@Configuration
public class ConfigBean {

        @Bean
        public RestTemplate getRestTemplate(){
                return new RestTemplate();
        }
}
```

注入RestTemplate 提供了多种便捷访问远程Http服务的方法

是一种简单便捷的访问restful服务模板类，是spring提供的用于访问rest服务客户端工具

DeptController_Consumer

```
package com.atguigu.springcloud.controller;

import java.util.List;

import org.bouncycastle.jcajce.provider.asymmetric.dsa.DSASigner.detDSA;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.PathVariable;
import org.springframework.web.bind.annotation.PostMapping;
import org.springframework.web.bind.annotation.RestController;
```

```
import org.springframework.web.client.RestTemplate;

import com.atguigu.springcloud.entities.Dept;

/**
 * 消费者没有service 只用消费
 * @author 27660
 *
 */
@RestController
public class DeptController_Consumer {

        private final String REST_URL_PREFIX= "http://localhost:8001" ;

        @Autowired
        private RestTemplate restTemplate ;

        @PostMapping("/consumer/dept/add")
        public boolean add(Dept dept){
                return restTemplate.postForObject(REST_URL_PREFIX+"/dept/add",
dept,Boolean.class);
        }

        @PostMapping("/consumer/dept/get/{id}")
        public Dept add(@PathVariable("id")Long  id){
                return restTemplate.getForObject(REST_URL_PREFIX+"/dept/get/"+id,
Dept.class);
        }


        @PostMapping("/consumer/dept/list")
        public List<Dept> list(Dept dept){
                return
restTemplate.getForObject(REST_URL_PREFIX+"/dept/list",List.class);
        }
}
```

入股启动出现这种问题

2018-09-14 10:41:08.271 WARN 21404 --- [ restartedMain] o.h.v.m.ParameterMessageInterpolator : HV000184: ParameterMessageInterpolator has been chosen, EL interpolation will not be supported 2018-09-14 10:41:08.290 INFO 21404 --- [ restartedMain] c.a.springcloud.DeptConsumer80_App : Started DeptConsumer80_App in 6.085 seconds (JVM running for 6.629) 2018-09-14 10:41:08.293 INFO 21404 --- [ Thread-9] s.c.a.AnnotationConfigApplicationContext : Closing org.springframework.context.annotation.AnnotationConfigApplicationContext@756d6b2b: startup date [Fri Sep 14 10:41:06 CST 2018]; parent: org.springframework.context.annotation.AnnotationConfigApplicationContext@dc50913 2018-09-14 10:41:08.297 INFO 21404 --- [ Thread-9] o.s.j.e.a.AnnotationMBeanExporter : Unregistering JMX-exposed beans on shutdown 2018-09-14 10:41:08.297 INFO 21404 --- [ Thread-9] o.s.j.e.a.AnnotationMBeanExporter : Unregistering JMX-exposed beans

缺少**server** 容器

引入一个jetty 就完事了

# 6.springcloud Eureka 服务注册发现

## 是什么

Netflix 在设计Eureka 时遵守的就是AP 原则

Eureka 是Netflix 的一个子模块，也是核心模块之一，Eureka是一个基于REST的服务，用于定位服务，实现在云端中间层服务发现和故障转移，服务注册与发现对于微服务架构来说非常重要，有了服务的发现注册，**只需要使用服务的标识符，就可以访问到服务**，不需要修改服务的调用配置文件**，功能类似于dubbo的注册中心，比如zookeeper**

springCloud 封装了netflix 公司开发的Eureka 模块来实现服务的注册和发现，（对比zookeeper）

Eureka 采用C-S 的设计架构，Eureka Server 作为服务注册功能的服务器，他是服务注册中心

系统的其他微服务，使用Eureka 的客户端连接到Eureka Server 并维持心跳连接，这样维护人员可以通过Eureka Server 来监控系统中的各个微服务是否正常运行，SpirngCloud的一些其他模块，比如（Zuul）就可以通过Eureka Server 来发现系统的其他微服务，并执行相关的逻辑

Eureka 包括两大组件 Eureka Server 和Eureka Client

Eureka Server 提供服务注册服务

各个节点启动后，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的节点的信息，服务节点信息可以在界面中直观的看到

EurekaClient 是一个java 客户端，用于简化Eureka Server 的交互，客户端同时也具备了一个内置的，使用轮询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30s）如果EurekaServer 在多个心跳周期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删除默认为90s

pom

```
<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>com.atguigu.springcloud</groupId>
                <artifactId>microservicecloud</artifactId>
                <version>0.0.1-SNAPSHOT</version>
        </parent>

