  
SpringCloud第一季 讲师:尚硅谷周阳

1.   
   微服务概述与SpringCloud
   1. 微服务与微服务架构

业界大牛马丁.福勒（Martin Fowler） 这样描述微服务：

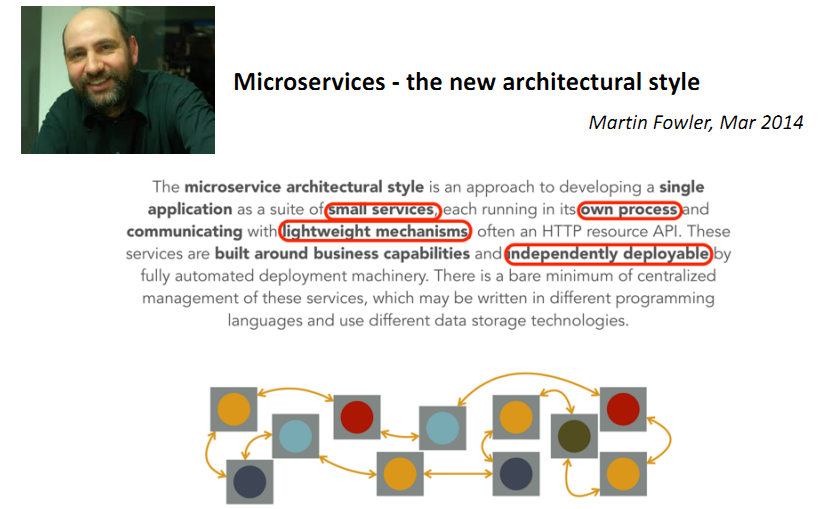
论文网址： https://martinfowler.com/articles/microservices.html

* 微服务

强调的是服务的大小，它关注的是某一个点，是具体解决某一个问题/提供落地对应服务的一个服务应用,

狭意的看,可以看作Eclipse里面的一个个微服务工程/或者Module

* 微服务架构



微服务架构是⼀种架构模式，它提倡将单⼀应⽤程序划分成⼀组⼩的服务，服务之间互相协调、互相配合，为⽤户提供最终价值。每个服务运⾏在其独⽴的进程中，服务与服务间采⽤轻量级的通信机制互相协作（通常是基于HTTP协议的RESTful API）。每个服务都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，应当尽量避免统⼀的、集中式的服务管理机制，对具体的⼀个服务⽽⾔，应根据业务上下⽂，选择合适的语⾔、⼯具对其进⾏构建。

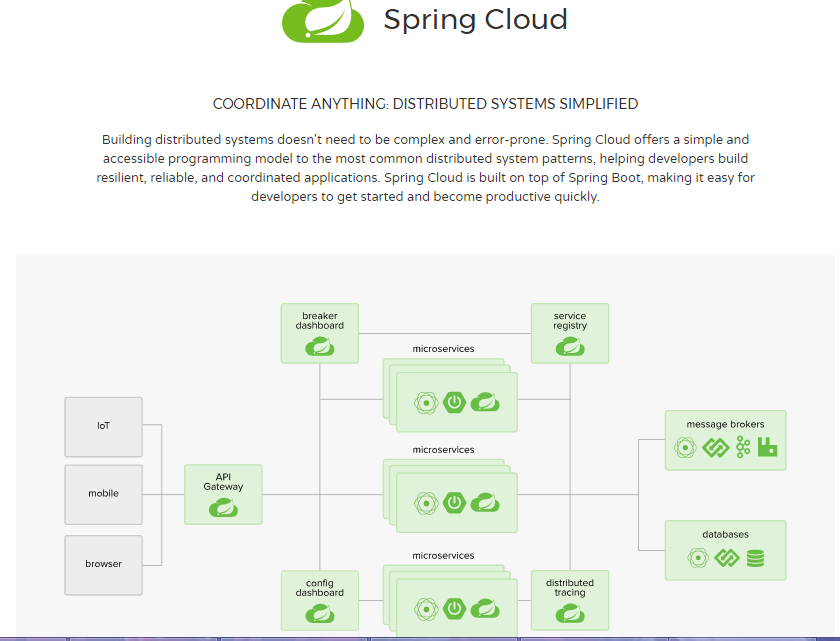
* + 1. 技术维度理解

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 地去耦合,每一个微服务提供单个业务功能的服务，一个服务做一件事， 从技术角度看就是一种小而独立的处理过程，类似进程概念，能够自行单独启动 或销毁，拥有自己独立的数据库。

* 1. 微服务技术栈有哪些

|  |  |  |
| --- | --- | --- |
| 微服务条目 | 落地技术 | 备注 |
| 服务开发 | Springboot、Spring、SpringMVC |  |
| 服务配置与管理 | Netflix公司的Archaius、阿里的Diamond等 |  |
| 服务注册与发现 | Eureka、Consul、Zookeeper等 |  |
| 服务调用 | Rest、RPC、gRPC |  |
| 服务熔断器 | Hystrix、Envoy等 |  |
| 负载均衡 | Ribbon、Nginx等 |  |
| 服务接口调用(客户端调用服务的简化工具) | Feign等 |  |
| 消息队列 | Kafka、RabbitMQ、ActiveMQ等 |  |
| 服务配置中心管理 | SpringCloudConfig、Chef等 |  |
| 服务路由（API网关） | Zuul等 |  |
| 服务监控 | Zabbix、Nagios、Metrics、Spectator等 |  |
| 全链路追踪 | Zipkin，Brave、Dapper等 |  |
| 服务部署 | Docker、OpenStack、Kubernetes等 |  |
| 数据流操作开发包 | SpringCloud Stream（封装与Redis,Rabbit、Kafka等发送接收消息） |  |
| 事件消息总线 | Spring Cloud Bus |  |
| ...... |  |  |

* 1. SpringCloud是什么
     1. 官网说明



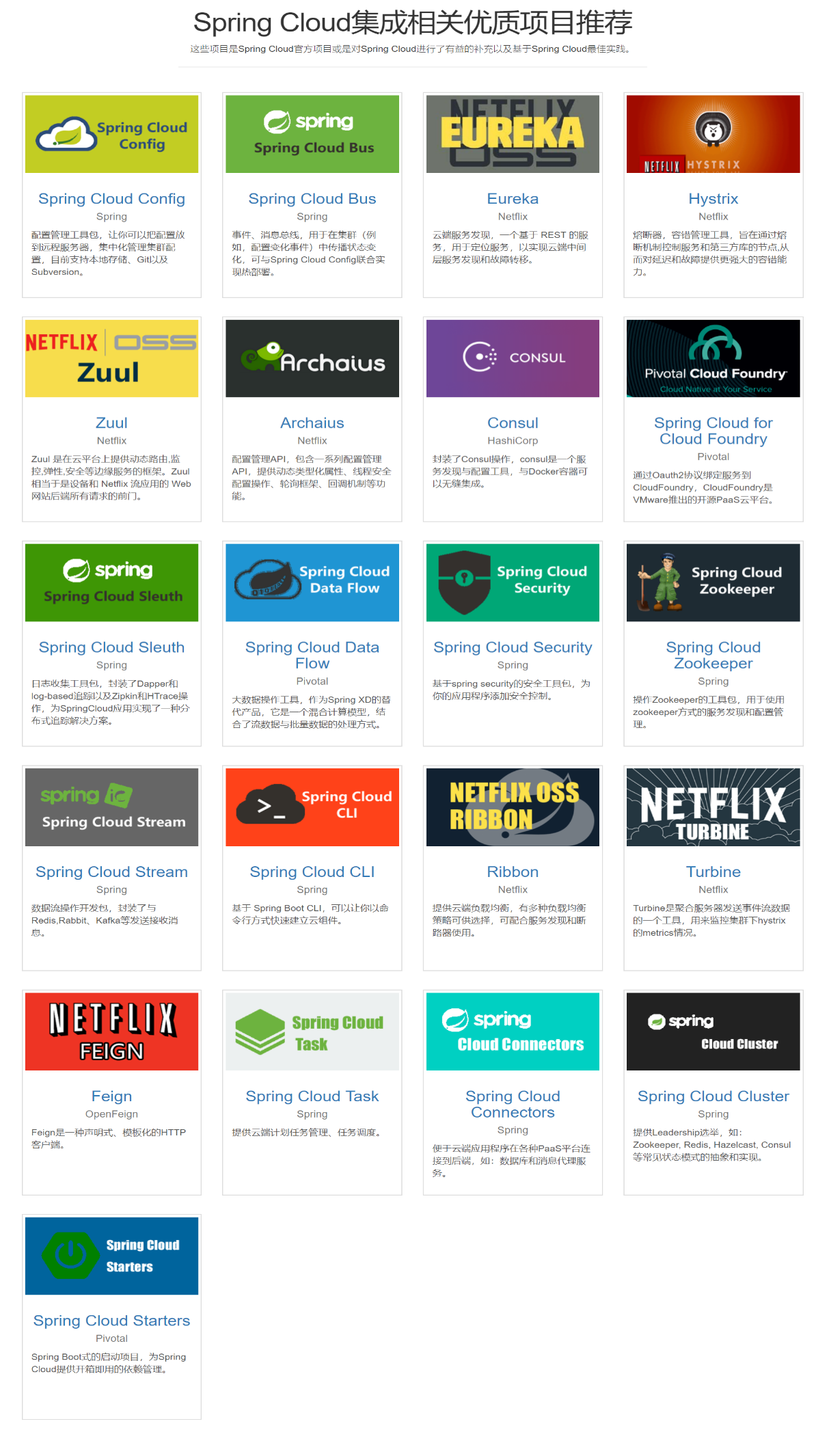
SpringCloud，基于SpringBoot提供了一套微服务解决方案，包括服务注册与发现，配置中心，全链路监控，服务网关，负载均衡，熔断器等组件，除了基于NetFlix的开源组件做高度抽象封装之外，还有一些选型中立的开源组件。

SpringCloud利用SpringBoot的开发便利性巧妙地简化了分布式系统基础设施的开发，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、分布式会话等等,它们都可以用SpringBoot的开发风格做到一键启动和部署。

SpringBoot并没有重复制造轮子，它只是将目前各家公司开发的比较成熟、经得起实际考验的服务框架组合起来，通过SpringBoot风格进行再封装屏蔽掉了复杂的配置和实现原理，最终给开发者留出了一套简单易懂、易部署和易维护的分布式系统开发工具包

* + 1. SpringCloud=分布式微服务架构下的一站式解决方案， 是各个微服务架构落地技术的集合体，俗称微服务全家桶

All



* + 1. SpringCloud和SpringBoot是什么关系

SpringBoot专注于快速方便的开发单个个体微服务。

SpringCloud是关注全局的微服务协调整理治理框架，它将SpringBoot开发的一个个单体微服务整合并管理起来，

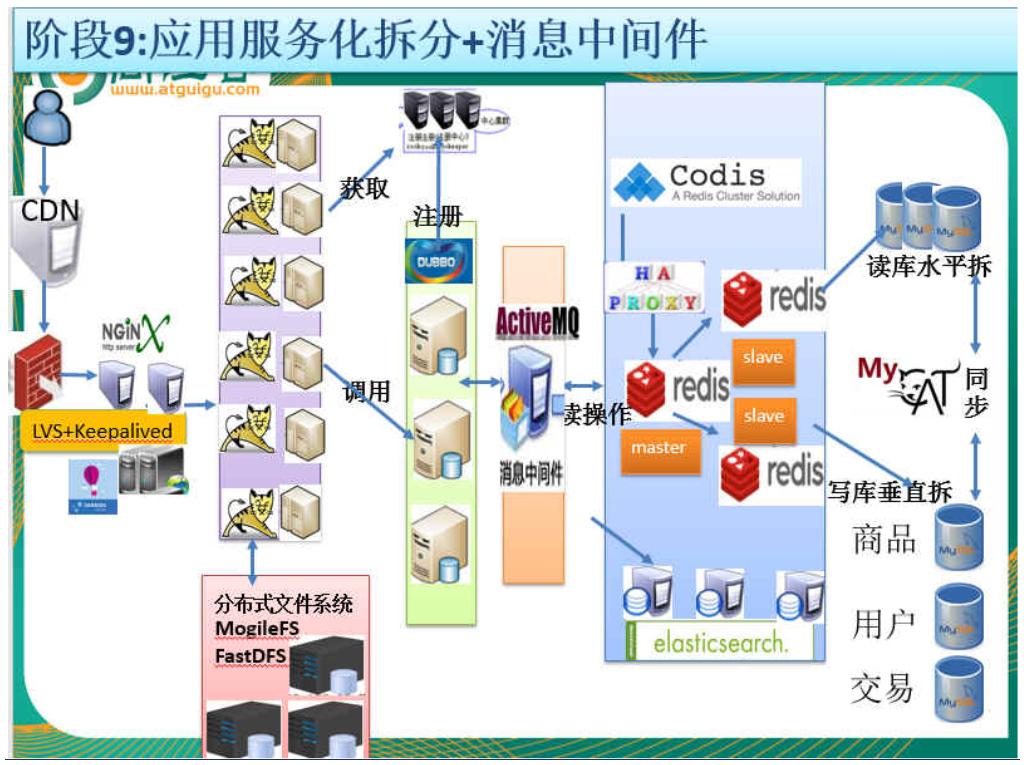
为各个微服务之间提供，配置管理、服务发现、断路器、路由、微代理、事件总线、全局锁、决策竞选、分布式会话等等集成服务

SpringBoot可以离开SpringCloud独立使用开发项目，但是SpringCloud离不开SpringBoot，属于依赖的关系.