        <artifactId>microservicecloud-eureka-7001</artifactId>

        <dependencies>
```

```
                <!--eureka-server服务端 -->
                <dependency>
                        <groupId>org.springframework.cloud</groupId>
                        <artifactId>spring-cloud-starter-eureka-
server</artifactId>
                        <exclusions>
                <exclusion>
                    <artifactId>spring-boot-starter-tomcat</artifactId>
                    <groupId>org.springframework.boot</groupId>
                </exclusion>
            </exclusions>
                </dependency>
                <dependency>
                        <groupId>org.springframework.boot</groupId>
                        <artifactId>spring-boot-starter-jetty</artifactId>
                </dependency>
                <!-- 修改后立即生效，热部署 -->
                <dependency>
                        <groupId>org.springframework</groupId>
                        <artifactId>springloaded</artifactId>
                </dependency>
                <dependency>
                        <groupId>org.springframework.boot</groupId>
                        <artifactId>spring-boot-devtools</artifactId>
                </dependency>
        </dependencies>

</project>
```

```
server:
  port: 7001

eureka:
  instance:
    hostname: eureka7001.com #eureka服务端的实例名称
  client:
    register-with-eureka: false     #false表示不向注册中心注册自己。
    fetch-registry: false     #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
    service-url:
      #单机 defaultZone: http://${eureka.instance.hostname}:${server.port}/eureka/
       #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（单机）。
      defaultZone:
http://eureka7002.com:7002/eureka/,http://eureka7003.com:7003/eureka/
```

# 将已有的部门微服务注册进eureka

1.pom

```
<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>com.atguigu.springcloud</groupId>
                <artifactId>microservicecloud</artifactId>
                <version>0.0.1-SNAPSHOT</version>
        </parent>

        <artifactId>microservicecloud-provider-dept-8001</artifactId>

        <dependencies>
                <!-- 引入自己定义的api通用包，可以使用Dept部门Entity -->
                <dependency>
                        <groupId>com.atguigu.springcloud</groupId>
                        <artifactId>microservicecloud-api</artifactId>
                        <version>${project.version}</version>
                </dependency>
                <!-- actuator监控信息完善 -->
                <dependency>
                        <groupId>org.springframework.boot</groupId>
                        <artifactId>spring-boot-starter-actuator</artifactId>
                </dependency>
                <!-- 将微服务provider侧注册进eureka -->
                <dependency>
                        <groupId>org.springframework.cloud</groupId>
                        <artifactId>spring-cloud-starter-eureka</artifactId>
                </dependency>
                <dependency>
                        <groupId>org.springframework.cloud</groupId>
                        <artifactId>spring-cloud-starter-config</artifactId>
                </dependency>
                <dependency>
                        <groupId>junit</groupId>
                        <artifactId>junit</artifactId>
                </dependency>
                <dependency>
                        <groupId>mysql</groupId>
                        <artifactId>mysql-connector-java</artifactId>
                </dependency>
                <dependency>
                        <groupId>com.alibaba</groupId>
                        <artifactId>druid</artifactId>
                </dependency>
                <dependency>
```

```
                        <groupId>ch.qos.logback</groupId>
                        <artifactId>logback-core</artifactId>
                </dependency>
                <dependency>
                        <groupId>org.mybatis.spring.boot</groupId>
                        <artifactId>mybatis-spring-boot-starter</artifactId>
                </dependency>
                <dependency>
                        <groupId>org.springframework.boot</groupId>
                        <artifactId>spring-boot-starter-jetty</artifactId>
                </dependency>
                <dependency>
                        <groupId>org.springframework.boot</groupId>
                        <artifactId>spring-boot-starter-web</artifactId>
                        <!-- 使用jetty排除tomcat -->
                        <exclusions>
                <exclusion>
                    <artifactId>spring-boot-starter-tomcat</artifactId>
                    <groupId>org.springframework.boot</groupId>
                </exclusion>
            </exclusions>
                </dependency>
                <dependency>
                        <groupId>org.springframework.boot</groupId>
                        <artifactId>spring-boot-starter-test</artifactId>
                </dependency>
                <!-- 修改后立即生效，热部署 -->
                <dependency>
                        <groupId>org.springframework</groupId>
                        <artifactId>springloaded</artifactId>
                </dependency>
                <dependency>
                        <groupId>org.springframework.boot</groupId>
                        <artifactId>spring-boot-devtools</artifactId>
                </dependency>
        </dependencies>

</project>
```

2.yml

```
server:
  port: 8001

mybatis:
  config-location: classpath:mybatis/mybatis.cfg.xml        # mybatis配置文件所在路
径
  type-aliases-package: com.atguigu.springcloud.entities    # 所有Entity别名类所在
包
  mapper-locations:
```

```
  - classpath:mybatis/mapper/**/*.xml                              # mapper映射文件

spring:
   application:
    name: microservicecloud-dept                                   #很重要
   datasource:
    type: com.alibaba.druid.pool.DruidDataSource                   # 当前数据源操作类型
    driver-class-name: org.gjt.mm.mysql.Driver                     # mysql驱动包
    url: jdbc:mysql://192.168.24.136:3306/springboot_test                  # 数据库名称
    username: root
    password: 123456
    dbcp2:
      min-idle: 5                                                  # 数据库连接池的最小维持
连接数
      initial-size: 5                                              # 初始化连接数
      max-total: 5                                                 # 最大连接数
      max-wait-millis: 200                                         # 等待连接获取的最大超时
时间

eureka:
  client: #客户端注册进eureka服务列表内
    service-url:
      defaultZone: http://localhost:7001/eureka
#       defaultZone:
http://eureka7001.com:7001/eureka/,http://eureka7002.com:7002/eureka/,http://eurek
a7003.com:7003/eureka/
  instance:
    instance-id: microservicecloud-dept8001
    prefer-ip-address: true     #访问路径可以显示IP地址

info:
  app.name: atguigu-microservicecloud
  company.name: www.atguigu.com
  build.artifactId: $project.artifactId$
  build.version: $project.version$
```

3.注解

```
package com.atguigu.springcloud;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.cloud.netflix.eureka.EnableEurekaClient;

/**
 * 启动类
 * @author 27660
 *
 */
@EnableEurekaClient
```

```
@SpringBootApplication
public class DeptProvider8001_App  {

        public static void main(String[] args) throws Exception {
                SpringApplication.run(DeptProvider8001_App.class, args);
        }

}
```

## Eureka 自我保护（红字)

某时刻某一个服务不可用了，eureka 不会理解清理，依旧会对服务的信息保存，

什么是自我保护?

默认情况下，如果eurekaServer 在一定时间内没有收到某个微服务实例的心跳，EurekaServer将会注销该实例，默认为90s，但是当网络分区故障发生时，微服务与eurekaServer之间无法通信，以上行为可能变得非常危险，因为微服务本身是健康的，此时不应该注册这个服务，eureka通过自我保护模式，来解决这个问题，当eureka通过自我保护模式来解决这个问题--当eurekaServer失去过多的客户端的时候（可能发生了网络分区故障）那么这个节点解决会进入自我保护模式，一旦进入该模式eurekaServer就会保护服务注册表中的信息，不在删除服务注册表中的数据，也就是不会注销任何微服务，当网络故障 恢复后，该eurekaserver 节点会自动退出自我保护状态。

在自我保护状态中，eurekaServer 会保护注册表中的信息，不在注销任何服务实例，当它收到的心跳数重新恢复到闸值以上时，该eureka Server 节点就会自动退出自我保护模式，他的设计哲学就是宁可保留错误的服务注册信息，也不盲目注销任何可能健康的服务实例，一句话讲解:好死不如赖活

综上，自我保护模式是一种应对网络异常的安全措施，他的架构哲学是宁可同时保留所有的微服务 （健康和不健康的都保留），也不盲目的注销任何健康的微服务，使用自我保护模式，它可以让eureka集群便的更加健壮稳定

在SpringCloud ，可以使用eureka.server.enable-self-preservation=false禁用自我保护模式

## 服务发现

反馈此服务的信息

在provider 服务提供者中注入

```
@Autowired
        private DiscoveryClient client;
```

书写服务

```
@RequestMapping(value = "/dept/discovery", method = RequestMethod.GET)
        public Object discovery()
        {
```

```
                List<String> list = client.getServices();
                System.out.println("**********" + list);

                List<ServiceInstance> srvList =
client.getInstances("MICROSERVICECLOUD-DEPT");
                for (ServiceInstance element : srvList) {
                        System.out.println(element.getServiceId() + "\t" +
element.getHost() + "\t" + element.getPort() + "\t"
                                        + element.getUri());
                }
                return this.client;
        }
```

在consumer 中调用微服务

```
        // 测试@EnableDiscoveryClient,消费端可以调用服务发现
        @RequestMapping(value = "/consumer/dept/discovery")
        public Object discovery()
        {
                return restTemplate.getForObject(REST_URL_PREFIX +
"/dept/discovery", Object.class);
        }
```

通过访问即可发现服务提供者信息

http://192.168.218.1/consumer/dept/discovery

# Eureka 集群搭建

什么叫做集群？

在不同的服务器上配置相同的服务，对外提供一个超大服务的整体，

新建工程7002 7003

1.pom 都要引入eureka client 客户端

```
 <dependencies>
                <!--eureka-server服务端 -->
                <dependency>
                        <groupId>org.springframework.cloud</groupId>
                        <artifactId>spring-cloud-starter-eureka-
server</artifactId>
                        <exclusions>
                <exclusion>
                    <artifactId>spring-boot-starter-tomcat</artifactId>
                    <groupId>org.springframework.boot</groupId>
                </exclusion>
            </exclusions>
                </dependency>
```

```
                <dependency>
                        <groupId>org.springframework.boot</groupId>
                        <artifactId>spring-boot-starter-jetty</artifactId>
                </dependency>
                <!-- 修改后立即生效，热部署 -->
                <dependency>
                        <groupId>org.springframework</groupId>
                        <artifactId>springloaded</artifactId>
                </dependency>
                <dependency>
                        <groupId>org.springframework.boot</groupId>
                        <artifactId>spring-boot-devtools</artifactId>
                </dependency>
        </dependencies>
```

2.yml中要集群的每个节点互相关联，互相知道彼此的存在