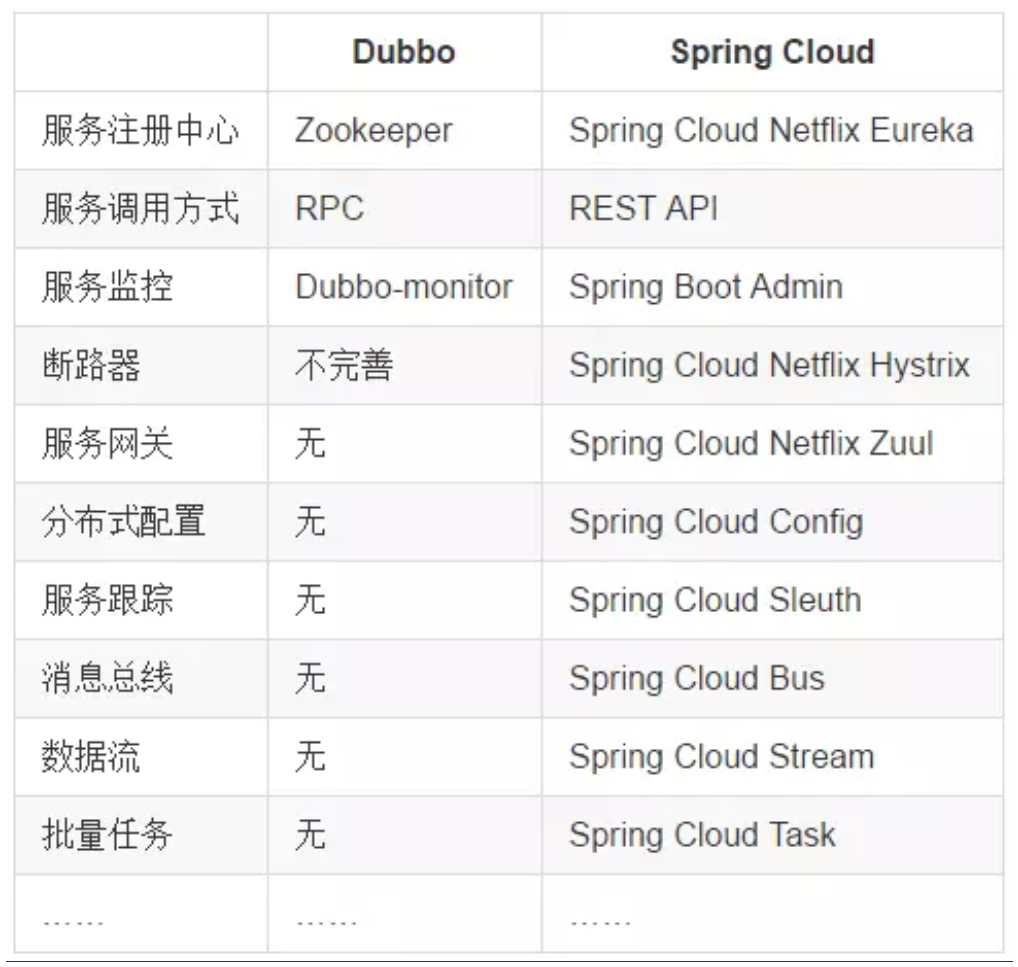
SpringBoot专注于快速、方便的开发单个微服务个体，SpringCloud关注全局的服务治理框架。

* + 1. Dubbo是怎么到SpringCloud的？ 哪些优缺点让你去技术选型

目前成熟的互联网架构（分布式+服务治理Dubbo）



我们把SpringCloud VS DUBBO进行一番对比



最大区别：SpringCloud抛弃了Dubbo的RPC通信，采用的是基于HTTP的REST方式。

严格来说，这两种方式各有优劣。虽然从一定程度上来说，后者牺牲了服务调用的性能，但也避免了上面提到的原生RPC带来的问题。而且REST相比RPC更为灵活，服务提供方和调用方的依赖只依靠一纸契约，不存在代码级别的强依赖，这在强调快速演化的微服务环境下，显得更加合适。

品牌机与组装机的区别

很明显，Spring Cloud的功能比DUBBO更加强大，涵盖面更广，而且作为Spring的拳头项目，它也能够与Spring Framework、Spring Boot、Spring Data、Spring Batch等其他Spring项目完美融合，这些对于微服务而言是至关重要的。使用Dubbo构建的微服务架构就像组装电脑，各环节我们的选择自由度很高，但是最终结果很有可能因为一条内存质量不行就点不亮了，总是让人不怎么放心，但是如果你是一名高手，那这些都不是问题；而Spring Cloud就像品牌机，在Spring Source的整合下，做了大量的兼容性测试，保证了机器拥有更高的稳定性，但是如果要在使用非原装组件外的东西，就需要对其基础有足够的了解。

社区支持与更新力度

最为重要的是，DUBBO停止了5年左右的更新，虽然2017.7重启了。对于技术发展的新需求，需要由开发者自行拓展升级（比如当当网弄出了DubboX），这对于很多想要采用微服务架构的中小软件组织，显然是不太合适的，中小公司没有这么强大的技术能力去修改Dubbo源码+周边的一整套解决方案，并不是每一个公司都有阿里的大牛+真实的线上生产环境测试过。

总结Cloud与Dubbo

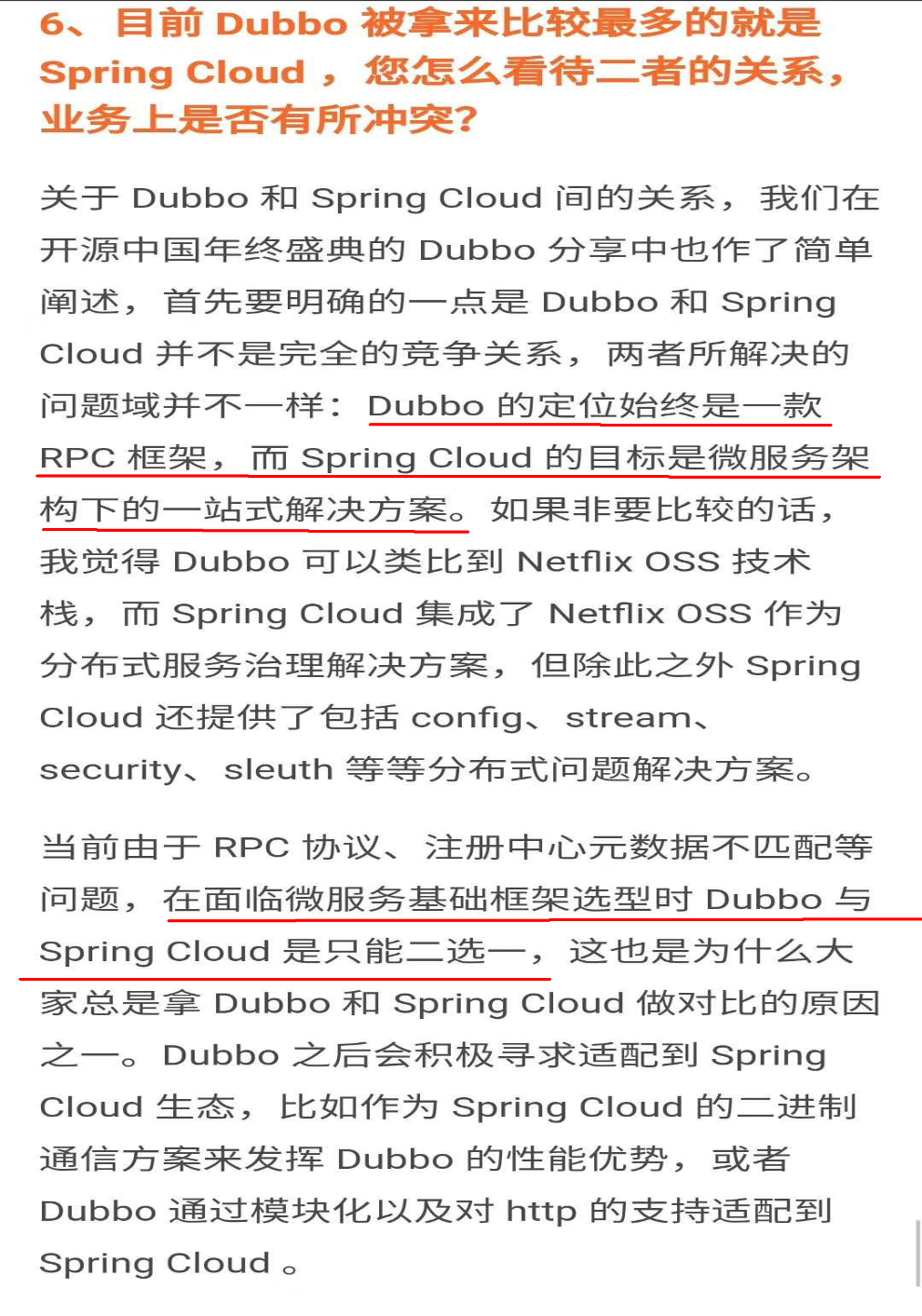
问题：

曾风靡国内的开源 RPC 服务框架 Dubbo 在重启维护后，令许多用户为之雀跃，但同时，也迎来了一些质疑的声音。互联网技术发展迅速，Dubbo 是否还能跟上时代？Dubbo 与 Spring Cloud 相比又有何优势和差异？是否会有相关举措保证 Dubbo 的后续更新频率？

人物：Dubbo重启维护开发的刘军，主要负责人之一

刘军，阿里巴巴中间件高级研发工程师，主导了 Dubbo 重启维护以后的几个发版计划，专注于高性能 RPC 框架和微服务相关领域。曾负责网易考拉 RPC 框架的研发及指导在内部使用，参与了服务治理平台、分布式跟踪系统、分布式一致性框架等从无到有的设计与开发过程。





* 1. 参考资料
     1. 官网

http://projects.spring.io/spring-cloud/

* + 1. 参考书

https://springcloud.cc/spring-cloud-netflix.html

本次开发API说明

http://cloud.spring.io/spring-cloud-static/Dalston.SR1/

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

springcloud中国社区

http://springcloud.cn/

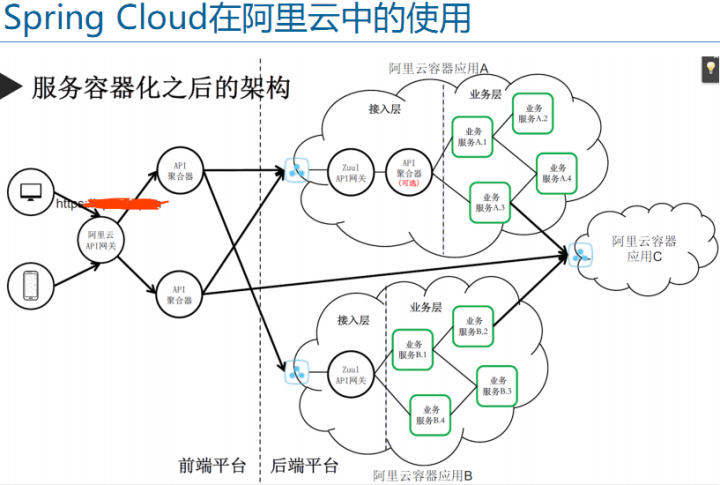
springcloud中文网

https://springcloud.cc/

* 1. SpringCloud国内使用情况
     1. 国内公司



* + 1. 阿里云



1.   
   Rest微服务构建 案例工程模块
   1. 总体介绍
      1. 承接着我们的springmvc+mybatis+mysql初级高级课程，以Dept部门模块做一个微服务通用案例 Consumer消费者（Client）通过REST调用Provider提供者（Server）提供的服务
      2. Maven的分包分模块架构复习

一个简单的Maven模块结构是这样的：

---- app-parent 一个父项目(app-parent)聚合很多子项目(app-util,app-dao,app-service,app-web)

|---- pom.xml (pom)

|

|-------- app-util

| |-------- pom.xml (jar)

|

|-------- app-dao

| |-------- pom.xml (jar)

|

|-------- app-service

| |-------- pom.xml (jar)

|

|-------- app-web

|-------- pom.xml (war)

一个Project带着多个Module子模块

MicroServiceCloud父工程（Project）下初次带着3个子模块（Module）

microservicecloud-api

封装的整体Entity/接口/公共配置等

microservicecloud-provider-dept-8001

微服务落地的服务提供者

microservicecloud-consumer-dept-80

微服务调用的客户端使用

80端口

80端口是为HTTP(HyperText Transport Protocol)即超文本传输协议开放的

此为上网冲浪使用次数最多的协议，主要用于WWW(World Wide Web)即万维网传输信息的协议。

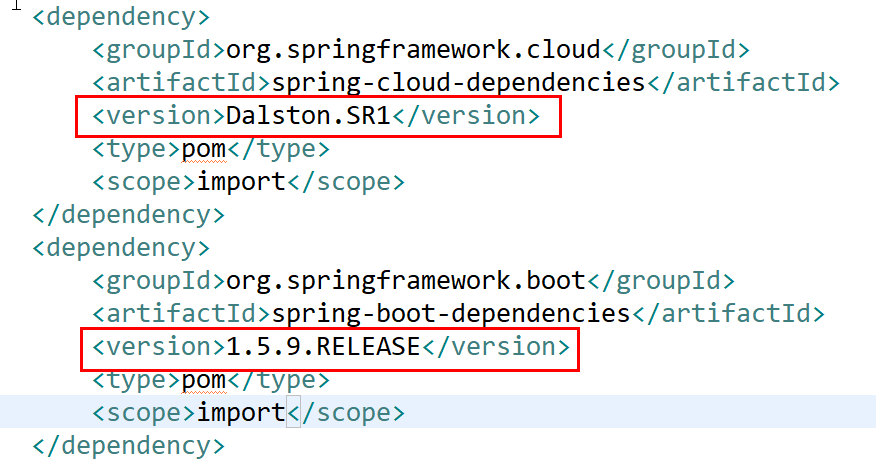
可以通过HTTP地址(即常说的"网址")加":80"来访问网站，

因为浏览网页服务默认的端口号都是80，因此只需输入网址即可，不用输入":80"了。

* + 1. 直接动手干



* 1. 本次springCloud版本



* 1. 构建步骤
     1. microservicecloud 整体父工程Project

新建父工程microservicecloud，切记是Packageing是pom模式



主要是定义POM文件，将后续各个子模块公用的jar包等统一提出来，类似一个抽象父类

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>

<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>

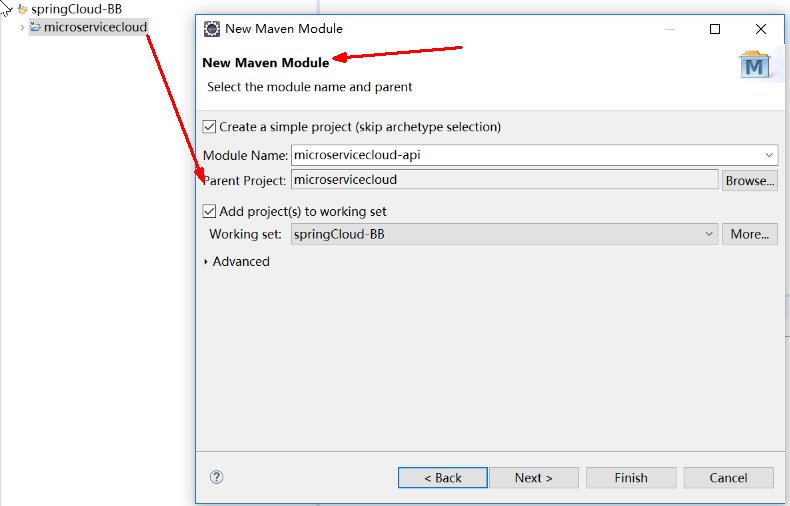
</dependencies>

</dependencyManagement>

</project>

* + 1. microservicecloud-api 公共子模块Module

新建microservicecloud-api



创建完成后请回到父工程查看pom文件变化

修改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><!-- 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>