```
server:
  port: 7003

eureka:
  instance:
    hostname: eureka7003.com #eureka服务端的实例名称
  client:
    register-with-eureka: false     #false表示不向注册中心注册自己。
    fetch-registry: false     #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
    service-url:
#       defaultZone: http://${eureka.instance.hostname}:${server.port}/eureka/
      #单机 #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（单机）。
      defaultZone:
http://eureka7001.com:7001/eureka/,http://eureka7002.com:7002/eureka/
```

```
server:
  port: 7002

eureka:
  instance:
    hostname: eureka7002.com #eureka服务端的实例名称
  client:
    register-with-eureka: false     #false表示不向注册中心注册自己。
    fetch-registry: false     #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
    service-url:
#       defaultZone: http://${eureka.instance.hostname}:${server.port}/eureka/
      #单机 #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（单机）。
```

```
      defaultZone:
http://eureka7001.com:7001/eureka/,http://eureka7003.com:7003/eureka/
```

```
server:
  port: 7001

eureka:
  instance:
    hostname: eureka7001.com #eureka服务端的实例名称
  client:
    register-with-eureka: false     #false表示不向注册中心注册自己。
    fetch-registry: false     #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
    service-url:
#       defaultZone: http://${eureka.instance.hostname}:${server.port}/eureka/
      #单机 #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（单机）。
      defaultZone:
http://eureka7002.com:7002/eureka/,http://eureka7003.com:7003/eureka/
```

然后注册服务要分别向三个eureka 集群中的每一个节点进行 注册

```
eureka:
  client: #客户端注册进eureka服务列表内
    service-url:
#      defaultZone: http://localhost:7001/eureka
       defaultZone:
http://eureka7001.com:7001/eureka/,http://eureka7002.com:7002/eureka/,http://eurek
a7003.com:7003/eureka/
  instance:
    instance-id: microservicecloud-dept8001
    prefer-ip-address: true     #访问路径可以显示IP地址
```

## Eureka 比 zookeeper 好处在哪里？

CAP

RDBMS （mysql/orcacle/sqlServer）===》ACID

NOSQL （redis/mongdb）===》CAP

## 区别

1. zookeeper 遵守CP eureka 遵守AP

   **传统数据库**

   在分布式数据库中CAP原理CAP +BASE

   传统的ACID分别是什么？

   **A（Atomicity）原子性**

   **C （Consistency）一致性**

   **I（Isolation）独立性**

   ```
   **D（Durability）持久性**
   ```

   nosql 数据库

   **C 强一致性(Consistency)**

   ```
   **A可用性(Availability)**
   ```

   **P 分区容错性(分区容错性)**

   CA ----RDBMS

   CP ---MongoDB Hbase Redis

   AP--- CouchDB Cassandra

   任何一个分布式系统都不能完成三个特性，只能三进二

   cap的理论的核心就是:一个分布式系统 不能同时很好的满足一致性，可用性，分区容错性，这三个需求，根据CAP 原理将NoSQL 数据库分成了CA 原则，满足CP原则和满足Cp原则和满足Ap原则三大类

   CA --单点集群，满足一致性，可用性的系统，通常在可拓展性不太强大

   cp -- 满足一致性，分区容忍必得系统，通常性能不是特别高

   ap -- 满足可用性，分区容忍性的系统，通常可能对一致性要求低一点

## 作为注册中心Eureka比zookeeper 好在哪里

著名的CAP理论指出，一个分布式系统不可能同时满足C（一致性）A（可用性）P（分区容错）由于分区容错性P是在分布式系统中要保证的，因此我们只能在A C之间进行权衡

**zookeeper 保证的是CP**

**Eureka保证的是AP**

zookeeper 保证CP

当注册中心查询服务列表时，我们可易容忍注册中心返回的是几分钟以前的注册信息，但是不能接受服务直接down掉不可用，也就是说，服务注册功能对高可用的要求要高于一致性，但是zk会出现这种情况，当master节点因为网络故障与其他节点失去联系时，剩余节点会重新leader 选举，问题在于leader的时间过长，30~120s，整个选举期间zk集群都是不可用的，这样就导致在选举期间注册服务的瘫痪,在云部署的环境下，因为网络问题是的zk集群失去master节点是较大概率会发生的事情，虽然服务能够最终恢复，但是漫长的选举时间导致注册长期的不可用是不能容忍的

Eureka 保证AP

Eureka看明白了这一点，因此在设计时，优先保证高可用，Eureka各个节点都是平等的，几个节点挂掉不会影响正常节点的工作，剩下的节点依然可以提供注册和查询服务，而Eureka 的客户端在向某个Eureka 注册发生失败的时候，会自动的切换到 其他的节点，只要有一天Eureka还在，就能保证注册服务的可用，只不过查到的信息可能不是最新的（不保证强一致）除此之外，Eureka还有一种自我保护机制，如果在15 分钟内超过85%的节点都没有了正常的心跳，那么Eureka就认为客户端和服务注册中心出现了网络故障，此时会出现一下几种情况：

1.Eureka不在从注册列表中移除因为长时间没有收到心跳而应该过期的服务

2.Eureka仍然能够接受新服务的注册和查询请求，但是不会同步到其他的节点上（保证当前节点可用）

3.当网路稳定时，当前实例新的注册会被同步到其他的节点上

因此，Eureka 可以很好的应对因为网络故障导致部分节点失去联系的情况而不会像zookeeper那样使整个注册服务瘫痪

# Ribbon负载均衡

## 1.概述

### 1.1是什么

Spring Cloud Ribbon 是基于Netflix Ribbon 实现的一套 **客户端 负载均衡工具**

简单的说，Ribbon是Netflix 发布的开源项目，主要功能提供客户端的**软件负载均衡算法**将Netflix 的中间服务连接在一起，Ribbon客户端组件提供一系列完善的配置项如连接超时，重试，就是在配置文件中列出load Balancer 简称(LB) 后面所有的机器，Ribbon会自动的帮助你完成基于某种规则（如简单轮询，随机连接）去连接这些机器，我们很容易使用Ribbon实现自定义的负载均衡算法

### 1.2能干嘛

LB ,即负载均衡（Load Banlance）在微服务或者集群分布式中经常用的一种应用。

负载均衡简单的说就是将用户的请求平摊到多个服务上，从而达到系统的HA（负载均衡）

常见的负载均衡的软件有Nginx LVS 硬件F5 等

相应的中间件，例如dubbo SpringCloud中均给我们提供了负载均衡**SpringCloud的负载均衡算法可以自定义**

### 1.2.1集中式 LB （偏硬件）

在服务的消费放和提供方之间使用独立的LB设施，可以是硬件 F5也可以是软件Nginx 该设施应该负责把访问请求通过某种策略转发至服务的提供方（贵）

### 1.2.2进程内LB （偏软件）

将LB 逻辑集成到消费放，消费放从服务从注册中心获知有哪些地址可用，然后自动从这些地址中选出一个合适的服务器，Ribbon 就属于进程内LB 他只是一个类库，集成与消费方进程，消费方通过他来获取服务方提供的地址

## 2.Ribbon配置初步

客户端的负载均衡（切记）

### 2.1 microservicecloud-consumer-dept-80 pom

ribbon需要和eureka 整合eureka和config有联系