</dependencies>

</project>

新建部门Entity且配合lombok使用

**package** com.atguigu.springcloud.entities;

**import** java.io.Serializable;

**import** lombok.Data;

**import** lombok.NoArgsConstructor;

**import** lombok.experimental.Accessors;

@SuppressWarnings("serial")

@NoArgsConstructor

@Data

@Accessors(chain=**true**)

**public** **class** Dept **implements** Serializable //必须序列化

{

**private** Long deptno; //主键

**private** String dname; //部门名称

**private** String db\_source;//来自那个数据库，因为微服务架构可以一个服务对应一个数据库，同一个信息被存储到不同数据库

**public** Dept(String dname)

{

**super**();

**this**.dname = dname;

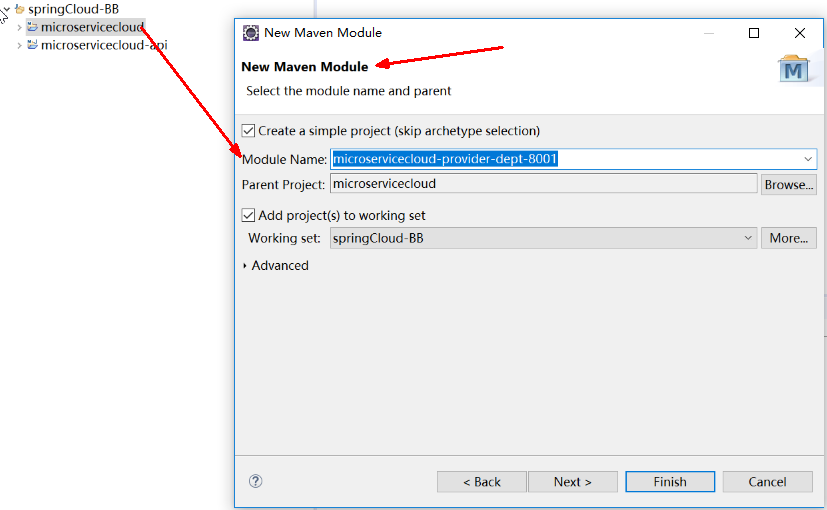
}

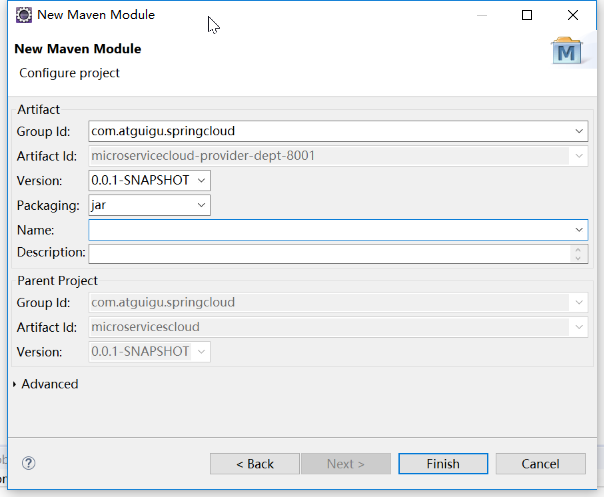
}

mvn clean install后给其它模块引用，达到通用目的。 也即需要用到部门实体的话，不用每个工程都定义一份，直接引用本模块即可。

* + 1. microservicecloud-provider-dept-8001 部门微服务提供者Module

新建microservicecloud-provider-dept-8001





创建完成后请回到父工程查看pom文件变化

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>

<dependency><!-- 引入自己定义的api通用包，可以使用Dept部门Entity -->

<groupId>com.atguigu.springcloud</groupId>

<artifactId>microservicecloud-api</artifactId>

<version>${project.version}</version>

</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

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

工程src/main/resources目录下新建mybatis文件夹后新建mybatis.cfg.xml文件

<?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>

MySQL创建部门数据库脚本

**DROP** DATABASE IF **EXISTS** cloudDB01;

**CREATE** DATABASE cloudDB01 **CHARACTER** **SET** UTF8;

USE cloudDB01;

**CREATE** **TABLE** dept

(

deptno BIGINT **NOT** **NULL** **PRIMARY** **KEY** AUTO\_INCREMENT,

dname **VARCHAR**(60),

db\_source **VARCHAR**(60)

);

**INSERT** **INTO** dept(dname,db\_source) **VALUES**('开发部',DATABASE());

**INSERT** **INTO** dept(dname,db\_source) **VALUES**('人事部',DATABASE());

**INSERT** **INTO** dept(dname,db\_source) **VALUES**('财务部',DATABASE());

**INSERT** **INTO** dept(dname,db\_source) **VALUES**('市场部',DATABASE());

**INSERT** **INTO** dept(dname,db\_source) **VALUES**('运维部',DATABASE());

**SELECT** \* **FROM** dept;

DeptDao部门接口

**package** com.atguigu.springcloud.dao;

**import** java.util.List;

**import** org.apache.ibatis.annotations.Mapper;

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

@Mapper

**public** **interface** DeptDao

{

**public** **boolean** addDept(Dept dept);

**public** Dept findById(Long id);

**public** List<Dept> findAll();

}

工程src/main/resources/mybatis目录下新建mapper文件夹后再建DeptMapper.xml

<?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>

DeptService部门服务接口

**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();

}

DeptServiceImpl部门服务接口实现类

**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();

}

}

DeptController部门微服务提供者REST

**package** com.atguigu.springcloud.controller;

**import** java.util.List;

**import** org.springframework.beans.factory.annotation.Autowired;

**import** org.springframework.web.bind.annotation.PathVariable;

**import** org.springframework.web.bind.annotation.RequestBody;

**import** org.springframework.web.bind.annotation.RequestMapping;

**import** org.springframework.web.bind.annotation.RequestMethod;

**import** org.springframework.web.bind.annotation.RestController;

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

**import** com.atguigu.springcloud.service.DeptService;

@RestController

**public** **class** DeptController

{

@Autowired

**private** DeptService service;

@RequestMapping(value="/dept/add",method=RequestMethod.***POST***)

**public** **boolean** add(@RequestBody Dept dept)

{

**return** service.add(dept);

}

@RequestMapping(value="/dept/get/{id}",method=RequestMethod.***GET***)

**public** Dept get(@PathVariable("id") Long id)

{

**return** service.get(id);

}

@RequestMapping(value="/dept/list",method=RequestMethod.***GET***)

**public** List<Dept> list()

{

**return** service.list();

}

}

DeptProvider8001\_App主启动类DeptProvider8001\_App

**package** com.atguigu.springcloud;

**import** org.springframework.boot.SpringApplication;

**import** org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication

**public** **class** DeptProvider8001\_App

{

**public** **static** **void** main(String[] args)

{

SpringApplication.*run*(DeptProvider8001\_App.**class**, args);

}

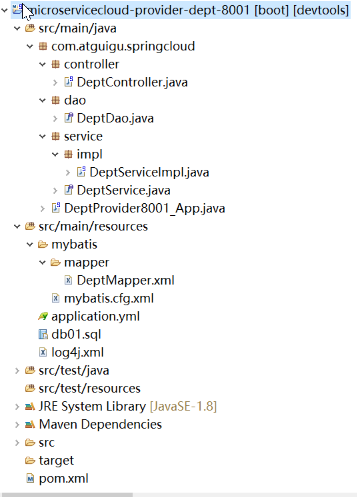
}

测试

http://localhost:8001/dept/get/2

http://localhost:8001/dept/list

最终工程展现



* + 1. microservicecloud-consumer-dept-80 部门微服务消费者Module

新建microservicecloud-consumer-dept-80

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>

<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

com.atguigu.springcloud.cfgbeans包下ConfigBean的编写（类似spring里面的applicationContext.xml写入的注入Bean）

**package** com.atguigu.springcloud.cfgbeans;

**import** org.springframework.context.annotation.Bean;

**import** org.springframework.context.annotation.Configuration;

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

@Configuration

**public** **class** ConfigBean

{

@Bean

**public** RestTemplate getRestTemplate()

{

**return** **new** RestTemplate();

}

}

com.atguigu.springcloud.controller包下新建DeptController\_Consumer部门微服务消费者REST

RestTemplate

是什么

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服务的客户端模板工具集

官网及使用

官网地址

https://docs.spring.io/spring-framework/docs/4.3.7.RELEASE/javadoc-api/org/springframework/web/client/RestTemplate.html

使用

使用restTemplate访问restful接口非常的简单粗暴无脑。

(url, requestMap, ResponseBean.class)这三个参数分别代表

REST请求地址、请求参数、HTTP响应转换被转换成的对象类型。

代码

**package** com.atguigu.springcloud.controller;

**import** java.util.List;

**import** org.springframework.beans.factory.annotation.Autowired;

**import** org.springframework.web.bind.annotation.PathVariable;

**import** org.springframework.web.bind.annotation.RequestMapping;

**import** org.springframework.web.bind.annotation.RestController;

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

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

@RestController

**public** **class** DeptController\_Consumer

{

**private** **static** **final** String ***REST\_URL\_PREFIX*** = "http://localhost:8001";

@Autowired

**private** RestTemplate restTemplate;

@RequestMapping(value="/consumer/dept/add")

**public** **boolean** add(Dept dept)

{

**return** restTemplate.postForObject(***REST\_URL\_PREFIX***+"/dept/add", dept, Boolean.**class**);

}

@RequestMapping(value="/consumer/dept/get/{id}")

**public** Dept get(@PathVariable("id") Long id)

{

**return** restTemplate.getForObject(***REST\_URL\_PREFIX***+"/dept/get/"+id, Dept.**class**);

}

@SuppressWarnings("unchecked")

@RequestMapping(value="/consumer/dept/list")

**public** List<Dept> list()

{

**return** restTemplate.getForObject(***REST\_URL\_PREFIX***+"/dept/list", List.**class**);

}

}

DeptConsumer80\_App主启动类

**package** com.atguigu.springcloud;

**import** org.springframework.boot.SpringApplication;

**import** org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication

**public** **class** DeptConsumer80\_App

{

**public** **static** **void** main(String[] args)

{

SpringApplication.*run*(DeptConsumer80\_App.**class**, args);

}

}

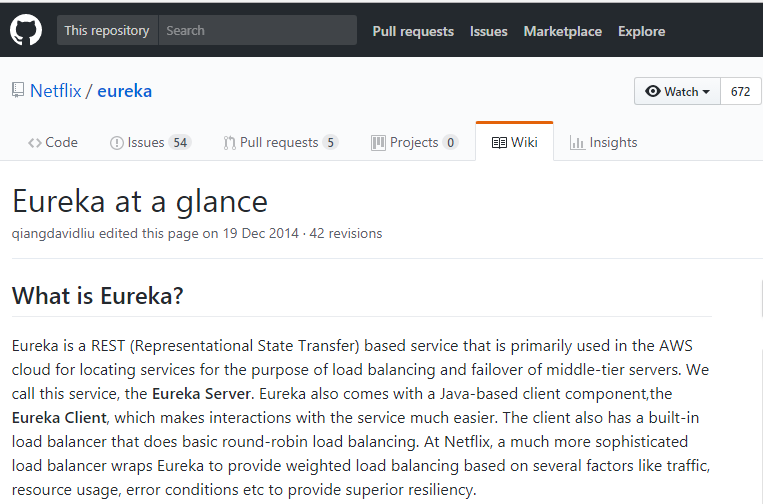
测试

http://localhost/consumer/dept/get/2

http://localhost/consumer/dept/list

http://localhost/consumer/dept/add?dname=AI

1.   
   Eureka服务注册与发现
   1. 是什么



Eureka是什么

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，以实现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是非常重要的，有了服务发现与注册，只需要使用服务的标识符，就可以访问到服务，而不需要修改服务调用的配置文件了。功能类似于dubbo的注册中心，比如Zookeeper。

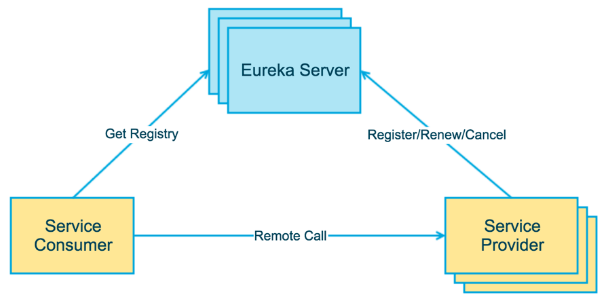
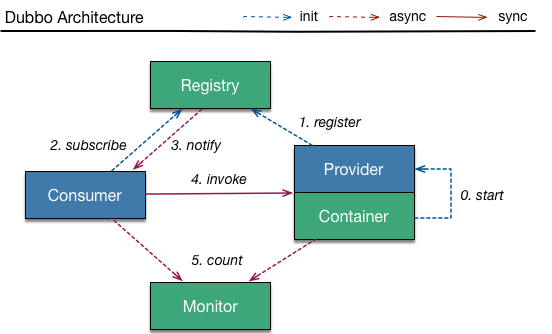
* + 1. 
  1. 原理讲解
     1. Eureka的基本架构