```
<!-- Ribbon相关 -->
                <dependency>
                        <groupId>org.springframework.cloud</groupId>
                        <artifactId>spring-cloud-starter-eureka</artifactId>
                </dependency>
                <dependency>
                        <groupId>org.springframework.cloud</groupId>
                        <artifactId>spring-cloud-starter-ribbon</artifactId>
                </dependency>
                <dependency>
                        <groupId>org.springframework.cloud</groupId>
                        <artifactId>spring-cloud-starter-config</artifactId>
                </dependency>
```

### 2.2 microservicecloud-consumer-dept-80 yml

配置eureka 让客户端访问注册中心

```
spring:
   application:
    name: microservicecloud-consumer-dept
eureka:
  client: #客户端注册进eureka服务列表内
    service-url:
#      defaultZone: http://localhost:7001/eureka
      defaultZone:
http://eureka7001.com:7001/eureka/,http://eureka7002.com:7002/eureka/,http://eurek
a7003.com:7003/eureka/
  instance:
    instance-id: microservicecloud-dept80
```

```
    prefer-ip-address: true      #访问路径可以显示IP地址
  #微服务信息
 info:
   app.name: atguigu-microservicecloud80
   company.name: www.atguigu.com
   build.artifactId: $project.artifactId$
   build.version: $project.version$
```

### 2.3 加@LoadBalanced 注解

```
        @Bean
        @LoadBalanced    //这一个注解代替了nigex配置的配置文件
        public RestTemplate getRestTemplate(){
                return new RestTemplate();
        }
```

### 2.4，让springboot支持eureka

```
@EnableEurekaClient
```

**@EnaleEurekaClient 和@EnableDiscoveryClient 区别，EnaleEurekaClient 主要使用用Eureka做注册中心的时候使用，但是@enableDisconveryClient 可以使用再其他不同的注册中心上，@EnableEurkaClient点开之后就是@EnableDiscoveryClient**

### 2.5容易出错的一部