Spring Cloud 封装了 Netflix 公司开发的 Eureka 模块来实现服务注册和发现(请对比Zookeeper)。

Eureka 采用了 C-S 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。SpringCloud 的一些其他模块（比如Zuul）就可以通过 Eureka Server 来发现系统中的其他微服务，并执行相关的逻辑。

请注意和Dubbo的架构对比

Eureka包含两个组件：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

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）

* + 1. 三大角色

Eureka Server 提供服务注册和发现

Service Provider服务提供方将自身服务注册到Eureka，从而使服务消费方能够找到

Service Consumer服务消费方从Eureka获取注册服务列表，从而能够消费服务

* 1. 构建步骤
     1. microservicecloud-eureka-7001 eureka服务注册中心Module

新建microservicecloud-eureka-7001

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>

</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: 7001

eureka:

instance:

hostname: localhost #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/ #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。

EurekaServer7001\_App主启动类

**package** com.atguigu.springcloud;

**import** org.springframework.boot.SpringApplication;

**import** org.springframework.boot.autoconfigure.SpringBootApplication;

**import** org.springframework.cloud.netflix.eureka.server.EnableEurekaServer;

@SpringBootApplication

@EnableEurekaServer//EurekaServer服务器端启动类,接受其它微服务注册进来

**public** **class** EurekaServer7001\_App

{

**public** **static** **void** main(String[] args)

{

SpringApplication.*run*(EurekaServer7001\_App.**class**, args);

}

}

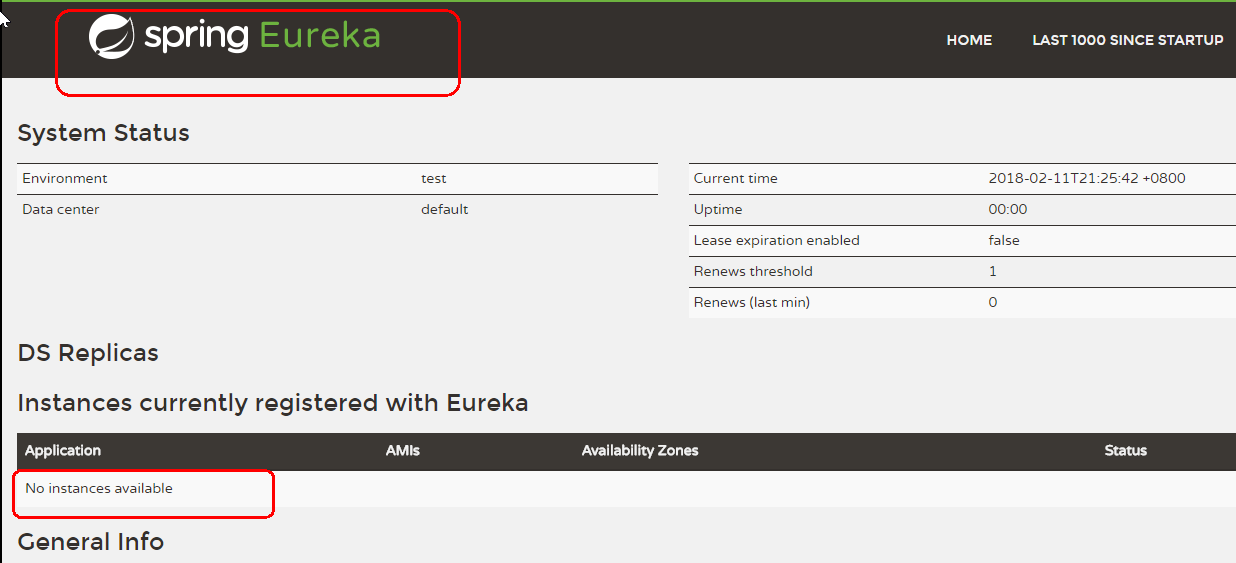
@EnableEurekaServer

另见: [@EnableEurekaClient](#a_EnableEurekaClient)

测试

http://localhost:7001/

结果页面



No application available 没有服务被发现 O(∩\_∩)O 因为没有注册服务进来当然不可能有服务被发现

* + 1. microservicecloud-provider-dept-8001 将已有的部门微服务注册进eureka服务中心

修改microservicecloud-provider-dept-8001

POM

修改部分

<!-- 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>

完整内容

<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>

<!-- 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

修改部分

eureka:

client: #客户端注册进eureka服务列表内

service-url:

defaultZone: http://localhost:7001/eureka

完整内容

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

DeptProvider8001\_App主启动类

**package** com.atguigu.springcloud;

**import** org.springframework.boot.SpringApplication;

**import** org.springframework.boot.autoconfigure.SpringBootApplication;

**import** org.springframework.cloud.netflix.eureka.EnableEurekaClient;

@SpringBootApplication

@EnableEurekaClient //本服务启动后会自动注册进eureka服务中

**public** **class** DeptProvider8001\_App

{

**public** **static** **void** main(String[] args)

{

SpringApplication.*run*(DeptProvider8001\_App.**class**, args);

}

}

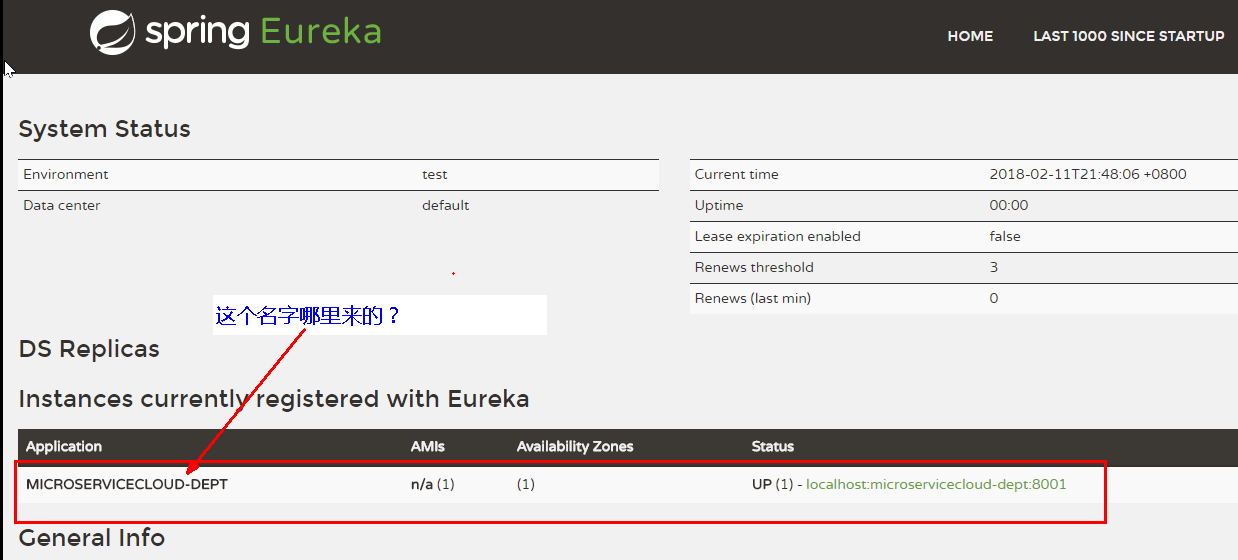
@EnableEurekaClient

另见: [@EnableEurekaServer](#a_EnableEurekaServer)

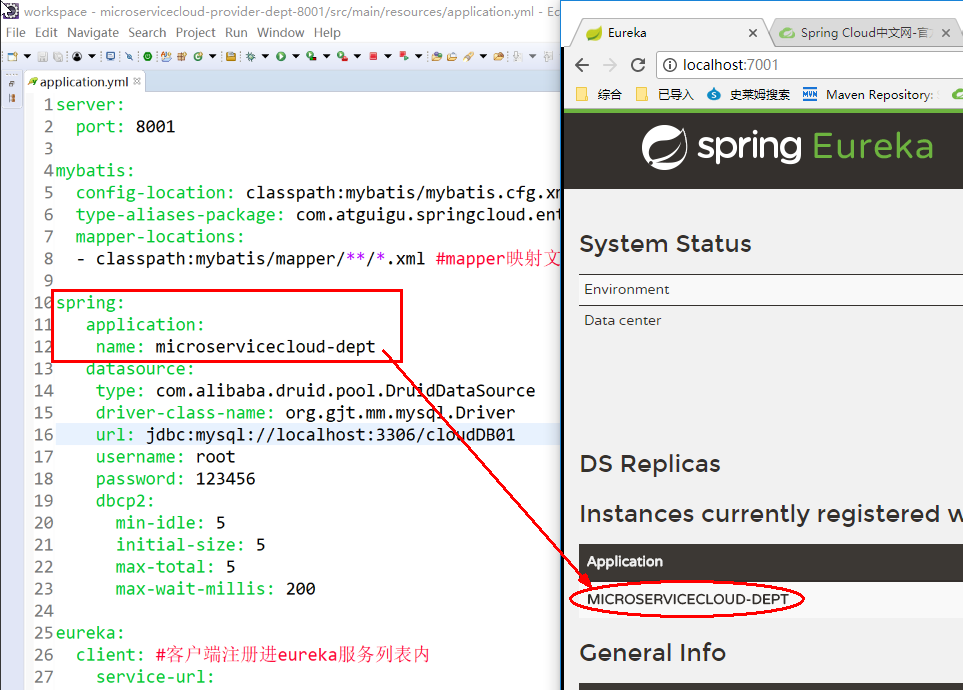
测试

先要启动EurekaServer

http://localhost:7001/



微服务注册名配置说明



* + 1. actuator与注册微服务信息完善

主机名称:服务名称修改

当前问题



含有主机名称

修改microservicecloud-provider-dept-8001

YML

修改部分



eureka:

client: #客户端注册进eureka服务列表内

service-url:

defaultZone: http://localhost:7001/eureka

instance:

instance-id: microservicecloud-dept8001

完整内容

server:

port: 8001

mybatis:

config-location: classpath:mybatis/mybatis.cfg.xml #mybatis所在路径

type-aliases-package: com.atguigu.springcloud.entities #entity别名类

mapper-locations:

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

spring:

application:

name: microservicecloud-dept

datasource:

type: com.alibaba.druid.pool.DruidDataSource

driver-class-name: org.gjt.mm.mysql.Driver

url: jdbc:mysql://localhost:3306/cloudDB01

username: root

password: 123456

dbcp2:

min-idle: 5

initial-size: 5

max-total: 5

max-wait-millis: 200

eureka:

client: #客户端注册进eureka服务列表内

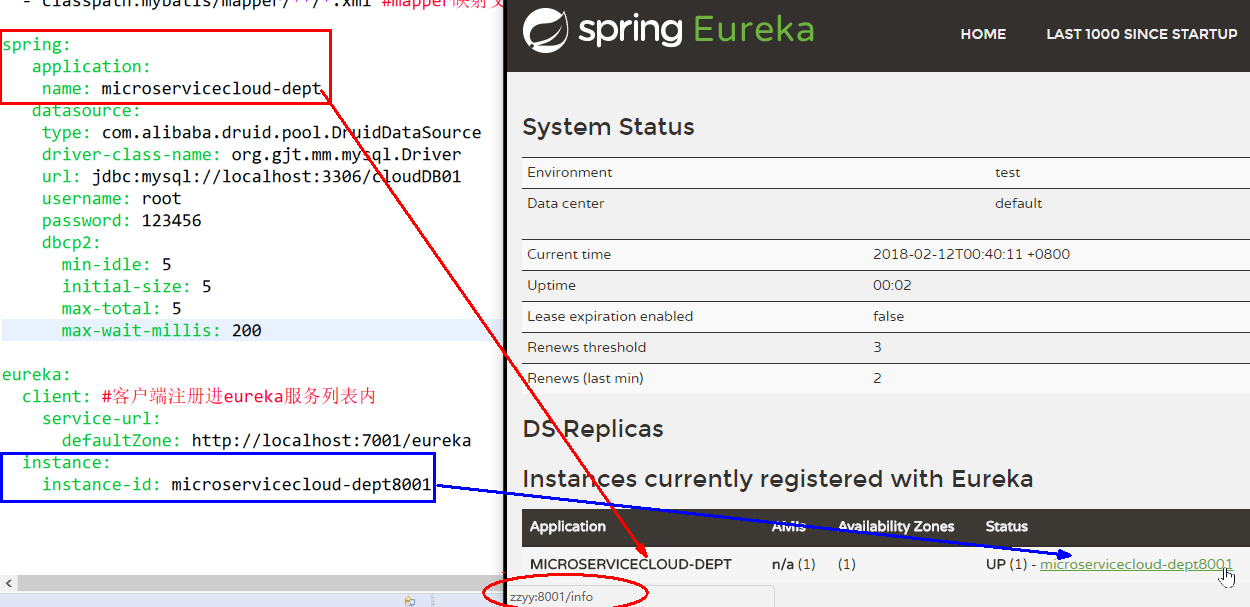
service-url:

defaultZone: http://localhost:7001/eureka

instance:

instance-id: microservicecloud-dept8001

修改之后



访问信息有IP信息提示

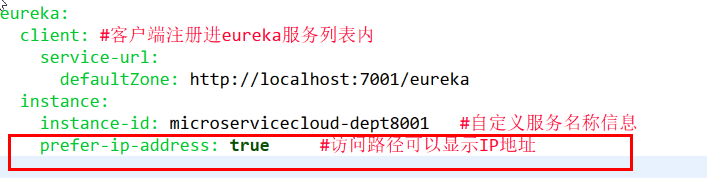
当前问题

没有IP提示

修改microservicecloud-provider-dept-8001

YML

修改部分



eureka:

client: #客户端注册进eureka服务列表内

service-url:

defaultZone: http://localhost:7001/eureka

instance:

instance-id: microservicecloud-dept8001 #自定义服务名称信息

prefer-ip-address: **true** #访问路径可以显示IP地址

完整内容

server:

port: 8001

mybatis:

config-location: classpath:mybatis/mybatis.cfg.xml #mybatis所在路径

type-aliases-package: com.atguigu.springcloud.entities #entity别名类

mapper-locations:

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

spring:

application:

name: microservicecloud-dept

datasource:

type: com.alibaba.druid.pool.DruidDataSource

driver-class-name: org.gjt.mm.mysql.Driver

url: jdbc:mysql://localhost:3306/cloudDB01

username: root

password: 123456

dbcp2:

min-idle: 5

initial-size: 5

max-total: 5

max-wait-millis: 200

eureka:

client: #客户端注册进eureka服务列表内

service-url:

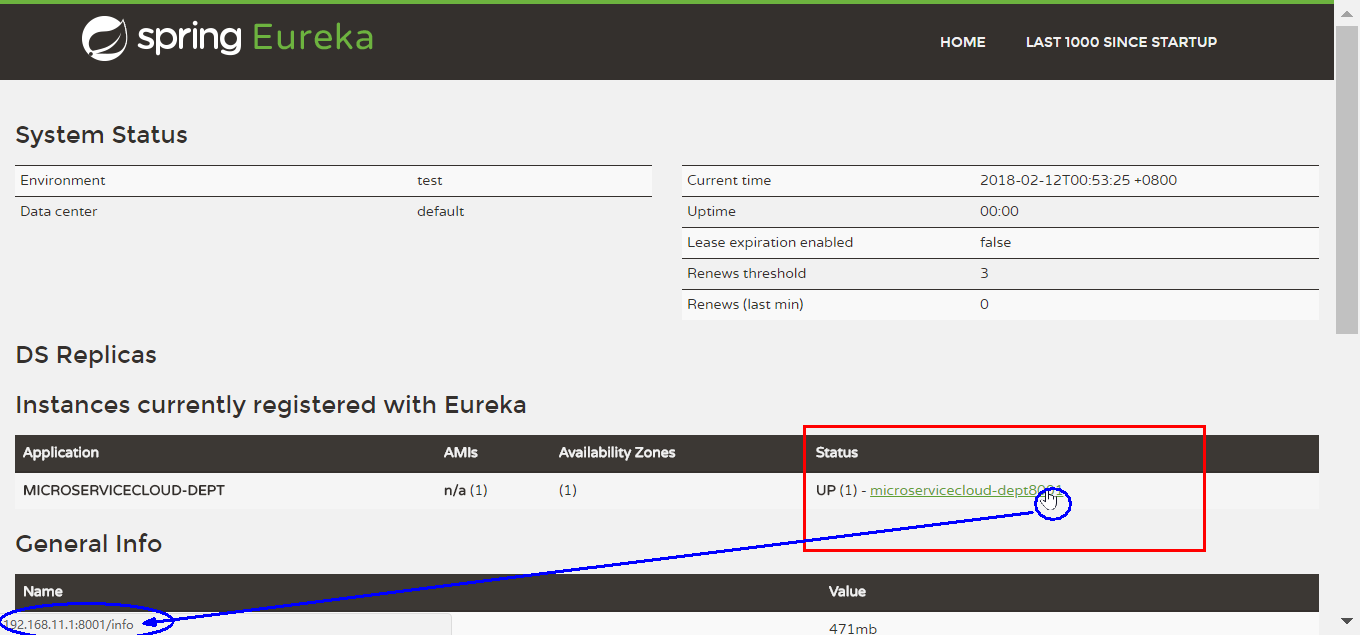
defaultZone: http://localhost:7001/eureka

instance:

instance-id: microservicecloud-dept8001 #自定义服务名称信息

prefer-ip-address: **true** #访问路径可以显示IP地址

修改之后



微服务info内容详细信息

当前问题

超链接点击服务报告ErrorPage

修改microservicecloud-provider-dept-8001

POM

修改部分

<dependency>

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

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

</dependency>

完整内容

<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>

<!-- 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>

<!-- actuator监控信息完善 -->

<dependency>

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

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

总的父工程microservicecloud修改pom.xml添加构建build信息

POM

修改部分

<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>

完整内容

<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>

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

<module>microservicecloud-consumer-dept-80</module>

<module>microservicecloud-eureka-7001</module>

</modules>

</project>

修改microservicecloud-provider-dept-8001

YML

修改部分

info:

app.name: atguigu-microservicecloud

company.name: www.atguigu.com

build.artifactId: $project.artifactId$

build.version: $project.version$

完整内容

server:

port: 8001

mybatis:

config-location: classpath:mybatis/mybatis.cfg.xml #mybatis所在路径

type-aliases-package: com.atguigu.springcloud.entities #entity别名类

mapper-locations:

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

spring:

application:

name: microservicecloud-dept

datasource:

type: com.alibaba.druid.pool.DruidDataSource

driver-class-name: org.gjt.mm.mysql.Driver

url: jdbc:mysql://localhost:3306/cloudDB01

username: root

password: 123456

dbcp2:

min-idle: 5

initial-size: 5

max-total: 5

max-wait-millis: 200

eureka:

client: #客户端注册进eureka服务列表内

service-url:

defaultZone: http://localhost:7001/eureka

instance:

instance-id: microservicecloud-dept8001 #自定义服务名称信息

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: $project.version$

* + 1. eureka自我保护

演示Case

故障现象





导致原因

什么是自我保护模式？

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，EurekaServer将会注销该实例（默认90秒）。但是当网络分区故障发生时，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。一旦进入该模式，EurekaServer就会保护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）。当网络故障恢复后，该Eureka Server节点会自动退出自我保护模式。

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。当它收到的心跳数重新恢复到阈值以上时，该Eureka Server节点就会自动退出自我保护模式。它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：好死不如赖活着

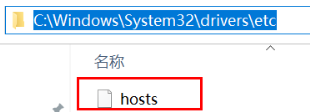
综上，自我保护模式是一种应对网络异常的安全保护措施。它的架构哲学是宁可同时保留所有微服务（健康的微服务和不健康的微服务都会保留），也不盲目注销任何健康的微服务。使用自我保护模式，可以让Eureka集群更加的健壮、稳定。

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

一句话：某时刻某一个微服务不可用了，eureka不会立刻清理，依旧会对该微服务的信息进行保存

* 1. 集群配置
     1. 原理说明
     2. 新建microservicecloud-eureka-7002/microservicecloud-eureka-7003
     3. 按照7001为模板粘贴POM
     4. 修改7002和7003的主启动类
     5. 修改映射配置

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



修改映射配置添加进hosts文件

127.0.0.1 eureka7001.com

127.0.0.1 eureka7002.com

127.0.0.1 eureka7003.com

* + 1. 3台eureka服务器的yml配置

7001

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/

7002

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/ #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。

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

7003

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/ #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。

defaultZone: http://eureka7001.com:7001/eureka/,http://eureka7002.com:7002/eureka/

* + 1. microservicecloud-provider-dept-8001 微服务发布到上面3台eureka集群配置中

YML

server:

port: 8001

mybatis:

config-location: classpath:mybatis/mybatis.cfg.xml #mybatis所在路径

type-aliases-package: com.atguigu.springcloud.entities #entity别名类

mapper-locations:

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

spring:

application:

name: microservicecloud-dept

datasource:

type: com.alibaba.druid.pool.DruidDataSource

driver-class-name: org.gjt.mm.mysql.Driver

url: jdbc:mysql://localhost:3306/cloudDB01

username: root

password: 123456

dbcp2:

min-idle: 5

initial-size: 5

max-total: 5

max-wait-millis: 200

eureka:

client: #客户端注册进eureka服务列表内

service-url:

defaultZone: http://eureka7001.com:7001/eureka/,http://eureka7002.com:7002/eureka/,http://eureka7003.com:7003/eureka/

instance:

instance-id: microservicecloud-dept8001 #自定义服务名称信息

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: $project.version$

1.   
   Ribbon负载均衡
   1. 概述
      1. 是什么

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

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



* + 1. 能干吗

LB（负载均衡）

LB，即负载均衡(Load Balance)，在微服务或分布式集群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的分配到多个服务上，从而达到系统的HA。

常见的负载均衡有软件Nginx，LVS，硬件 F5等。

相应的在中间件，例如：dubbo和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。

集中式LB

集中式LB

即在服务的消费方和提供方之间使用独立的LB设施(可以是硬件，如F5, 也可以是软件，如nginx), 由该设施负责把访问请求通过某种策略转发至服务的提供方；

进程内LB

进程内LB

将LB逻辑集成到消费方，消费方从服务注册中心获知有哪些地址可用，然后自己再从这些地址中选择出一个合适的服务器。

Ribbon就属于进程内LB，它只是一个类库，集成于消费方进程，消费方通过它来获取到服务提供方的地址。

* + 1. 官网资料

https://github.com/Netflix/ribbon/wiki/Getting-Started

* 1. Ribbon配置初步
     1. 修改microservicecloud-consumer-dept-80工程
     2. 修改pom.xml文件

内容

<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>

<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>

<!-- 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>

</dependencies>

</project>

* + 1. 修改application.yml 追加eureka的服务注册地址

内容

server:

port: 80

eureka:

client:

register-with-eureka: **false**

service-url:

defaultZone: http://eureka7001.com:7001/eureka/,http://eureka7002.com:7002/eureka/,http://eureka7003.com:7003/eureka/

* + 1. 对ConfigBean进行新注解@LoadBalanced 获得Rest时加入Ribbon的配置

**package** com.atguigu.springcloud.cfgbeans;

**import** org.springframework.cloud.client.loadbalancer.LoadBalanced;

**import** org.springframework.context.annotation.Bean;

**import** org.springframework.context.annotation.Configuration;

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

@Configuration

**public** **class** ConfigBean

{

@Bean

@LoadBalanced

**public** RestTemplate getRestTemplate()

{

**return** **new** RestTemplate();

}

}

* + 1. 主启动类DeptConsumer80\_App添加@EnableEurekaClient

**package** com.atguigu.springcloud;

**import** org.springframework.boot.SpringApplication;

**import** org.springframework.boot.autoconfigure.SpringBootApplication;

**import** org.springframework.cloud.netflix.eureka.EnableEurekaClient;

@SpringBootApplication

@EnableEurekaClient

**public** **class** DeptConsumer80\_App

{

**public** **static** **void** main(String[] args)

{

SpringApplication.*run*(DeptConsumer80\_App.**class**, args);

}

}

* + 1. 修改DeptController\_Consumer客户端访问类

**package** com.atguigu.springcloud.controller;

**import** java.util.List;

**import** org.springframework.beans.factory.annotation.Autowired;

**import** org.springframework.web.bind.annotation.PathVariable;

**import** org.springframework.web.bind.annotation.RequestMapping;

**import** org.springframework.web.bind.annotation.RestController;

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

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

@RestController

**public** **class** DeptController\_Consumer

{

//private static final String REST\_URL\_PREFIX = "http://localhost:8001";

**private** **static** **final** String ***REST\_URL\_PREFIX*** = "http://MICROSERVICECLOUD-DEPT";

@Autowired

**private** RestTemplate restTemplate;

@RequestMapping(value="/consumer/dept/add")

**public** **boolean** add(Dept dept)

{

**return** restTemplate.postForObject(***REST\_URL\_PREFIX***+"/dept/add", dept, Boolean.**class**);

}

@RequestMapping(value="/consumer/dept/get/{id}")

**public** Dept get(@PathVariable("id") Long id)

{

**return** restTemplate.getForObject(***REST\_URL\_PREFIX***+"/dept/get/"+id, Dept.**class**);

}

@SuppressWarnings("unchecked")

@RequestMapping(value="/consumer/dept/list")

**public** List<Dept> list()

{

**return** restTemplate.getForObject(***REST\_URL\_PREFIX***+"/dept/list", List.**class**);

}

}

* + 1. 先启动3个eureka集群后，再启动microservicecloud-provider-dept-8001并注册进eureka



* + 1. 启动microservicecloud-consumer-dept-80
    2. 测试

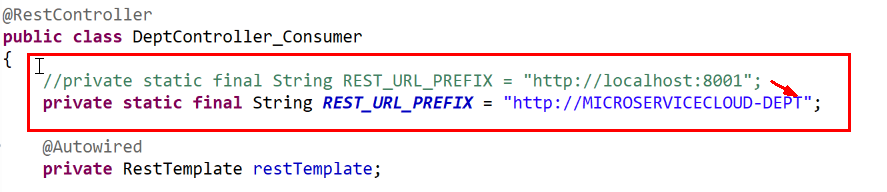
http://localhost/consumer/dept/get/1

http://localhost/consumer/dept/list

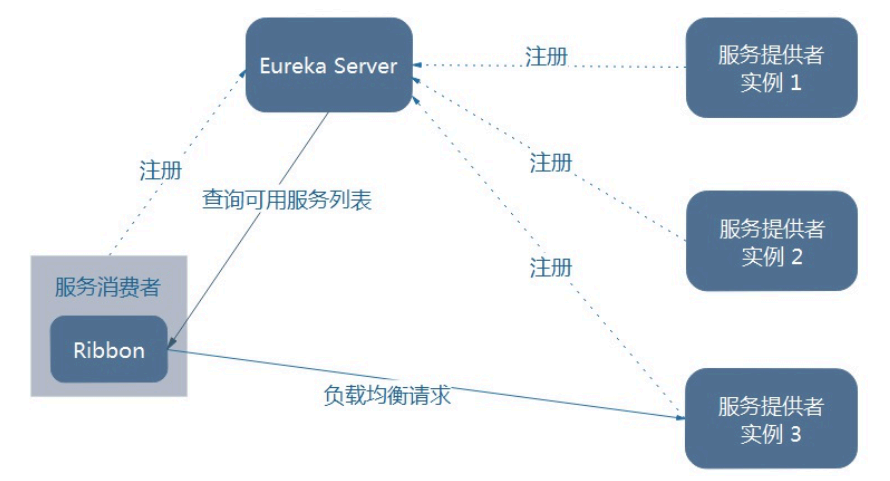
http://localhost/consumer/dept/add?dname=大数据部

* + 1. 小总结

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



* 1. Ribbon负载均衡
     1. 架构说明



Ribbon在工作时分成两步

第一步先选择 EurekaServer ,它优先选择在同一个区域内负载较少的server.