```
//      private final String REST_URL_PREFIX= "http://localhost:8001" ;
        private final String REST_URL_PREFIX= "http://MICROSERVICECLOUD-DEPT"; //
按照名字访问微服务
```

使用rest调用的时候一定要加http

## 3.负载均衡

### 3.1架构说明

默认ribbon是轮询 的在rurkaserver 上调用服务

第一步先选择EurkaServer ， 他们优先选择在同一区域内负载均衡较少的server

第二步根据用户指定策略，再从server取到的微服务注册表中选择一个地址

ribbon提供了多种策略，轮询， 随机， 根据响应时间加权重

### 3.2 新建8001 8002

复制粘贴2001

### 3.3总结

Ribbon其实就是一个软负载均衡的客户端组件，

他可以额他的所需请求的客户端结合使用，和eureka结合只是其中的一个实例

## 4.ribbon核心组件IRule

### irule 根据特定的算法从服务列表中选取一个要访问的服务

默认提供了7中算法

主要都在这个com.netflix.loadbalancer包下面，他们都实现了一个接口Irule 也就是说如果我们自己实现Irule我们就可以自己实现我们自己定制的规则

**RoundRobinRule** 轮询规则

**RandomRule** 随机规则

**AvailabilityFilteringRule** 会先过滤掉由于多次访问故障而引起的断路器跳闸状态的服务，还有并发的链接数量闸值服务，然后对于剩余的服务列表按照轮询策略进行访问

**WeightedResponseTimeRule** 根据平均响应时间计算所有的服务权重，响应时间越快权重越大被选中的几率就越高，刚启动时信息量不足，则使用RoundRobinRule策略，等统计信息足够，会切换到 **WeightedResponseTimeRule**

**RetryRule** 先按照RoundRobinRule 的策略获取服务，如果获取服务失败则载指定的时间内重试，获取可用的服务

**BestAvailableRule** 先会过滤由于多次访问故障而处于路由器跳闸状态的服务，然后选择一个并发量小的服务

**ZonAvoidaceRule** 默认规则， 符合判断Server 所在的区域的性能和server 的可用性选择服务器

### 4.1切换负载均衡规则

使用springboot我们只需要把我们自己配置的规则放入ioc容器即可

```
        @Bean
        public IRule myRule(){
                return new RandomRule();
        }
```

### 4.2Ribbon自定义规则

以后学习。。。

# 5.Feign 负载均衡

## 5.1是什么

feign 是一个声明式WebService客户端，使用Feign能让编写WebService 客户端更加简单，他的使用方法是定义一个接口，然后在上面添加注解，同时支持JAX-RS标准的注解，Feign也支持可插拔使的编码器，和解码器，SpringCloud对Feign进行了封装，使其支持了SpringMVC标准注解和HttpMessageConverters Feign可以和Eureka和Ribbon组合使用以支持负载均衡

Feign是一个声明式的Web服务客户端，使得编写Web服务客户端变得非常容易，

**只需要创建一个接口，然后在上面添加注解即可**

社区比较支持面向接口的调用，所以springcloud提供了Feign ，可以实现接口注解，面向接口掉调用，

## Feign能干什么

feign旨在使编写javahttp客户端变得更加容易，

前面使用ribbon+restTemplate时，利用RestTemplate对http请求的封装处理，形成一套模板化的调用方法，实际开发中，由于对服务的调用可能不只是移除，往往一个接口可能会被多出调用，所以通常我们定义和实现依赖服务接口的定义，在Feign的实现下，我们只需要创建一个接口并使用注解的方式来配置他，即可完成服务提供方的接口绑定，简化了springcloud ribbon时候自动封装的大量开发

**Feign集成了ribbon**

和ribbon不同的使，通过feign只需要通过自定义服务绑定接口且以声明式的方法，优雅而简单的 实现服务的调用

## 5.2 使用feign 使用面向接口的方式调用服务

### 5.2.1 创建接口描述，

1.引入需要的starter

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

2.创建接口描述

```
@FeignClient(value="MICROSERVICECLOUD-DEPT")
public interface DeptClientService {

        @GetMapping("/hello")
        public String hello();
```

```
}
```

加上@FeignClient 注解， 然后写调用的服务名称

### 5.2.1 创建microservicecloud-consumer-dept-feign client

1.引入starter

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

2.配置yml

```
server:
  port: 80

spring:
   application:
    name: microservicecloud-consumer-dept
eureka:
  client: #客户端注册进eureka服务列表内
    service-url:
#      defaultZone: http://localhost:7001/eureka
      defaultZone:
http://eureka7001.com:7001/eureka/,http://eureka7002.com:7002/eureka/,http://eurek
a7003.com:7003/eureka/
  instance:
    instance-id: microservicecloud-dept-feign
    prefer-ip-address: true     #访问路径可以显示IP地址
 #微服务信息
info:
  app.name: atguigu-microservicecloud80
  company.name: www.atguigu.com
  build.artifactId: $project.artifactId$
  build.version: $project.version$
```

4编写controller 调用接口

```
 *
 */
@RestController
public class DeptController_Consumer {
```

```
        @Autowired
        private DeptClientService deptClientService ;

        @GetMapping("/hello")
        public String hello(){
                return deptClientService.hello();
        }
```

5.编写启动类，标注@EnableFeignClients 注解

```
@EnableFeignClients
@EnableEurekaClient
@SpringBootApplication
public class DeptConsumer_feign_App {

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

}
```

然后即可

feign的负载均衡规则默认和ribbon一样都是轮询，我们可以像该Ribbon一样进行操作

```
        @Bean
        public IRule myRule(){
                return new RandomRule();
        }
```

# 6.Hystrix 断路器（熔断器）

他的思想和spring的前置通知环绕通知有相似之处，他就是程序出异常，调用超时的时候怎么办

## 6.1概述

### 6.1.1分布式面临的问题

复杂的分布式体系中的应用程序有数十个依赖关系，每个依赖关系在某些时候将不可避免的失败，down机

服务雪崩

多个服务之间调用的时候，假设服务A调用服务B和服务C，服务B和服务C 有调用了其他 的微服务，这就是所谓的“扇出”如果扇出的链路上某个微服务的调用响应时间过长或者不可用，对微服务A的调用就会占用越来越多的系统资源，而引起系统崩溃，所谓的雪崩效应

对于流量高的应用来说，单一的后端依赖可能会导致所有服务器上的资源几秒泡汤，比失败更糟糕的是，这些应用进程还可能导致服务之间的延迟增加，备份队列，线程和其他系统资源紧张导致，整个系统更多级联故障，这些都表示需要对故障的延迟进行隔离和管理，以便单个依赖关系的失败，不能取消整体应用程序或者系统

### 6.1.2 是什么

Hystrix 是一个用于处理分布式系统的延迟和容错的开源库，在分布式系统中，许多依赖不可避免会调用失败，比如超时，异常等，Hystrix 能够保证在一个依赖出现问题的情况西，不会导致整体服务失败，避免级联故障，提高分布式系统的弹性

“断路器”本身是一种开关装置，当某个服务单元发生故障后，通过断路器的故障监控（类似于保险丝）**向调用方返回一个符合预期的，可处理的备选响应（FallBack）而不是长时间的等待或者抛出调用方法无法处理异常，**这样就保证了服务调用方线程不会被长时间，不必要的占用，从而避免了故障再分布式系统的蔓延，乃至雪崩

## 6.2服务熔断

熔断机制是应对雪崩效应的一种微服务链路保护机制

当扇出链路的某个服务不可用或者响应时间太长时候，会进行服务的降级，进而熔断该节点微服务的调用，，快速返回“错误”的相应信息，当检测到该节点微服务调用正常后恢复调用链路，在SpringCloud框架里熔断机制通过Hystrix实现，Hystrix会监控服务之间调用状态，当失败调用到达一定的闸值，缺省是5秒内20次调用失败就会启动熔弹机制，熔断机制的注解是@HystrixCommand

**1，引入对应的starter**

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

**2.配置对应的yml**

```
server:
  port: 8001


spring:
   application:
    name: microservicecloud-dept                                #很重要

eureka:
  client: #客户端注册进eureka服务列表内
    service-url:
```

```
#      defaultZone: http://localhost:7001/eureka
       defaultZone:
http://eureka7001.com:7001/eureka/,http://eureka7002.com:7002/eureka/,http://eurek
a7003.com:7003/eureka/
  instance:
    instance-id: microservicecloud-dept8001-hystix
    prefer-ip-address: true     #访问路径可以显示IP地址

 #微服务信息
info:
  app.name: atguigu-microservicecloud
  company.name: www.atguigu.com
  build.artifactId: $project.artifactId$
  build.version: $project.version$
```

**3.设置熔断方法用@HystrixCommand **

```
//一旦调用服务方法失败并抛出了错误信息后，会自动调用@HystrixCommand标注好的
fallbackMethod调用类中的指定方法
        @HystrixCommand(fallbackMethod="processHystrix_Get")
        @GetMapping("/hello")
        public String hello() {
                throw new RuntimeException();
        }

        public String processHystrix_Get()
        {
                return "hello请求出现异常,调用了processHystrix_Get指定的方法";
        }
```

**5.启动springboot对Hystrix的支持@EnableCircuitBreaker**

```
@SpringBootApplication
@EnableEurekaClient //本服务启动后会自动注册进eureka服务中
@EnableDiscoveryClient //服务发现 这个就是返回此服务的信息通过
@EnableCircuitBreaker//对hystrixR熔断机制的支持
public class DeptProvider_hystrix8001__App  {

        public static void main(String[] args) throws Exception {
                SpringApplication.run(DeptProvider_hystrix8001__App.class, args);
        }

}
```

# 6.3服务降级

**服务降级处理是在客户端与服务端没有关系**

整体资源快不够，忍痛将某些服务先关闭，待度过难关在开启

## 6.3.1服务降级实现步骤

**1.编写DeptClientServiceFallbackFactory 实现FallbackFactory传递泛型DetpClientService**

```
@Component // 不要忘记添加，不要忘记添加
public class DeptClientServiceFallbackFactory implements
FallbackFactory<DeptClientService>
{
        @Override
        public DeptClientService create(Throwable throwable)
        {
                return new DeptClientService() {

                        @Override
                        public String hello() {
                                return "客户端开启熔断措施";
                        }