第二步再根据用户指定的策略，在从server取到的服务注册列表中选择一个地址。

其中Ribbon提供了多种策略：比如轮询、随机和根据响应时间加权。

* + 1. 参考microservicecloud-provider-dept-8001，新建两份，分别命名为8002，8003
    2. 新建8002/8003数据库，各自微服务分别连各自的数据库

8002SQL脚本

**DROP** DATABASE IF **EXISTS** cloudDB02;

**CREATE** DATABASE cloudDB02 **CHARACTER** **SET** UTF8;

USE cloudDB02;

**CREATE** **TABLE** dept

(

deptno BIGINT **NOT** **NULL** **PRIMARY** **KEY** AUTO\_INCREMENT,

dname **VARCHAR**(60),

db\_source **VARCHAR**(60)

);

**INSERT** **INTO** dept(dname,db\_source) **VALUES**('开发部',DATABASE());

**INSERT** **INTO** dept(dname,db\_source) **VALUES**('人事部',DATABASE());

**INSERT** **INTO** dept(dname,db\_source) **VALUES**('财务部',DATABASE());

**INSERT** **INTO** dept(dname,db\_source) **VALUES**('市场部',DATABASE());

**INSERT** **INTO** dept(dname,db\_source) **VALUES**('运维部',DATABASE());

**SELECT** \* **FROM** dept;

8003SQL脚本

**DROP** DATABASE IF **EXISTS** cloudDB03;

**CREATE** DATABASE cloudDB03 **CHARACTER** **SET** UTF8;

USE cloudDB03;

**CREATE** **TABLE** dept

(

deptno BIGINT **NOT** **NULL** **PRIMARY** **KEY** AUTO\_INCREMENT,

dname **VARCHAR**(60),

db\_source **VARCHAR**(60)

);

**INSERT** **INTO** dept(dname,db\_source) **VALUES**('开发部',DATABASE());

**INSERT** **INTO** dept(dname,db\_source) **VALUES**('人事部',DATABASE());

**INSERT** **INTO** dept(dname,db\_source) **VALUES**('财务部',DATABASE());

**INSERT** **INTO** dept(dname,db\_source) **VALUES**('市场部',DATABASE());

**INSERT** **INTO** dept(dname,db\_source) **VALUES**('运维部',DATABASE());

**SELECT** \* **FROM** dept;

* + 1. 修改8002/8003各自YML

8002YML

server:

port: 8002

mybatis:

config-location: classpath:mybatis/mybatis.cfg.xml #mybatis所在路径

type-aliases-package: com.atguigu.springcloud.entities #entity别名类

mapper-locations:

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

spring:

application:

name: microservicecloud-dept

datasource:

type: com.alibaba.druid.pool.DruidDataSource

driver-class-name: org.gjt.mm.mysql.Driver

url: jdbc:mysql://localhost:3306/cloudDB02

username: root

password: 123456

dbcp2:

min-idle: 5

initial-size: 5

max-total: 5

max-wait-millis: 200

eureka:

client: #客户端注册进eureka服务列表内

service-url:

defaultZone: http://eureka7001.com:7001/eureka/,http://eureka7002.com:7002/eureka/,http://eureka7003.com:7003/eureka/

instance:

instance-id: microservicecloud-dept8002 #自定义服务名称信息

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: $project.version$

8003YML

server:

port: 8003

mybatis:

config-location: classpath:mybatis/mybatis.cfg.xml #mybatis所在路径

type-aliases-package: com.atguigu.springcloud.entities #entity别名类

mapper-locations:

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

spring:

application:

name: microservicecloud-dept

datasource:

type: com.alibaba.druid.pool.DruidDataSource

driver-class-name: org.gjt.mm.mysql.Driver

url: jdbc:mysql://localhost:3306/cloudDB03

username: root

password: 123456

dbcp2:

min-idle: 5

initial-size: 5

max-total: 5

max-wait-millis: 200

eureka:

client: #客户端注册进eureka服务列表内

service-url:

defaultZone: http://eureka7001.com:7001/eureka/,http://eureka7002.com:7002/eureka/,http://eureka7003.com:7003/eureka/

instance:

instance-id: microservicecloud-dept8003 #自定义服务名称信息

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: $project.version$

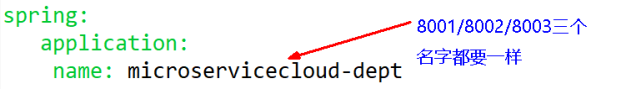
备注

端口

数据库链接



对外暴露的统一的服务实例名



* + 1. 启动3个eureka集群配置区
    2. 启动3个Dept微服务启动并各自测试通过

http://localhost:8001/dept/list

http://localhost:8002/dept/list

http://localhost:8003/dept/list

* + 1. 启动microservicecloud-consumer-dept-80
    2. 客户端通过Ribbo完成负载均衡并访问上一步的Dept微服务

http://localhost/consumer/dept/list

注意观察看到返回的数据库名字，各不相同，负载均衡实现

* + 1. 总结：Ribbon其实就是一个软负载均衡的客户端组件， 他可以和其他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。

1.   
   Feign负载均衡
   1. 概述

官网解释：

http://projects.spring.io/spring-cloud/spring-cloud.html#spring-cloud-feign

Feign是一个声明式WebService客户端。使用Feign能让编写Web Service客户端更加简单, 它的使用方法是定义一个接口，然后在上面添加注解，同时也支持JAX-RS标准的注解。Feign也支持可拔插式的编码器和解码器。Spring Cloud对Feign进行了封装，使其支持了Spring MVC标准注解和HttpMessageConverters。Feign可以与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。

参考官网：https://github.com/OpenFeign/feign

Feign能干什么

Feign旨在使编写Java Http客户端变得更容易。

前面在使用Ribbon+RestTemplate时，利用RestTemplate对http请求的封装处理，形成了一套模版化的调用方法。但是在实际开发中，由于对服务依赖的调用可能不止一处，往往一个接口会被多处调用，所以通常都会针对每个微服务自行封装一些客户端类来包装这些依赖服务的调用。所以，Feign在此基础上做了进一步封装，由他来帮助我们定义和实现依赖服务接口的定义。在Feign的实现下，我们只需创建一个接口并使用注解的方式来配置它(以前是Dao接口上面标注Mapper注解,现在是一个微服务接口上面标注一个Feign注解即可)，即可完成对服务提供方的接口绑定，简化了使用Spring cloud Ribbon时，自动封装服务调用客户端的开发量。

Feign集成了Ribbon

利用Ribbon维护了MicroServiceCloud-Dept的服务列表信息，并且通过轮询实现了客户端的负载均衡。而与Ribbon不同的是，通过feign只需要定义服务绑定接口且以声明式的方法，优雅而简单的实现了服务调用

* + 1. 
  1. Feign使用步骤
     1. 参考microservicecloud-consumer-dept-80
     2. 新建microservicecloud-consumer-dept-feign

修改主启动类名字

DeptConsumer80\_Feign\_App

* + 1. microservicecloud-consumer-dept-feign工程pom.xml修改，主要添加对feign的支持

<dependency>

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

<artifactId>spring-cloud-starter-feign</artifactId>

</dependency>

* + 1. 修改microservicecloud-api工程

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><!-- 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>

新建DeptClientService接口并新增注解@FeignClient

**package** com.atguigu.springcloud.service;

**import** java.util.List;

**import** org.springframework.cloud.netflix.feign.FeignClient;

**import** org.springframework.web.bind.annotation.PathVariable;

**import** org.springframework.web.bind.annotation.RequestMapping;

**import** org.springframework.web.bind.annotation.RequestMethod;

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

@FeignClient(value = "MICROSERVICECLOUD-DEPT")

**public** **interface** DeptClientService

{

@RequestMapping(value = "/dept/get/{id}",method = RequestMethod.***GET***)

**public** Dept get(@PathVariable("id") **long** id);

@RequestMapping(value = "/dept/list",method = RequestMethod.***GET***)

**public** List<Dept> list();

@RequestMapping(value = "/dept/add",method = RequestMethod.***POST***)

**public** **boolean** add(Dept dept);

}

@FeignClient

另见: [@EnableFeignClients](#a_EnableFeignClients)

mvn clean

mvn install

* + 1. microservicecloud-consumer-dept-feign工程修改Controller，添加上一步新建的DeptClientService接口

**package** com.atguigu.springcloud.controller;

**import** java.util.List;

**import** org.springframework.beans.factory.annotation.Autowired;

**import** org.springframework.web.bind.annotation.PathVariable;

**import** org.springframework.web.bind.annotation.RequestMapping;

**import** org.springframework.web.bind.annotation.RestController;

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

**import** com.atguigu.springcloud.service.DeptClientService;

@RestController

**public** **class** DeptController\_Feign

{

@Autowired

**private** DeptClientService service = **null**;

@RequestMapping(value = "/consumer/dept/get/{id}")

**public** Dept get(@PathVariable("id") Long id)

{

**return** **this**.service.get(id);

}

@RequestMapping(value = "/consumer/dept/list")

**public** List<Dept> list()

{

**return** **this**.service.list();

}

@RequestMapping(value = "/consumer/dept/add")

**public** Object add(Dept dept)

{

**return** **this**.service.add(dept);

}

}

* + 1. microservicecloud-consumer-dept-feign工程修改主启动类

**package** com.atguigu.springcloud;

**import** org.springframework.boot.SpringApplication;

**import** org.springframework.boot.autoconfigure.SpringBootApplication;

**import** org.springframework.cloud.netflix.eureka.EnableEurekaClient;

**import** org.springframework.cloud.netflix.feign.EnableFeignClients;

@SpringBootApplication

@EnableEurekaClient

@EnableFeignClients(basePackages= {"com.atguigu.springcloud"})

@ComponentScan("com.atguigu.springcloud")

**public** **class** DeptConsumer80\_Feign\_App

{

**public** **static** **void** main(String[] args)

{

SpringApplication.*run*(DeptConsumer80\_Feign\_App.**class**, args);

}

}

@EnableFeignClients

另见: [@FeignClient](#a_FeignClient)

* + 1. 测试

启动3个eureka集群

启动3个部门微服务8001/8002/8003

启动Feign自己启动

http://localhost/consumer/dept/list

Feign自带负载均衡配置项

* + 1. 小总结

Feign通过接口的方法调用Rest服务（之前是Ribbon+RestTemplate），

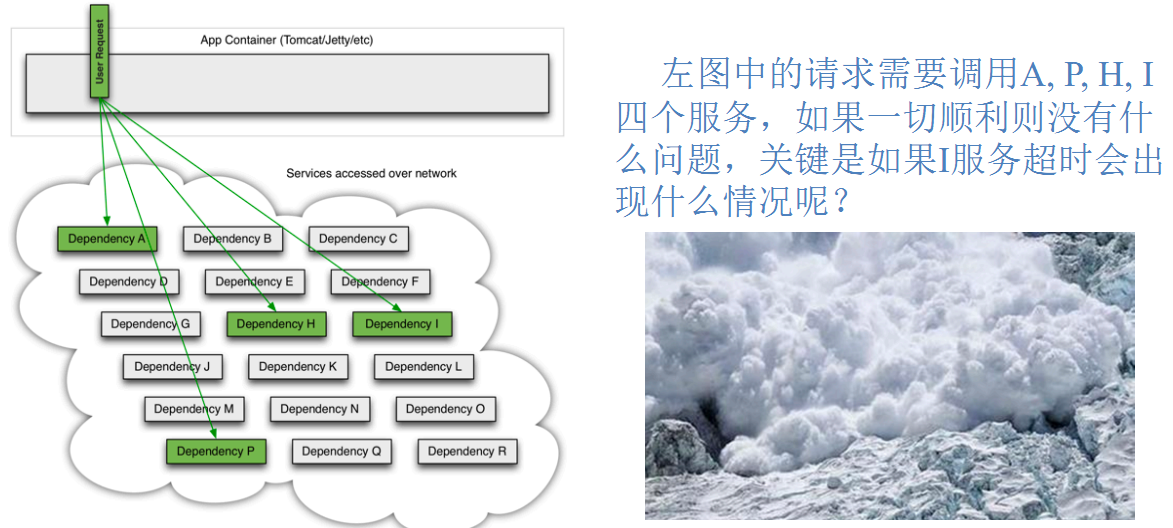
该请求发送给Eureka服务器（http://MICROSERVICECLOUD-DEPT/dept/list）,

通过Feign直接找到服务接口，由于在进行服务调用的时候融合了Ribbon技术，所以也支持负载均衡作用。

1.   
   Hystrix断路器
   1. 概述
      1. 分布式系统面临的问题

分布式系统面临的问题

复杂分布式体系结构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地失败。



服务雪崩

多个微服务之间调用的时候，假设微服务A调用微服务B和微服务C，微服务B和微服务C又调用其它的微服务，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，进而引起系统崩溃，所谓的“雪崩效应”.