                };
        }
}
```

**2.在DetpClientService上添加fallbackFacktory**

```
@FeignClient(value = "MICROSERVICECLOUD-
DEPT",fallbackFactory=DeptClientServiceFallbackFactory.class)
```

**3.在配置文件中配置**

```
feign:
  hystrix:
    enabled: true
```

# 6.4服务熔断小总结

spring有两个重要的技术支柱ioc aop

# 6.5服务监控HystrixDashboard

除了隔离依赖服务调用之外，Hystrix还提供了准时的调用监控（Hystrix Dashboard）Hystrix会雌花序的记录 所有通过Hystrix发起的请求执行信息，并以统计图和图形的形式展示给用户，包括每秒请求多少剁成，多少失败等，Netflix通过hystrix-metrics-event-steam 项目实现了对以上指标的监控,Spring Cloud提供了Hystrix Dashboard的整合，对监控内容转化为可视化界面

### 6.5.1 构建springboot项目 ，启动服务监控HystrixDashboard 豪猪

**1.引入依赖starter**

```
<!-- hystrix和 hystrix-dashboard相关 -->
                <dependency>
                        <groupId>org.springframework.cloud</groupId>
                        <artifactId>spring-cloud-starter-hystrix</artifactId>
                </dependency>
                <dependency>
                        <groupId>org.springframework.cloud</groupId>
                        <artifactId>spring-cloud-starter-hystrix-
dashboard</artifactId>
                </dependency>
```

**2.配置properties**

```
server:
  port: 9001
```

**3.使用@EnablHystrixDashboard 开启springboot 对HystrixDashboard的支持**

```
@SpringBootApplication
@EnableHystrixDashboard
public class DeptConsumer_DashBoard_App
{
        public static void main(String[] args)
        {
                SpringApplication.run(DeptConsumer_DashBoard_App.class, args);
        }
}
```

**4.run**

http://localhost:9001/hystrix

**5.实现监控**

http://localhost:8001/hystrix.stream

在要监控的微服务后面加上hhystrix.stream

可以使用可视化界面

# 7. Zuul路由网关（GateWay）

## 7.1概述

7.1.1 是什么

Zuul包含了对请求将外部请求转发到服务实例上，是实现外部访问统一入口的基础而过滤器的功能负责对请求处理过程进行干涉，是实现请求校验，服务聚合等功能的基础，Zuul和Eureka进行整合，将Zuul自身注册为Eureka服务分治下的应用，同时Eureka中获得其他服务的信息，也既以后的访问微服务都是从Zuul跳转后获得，

注意：Zuul服务最终还是会注册到 Eureka 中

**提供= 代理+路由+过滤三大功能 **

## 7.2概述

构建项目

创建microservicecloud-zuul-gateway-9527

**1.pom文件 **

```
<!-- zuul路由网关 -->
                <dependency>
                        <groupId>org.springframework.cloud</groupId>
                        <artifactId>spring-cloud-starter-zuul</artifactId>
                </dependency>
                <dependency>
                        <groupId>org.springframework.cloud</groupId>
                        <artifactId>spring-cloud-starter-eureka</artifactId>
                </dependency>
```

**2.application.yml**

```
server:
  port: 9527

spring:
  application:
    name: microservicecloud-zuul-gateway

eureka:
  client:
    service-url:
```

```
      defaultZone:
http://eureka7001.com:7001/eureka,http://eureka7002.com:7002/eureka,http://eureka7
003.com:7003/eureka
  instance:
    instance-id: gateway-9527.com
    prefer-ip-address: true



    info:
  app.name: atguigu-microcloud
  company.name: www.atguigu.com
  build.artifactId: $project.artifactId$
  build.version: $project.version$
```

**3.**创建启动类，添加**@eablexxx** 注解

```
package com.atguigu.springcloud;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.cloud.netflix.zuul.EnableZuulProxy;

@SpringBootApplication
@EnableZuulProxy
public class Zuul_9527_StartSpringCloudApp
{
        public static void main(String[] args)
        {
                SpringApplication.run(Zuul_9527_StartSpringCloudApp.class, args);
        }
}
```

**4.**测试

这个时候通过zuul 和微服务ip 都是可以访问的我们可以通过一下配置让只能从zuul进行调用微服务

```
zuul:
  #ignored-services: microservicecloud-dept
  prefix: /atguigu
  #ignored-services 隐藏微服务，不能通过微服务名称调用
  ignored-services: "*"
  routes:
    mydept.serviceId: microservicecloud-dept
    mydept.path: /mydept/**
```

# 8.SpringCloud Config 分布式配置中心

## 8.1概述

eureka 微服务的注册中心

ribbon 实现负载均衡

feign 是一种基于面向接口调用的负载均衡解决方案，他集成了ribbon

hystrix 服务熔断降级 （豪猪）

zuul 路由网关，通过路由网关访问所有的微服务，是微服务的入口

config 集中微服务 的配置，配置服务器为每个不同的微服务应用所用的环境提供了中心化的配置