对于高流量的应用来说，单一的后端依赖可能会导致所有服务器上的所有资源都在几秒钟内饱和。比失败更糟糕的是，这些应用程序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，导致整个系统发生更多的级联故障。这些都表示需要对故障和延迟进行隔离和管理，以便单个依赖关系的失败，不能取消整个应用程序或系统。

* + 1. 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，在分布式系统里，许多依赖不可避免的会调用失败，比如超时、异常等，Hystrix能够保证在一个依赖出问题的情况下，不会导致整体服务失败，避免级联故障，以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之后，通过断路器的故障监控（类似熔断保险丝），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无法处理的异常，这样就保证了服务调用方的线程不会被长时间、不必要地占用，从而避免了故障在分布式系统中的蔓延，乃至雪崩。



* + 1. 能干嘛

服务降级

服务熔断

服务限流

接近实时的监控

。。。。。。

* + 1. 官网资料

https://github.com/Netflix/Hystrix/wiki/How-To-Use

* 1. 服务熔断
     1. 是什么

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。

当扇出链路的某个微服务不可用或者响应时间太长时，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"的响应信息。当检测到该节点微服务调用响应正常后恢复调用链路。在SpringCloud框架里熔断机制通过Hystrix实现。Hystrix会监控微服务间调用的状况，当失败的调用到一定阈值，缺省是5秒内20次调用失败就会启动熔断机制。熔断机制的注解是@HystrixCommand。

* + 1. 参考microservicecloud-provider-dept-8001
    2. 新建microservicecloud-provider-dept-hystrix-8001
    3. POM

修改内容

<!-- hystrix -->

<dependency>

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

<artifactId>spring-cloud-starter-hystrix</artifactId>

</dependency>

全部内容

<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-hystrix-8001</artifactId>

<dependencies>

<!-- hystrix -->

<dependency>

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

<artifactId>spring-cloud-starter-hystrix</artifactId>

</dependency>

<!-- 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>

<!-- 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>

<!-- actuator监控信息完善 -->

<dependency>

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

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

* + 1. YML

server:

port: 8001

mybatis:

config-location: classpath:mybatis/mybatis.cfg.xml #mybatis所在路径

type-aliases-package: com.atguigu.springcloud.entities #entity别名类

mapper-locations:

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

spring:

application:

name: microservicecloud-dept

datasource:

type: com.alibaba.druid.pool.DruidDataSource

driver-class-name: org.gjt.mm.mysql.Driver

url: jdbc:mysql://localhost:3306/cloudDB01

username: root

password: 123456

dbcp2:

min-idle: 5

initial-size: 5

max-total: 5

max-wait-millis: 200

eureka:

client: #客户端注册进eureka服务列表内

service-url:

defaultZone: http://eureka7001.com:7001/eureka/,http://eureka7002.com:7002/eureka/,http://eureka7003.com:7003/eureka/

instance:

instance-id: microservicecloud-dept8001-hystrix #自定义服务名称信息

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: $project.version$

* + 1. 修改DeptController

@HystrixCommand报异常后如何处理



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

代码内容

**package** com.atguigu.springcloud.controller;

**import** org.springframework.beans.factory.annotation.Autowired;

**import** org.springframework.web.bind.annotation.PathVariable;

**import** org.springframework.web.bind.annotation.RequestMapping;

**import** org.springframework.web.bind.annotation.RequestMethod;

**import** org.springframework.web.bind.annotation.RestController;

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

**import** com.atguigu.springcloud.service.DeptService;

**import** com.netflix.hystrix.contrib.javanica.annotation.HystrixCommand;

@RestController

**public** **class** DeptController

{

@Autowired

**private** DeptService service = **null**;

@RequestMapping(value="/dept/get/{id}",method=RequestMethod.***GET***)

@HystrixCommand(fallbackMethod = "processHystrix\_Get")

**public** Dept get(@PathVariable("id") Long id)

{

Dept dept = **this**.service.get(id);

**if**(**null** == dept)

{

**throw** **new** RuntimeException("该ID："+id+"没有没有对应的信息");

}

**return** dept;

}

**public** Dept processHystrix\_Get(@PathVariable("id") Long id)

{

**return** **new** Dept().setDeptno(id)

.setDname("该ID："+id+"没有没有对应的信息,null--@HystrixCommand")

.setDb\_source("no this database in MySQL");

}

}

* + 1. 修改主启动类DeptProvider8001\_Hystrix\_App并添加新注解@EnableCircuitBreaker

**package** com.atguigu.springcloud;

**import** org.springframework.boot.SpringApplication;

**import** org.springframework.boot.autoconfigure.SpringBootApplication;

**import** org.springframework.cloud.client.circuitbreaker.EnableCircuitBreaker;

**import** org.springframework.cloud.client.discovery.EnableDiscoveryClient;

**import** org.springframework.cloud.netflix.eureka.EnableEurekaClient;

@SpringBootApplication

@EnableEurekaClient //本服务启动后会自动注册进eureka服务中

@EnableCircuitBreaker//对hystrixR熔断机制的支持

**public** **class** DeptProvider8001\_Hystrix\_App

{

**public** **static** **void** main(String[] args)

{

SpringApplication.*run*(DeptProvider8001\_Hystrix\_App.**class**, args);

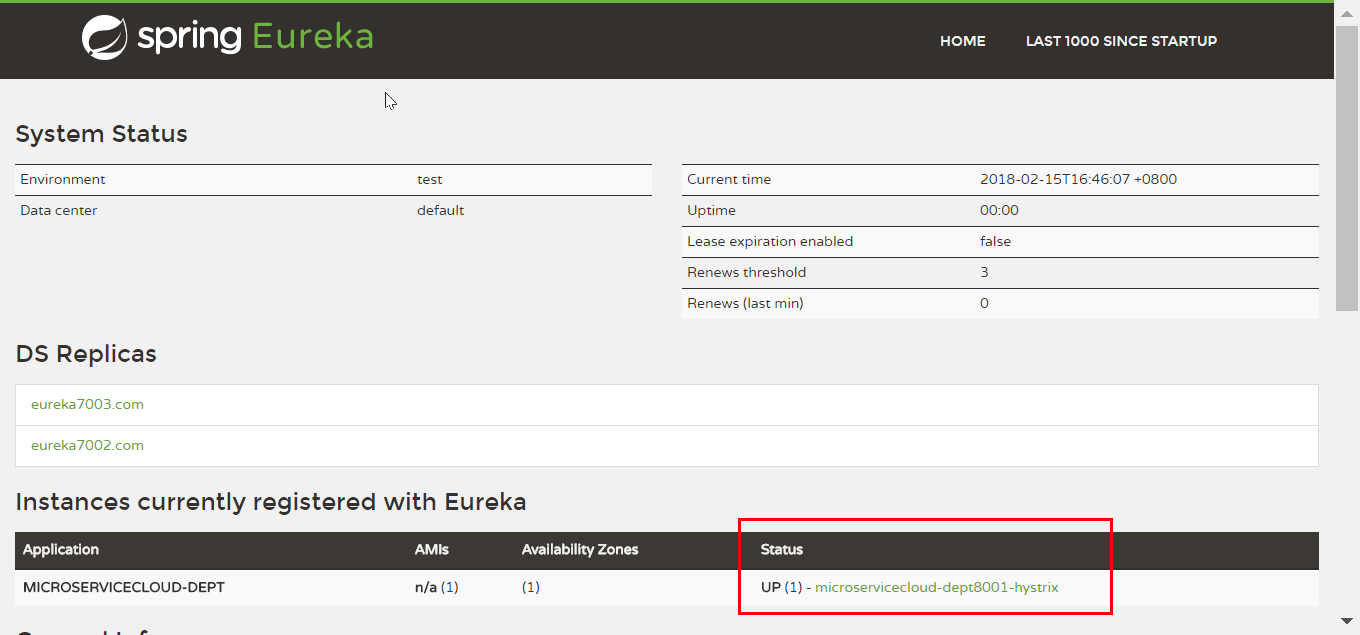
}

}

* + 1. 测试

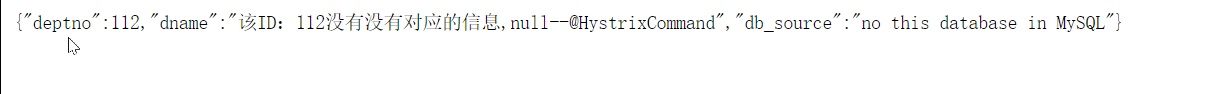
3个eureka先启动

主启动类DeptProvider8001\_Hystrix\_App



Consumer启动microservicecloud-consumer-dept-80

http://localhost/consumer/dept/get/112



如果对应的ID：112，数据库里面没有这个记录，我们报错后统一返回。

* 1. 服务降级
     1. 是什么

整体资源快不够了，忍痛将某些服务先关掉，待渡过难关，再开启回来。

* + 1. 服务降级处理是在客户端实现完成的，与服务端没有关系
    2. 修改microservicecloud-api工程， 根据已经有的DeptClientService接口新建一个实现了 FallbackFactory接口的类DeptClientServiceFallbackFactory

**package** com.atguigu.springcloud.service;

**import** java.util.List;

**import** org.springframework.stereotype.Component;

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

**import** feign.hystrix.FallbackFactory;

@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** Dept get(**long** id)

{

**return** **new** Dept().setDeptno(id)

.setDname("该ID："+id+"没有没有对应的信息,Consumer客户端提供的降级信息,此刻服务Provider已经关闭")

.setDb\_source("no this database in MySQL");

}

@Override

**public** List<Dept> list()

{

**return** **null**;

}

@Override

**public** **boolean** add(Dept dept)

{

**return** **false**;

}

};

}

}

千万不要忘记在类上面新增@Component注解，大坑！！！

* + 1. 修改microservicecloud-api工程，DeptClientService接口在注解@FeignClient中添加fallbackFactory属性值

**package** com.atguigu.springcloud.service;

**import** java.util.List;

**import** org.springframework.cloud.netflix.feign.FeignClient;

**import** org.springframework.web.bind.annotation.PathVariable;

**import** org.springframework.web.bind.annotation.RequestMapping;

**import** org.springframework.web.bind.annotation.RequestMethod;

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

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

**public** **interface** DeptClientService

{

@RequestMapping(value = "/dept/get/{id}",method = RequestMethod.***GET***)

**public** Dept get(@PathVariable("id") **long** id);

@RequestMapping(value = "/dept/list",method = RequestMethod.***GET***)

**public** List<Dept> list();

@RequestMapping(value = "/dept/add",method = RequestMethod.***POST***)

**public** **boolean** add(Dept dept);

}

* + 1. microservicecloud-api工程

mvn clean install

* + 1. microservicecloud-consumer-dept-feign工程修改YML

YML

server:

port: 80

feign:

hystrix:

enabled: **true**

eureka:

client:

register-with-eureka: **false**

service-url:

defaultZone: http://eureka7001.com:7001/eureka/,http://eureka7002.com:7002/eureka/,http://eureka7003.com:7003/eureka/

* + 1. 测试

3个eureka先启动

微服务microservicecloud-provider-dept-8001启动

microservicecloud-consumer-dept-feign启动

正常访问测试

http://localhost/consumer/dept/get/1

故意关闭微服务microservicecloud-provider-dept-8001

客户端自己调用提示

http://localhost/consumer/dept/get/1



此时服务端provider已经down了，但是我们做了服务降级处理，让客户端在服务端不可用时也会获得提示信息而不会挂起耗死服务器

* 1. 服务监控hystrixDashboard
     1. 概述

除了隔离依赖服务的调用以外，Hystrix还提供了准实时的调用监控（Hystrix Dashboard），Hystrix会持续地记录所有通过Hystrix发起的请求的执行信息，并以统计报表和图形的形式展示给用户，包括每秒执行多少请求多少成功，多少失败等。Netflix通过hystrix-metrics-event-stream项目实现了对以上指标的监控。Spring Cloud也提供了Hystrix Dashboard的整合，对监控内容转化成可视化界面。

* + 1. Case步骤

新建工程microservicecloud-consumer-hystrix-dashboard

POM

修改内容

<!-- 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>

全部内容

<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-hystrix-dashboard</artifactId>

<dependencies>

<!-- 自己定义的api -->

<dependency>

<groupId>com.atguigu.springcloud</groupId>

<artifactId>microservicecloud-api</artifactId>

<version>${project.version}</version>

</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>

<!-- 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>

<!-- feign相关 -->

<dependency>

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

<artifactId>spring-cloud-starter-feign</artifactId>

</dependency>

<!-- 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>

</dependencies>

</project>

YML

server:

port: 9001

主启动类改名+新注解@EnableHystrixDashboard

**package** com.atguigu.springcloud;

**import** org.springframework.boot.SpringApplication;

**import** org.springframework.boot.autoconfigure.SpringBootApplication;

**import** org.springframework.cloud.netflix.hystrix.dashboard.EnableHystrixDashboard;

@SpringBootApplication

@EnableHystrixDashboard

**public** **class** DeptConsumer\_DashBoard\_App

{

**public** **static** **void** main(String[] args)

{

SpringApplication.*run*(DeptConsumer\_DashBoard\_App.**class**,args);

}

}

所有Provider微服务提供类(8001/8002/8003)都需要监控依赖配置

<!-- actuator监控信息完善 -->

<dependency>

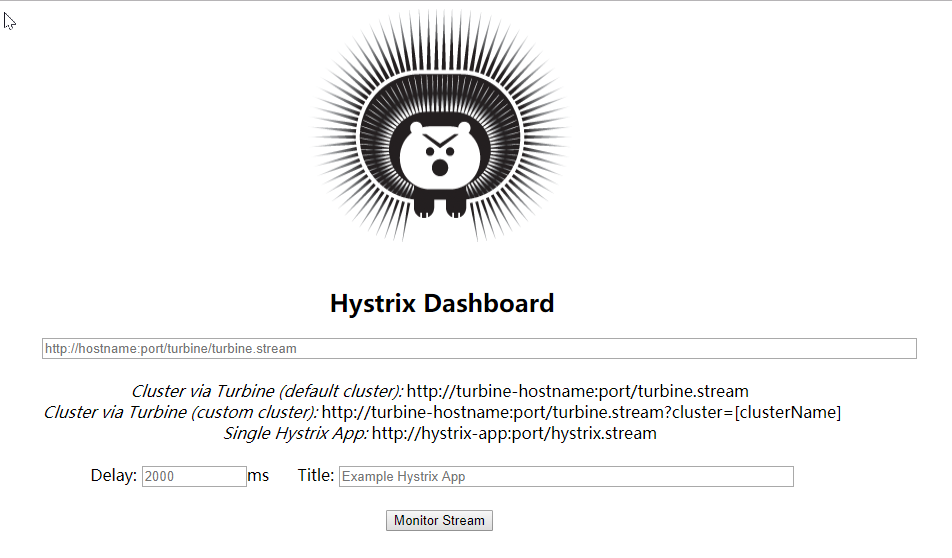
<groupId>org.springframework.boot</groupId>

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

</dependency>

启动microservicecloud-consumer-hystrix-dashboard该微服务监控消费端

http://localhost:9001/hystrix



启动3个eureka集群

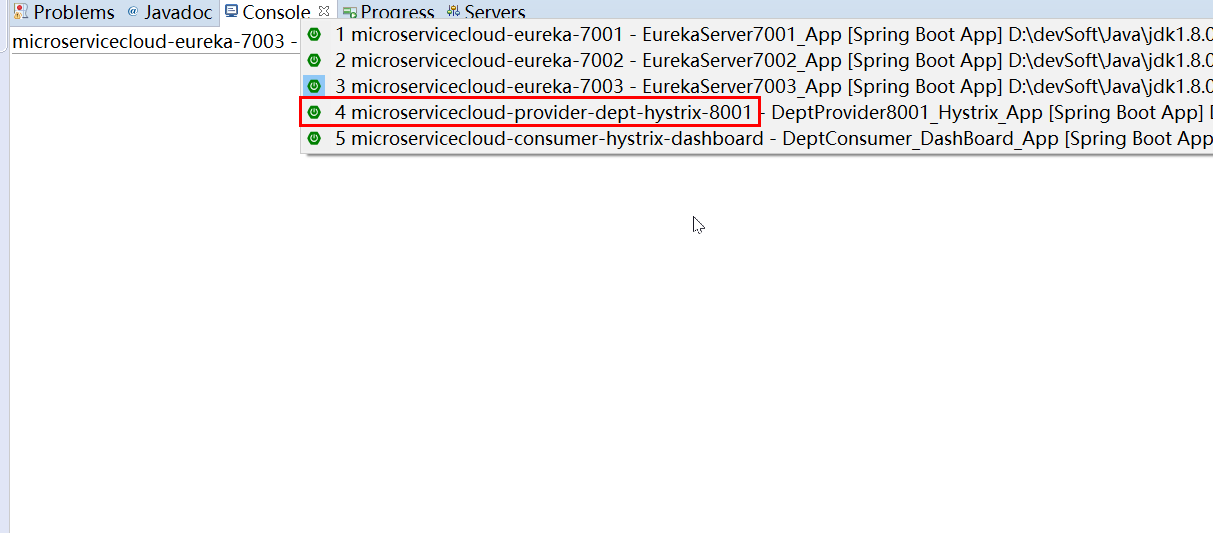
启动microservicecloud-provider-dept-hystrix-8001

http://localhost:8001/dept/get/1

http://localhost:8001/hystrix.stream



启动的相关微服务工程

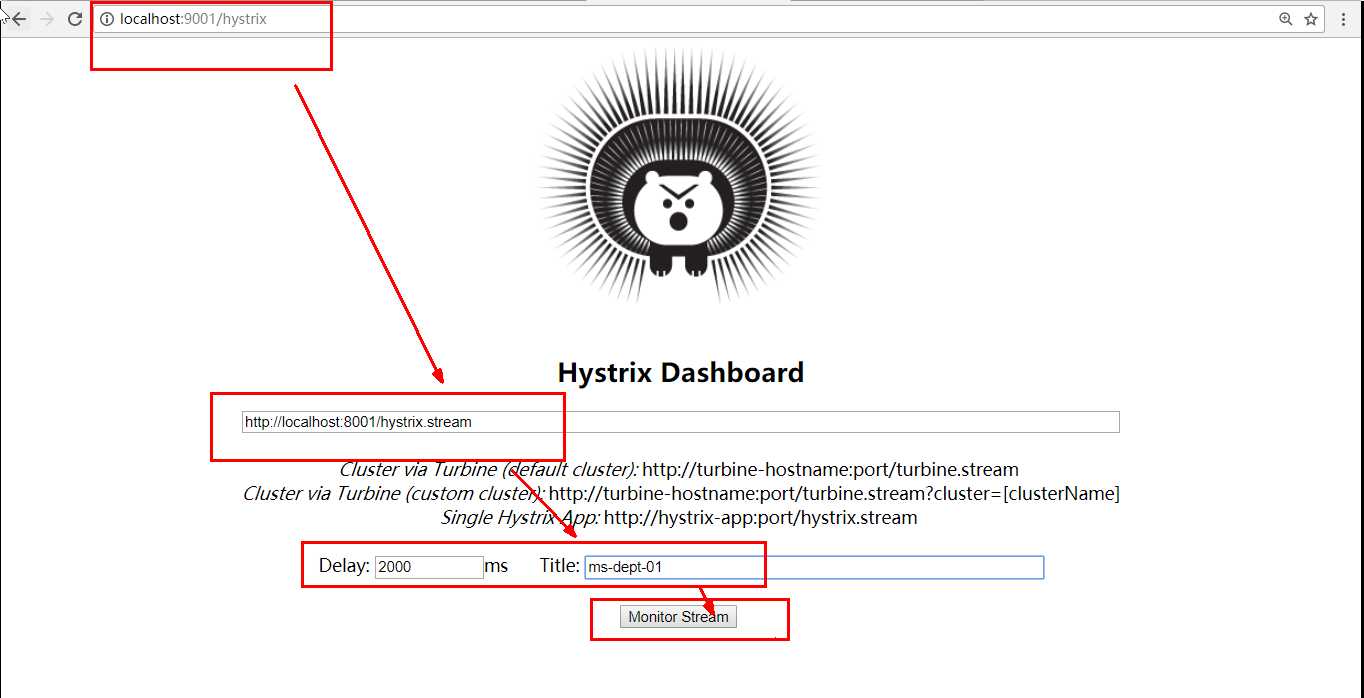


监控测试

多次刷新http://localhost:8001/dept/get/1

观察监控窗口

填写监控地址



1：Delay：该参数用来控制服务器上轮询监控信息的延迟时间，默认为2000毫秒，可以通过配置该属性来降低客户端的网络和CPU消耗。

2：Title：该参数对应了头部标题Hystrix Stream之后的内容，默认会使用具体监控实例的URL，可以通过配置该信息来展示更合适的标题。

监控结果



如何看上图

如何看？

7色

1圈

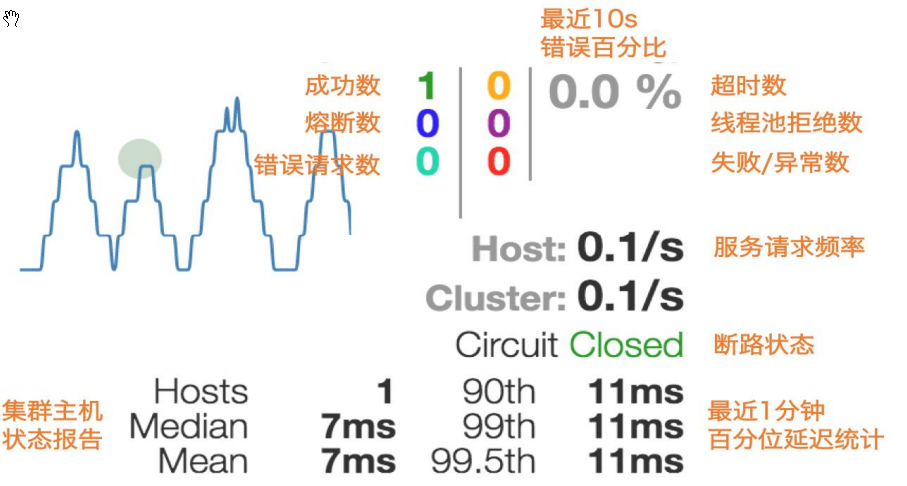
实心圆：共有两种含义。它通过颜色的变化代表了实例的健康程度，它的健康度从绿色<黄色<橙色<红色递减。

该实心圆除了颜色的变化之外，它的大小也会根据实例的请求流量发生变化，流量越大该实心圆就越大。所以通过该实心圆的展示，就可以在大量的实例中快速的发现故障实例和高压力实例。

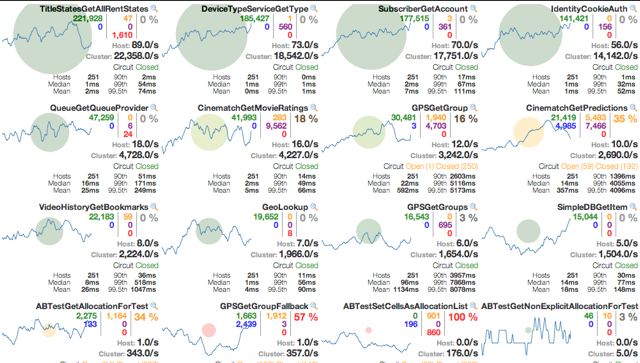
1线

曲线：用来记录2分钟内流量的相对变化，可以通过它来观察到流量的上升和下降趋势。

整图说明



搞懂一个才能看懂复杂的



1.   
   zuul路由网关
   1. 概述
      1. 是什么

Zuul包含了对请求的路由和过滤两个最主要的功能：

其中路由功能负责将外部请求转发到具体的微服务实例上，是实现外部访问统一入口的基础而过滤器功能则负责对请求的处理过程进行干预，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服务治理下的应用，同时从Eureka中获得其他微服务的消息，也即以后的访问微服务都是通过Zuul跳转后获得。

注意：Zuul服务最终还是会注册进Eureka

提供=代理+路由+过滤三大功能

* + 1. 能干嘛

路由

过滤

* + 1. 官网资料

https://github.com/Netflix/zuul/wiki/Getting-Started

* 1. 路由基本配置
     1. 新建Module模块microservicecloud-zuul-gateway-9527
     2. POM

修改内容

<dependency>

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

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

</dependency>

<dependency>

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

<artifactId>spring-cloud-starter-zuul</artifactId>

</dependency>

全部内容

<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-zuul-gateway-9527</artifactId>

<dependencies>

<!-- 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>

<!-- actuator监控 -->

<dependency>

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

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

</dependency>

<!-- hystrix容错-->

<dependency>

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

<artifactId>spring-cloud-starter-hystrix</artifactId>

</dependency>

<dependency>

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

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

</dependency>

<!-- 日常标配 -->

<dependency>

<groupId>com.atguigu.springcloud</groupId>

<artifactId>microservicecloud-api</artifactId>

<version>${project.version}</version>

</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>

<!-- 热部署插件 -->

<dependency>

<groupId>org.springframework</groupId>

<artifactId>springloaded</artifactId>

</dependency>

<dependency>

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

<artifactId>spring-boot-devtools</artifactId>

</dependency>

</dependencies>

</project>

* + 1. 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://eureka7003.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: $project.version$

* + 1. hosts修改

127.0.0.1 myzuul.com

* + 1. 主启动类

**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)

{

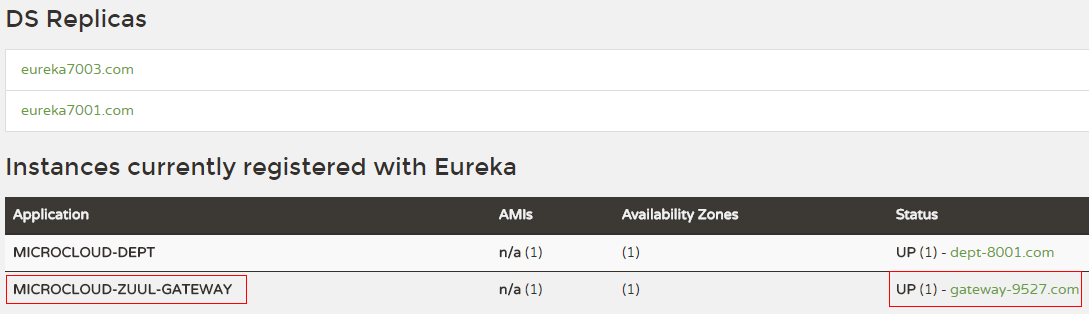
SpringApplication.*run*(Zuul\_9527\_StartSpringCloudApp.**class**, args);

}

}

@EnableZuulProxy

* + 1. 启动



三个eureka集群

一个服务提供类microservicecloud-provider-dept-8001

一个路由

* + 1. 测试

不用路由

http://localhost:8001/dept/get/2

启用路由

http://myzuul.com:9527/microservicecloud-dept/dept/get/2

* 1. 路由访问映射规则
     1. 工程microservicecloud-zuul-gateway-9527
     2. 代理名称

YML

before

http://myzuul.com:9527/microservicecloud-dept/dept/get/2

|  |
| --- |
| zuul:  routes:  mydept.serviceId: microservicecloud-dept  mydept.path: /mydept/\*\* |

after

http://myzuul.com:9527/mydept/dept/get/1

此时问题

路由访问OK

http://myzuul.com:9527/mydept/dept/get/1

原路径访问OK

http://myzuul.com:9527/microservicecloud-dept/dept/get/2

* + 1. 原真实服务名忽略

YML

zuul:

ignored-services: microservicecloud-dept

routes:

mydept.serviceId: microservicecloud-dept

mydept.path: /mydept/\*\*

单个具体，多个可以用"\*"

zuul:

ignored-services: "\*"

routes:

mydept.serviceId: microservicecloud-dept

mydept.path: /mydept/\*\*

* + 1. 设置统一公共前缀

YML

zuul:

prefix: /atguigu

ignored-services: "\*"

routes:

mydept.serviceId: microservicecloud-dept

mydept.path: /mydept/\*\*

http://myzuul.com:9527/atguigu/mydept/dept/get/1

* + 1. 最后YML

server:

port: 9527

spring:

application:

name: microservicecloud-zuul-gateway

zuul:

prefix: /atguigu

ignored-services: "\*"

routes:

mydept.serviceId: microservicecloud-dept

mydept.path: /mydept/\*\*

eureka:

client:

service-url:

defaultZone: http://eureka7001.com:7001/eureka,http://eureka7002.com:7002/eureka,http://eureka7003.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: $project.version$