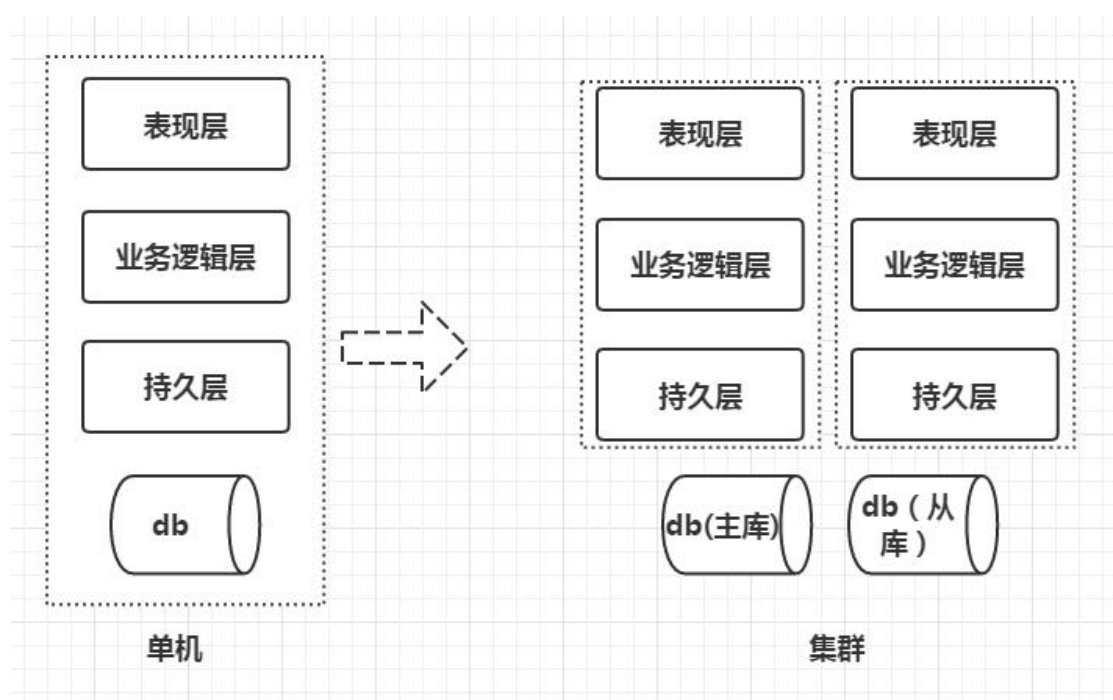


# SpringCloud 技术文档整理

## 一. 微服务架构衍变

### 1.1 单体架构



在应用初期，**单体应用从成本、开发时间和运维等方面都有明显的优势**，但是随着业务量和用户量的增加，导致代码的可读性和可维护性很差。同时面对海量的用户，数据库也会成为瓶颈。单体架构已经不能满足复杂的业务和海量的用户系统，改变单体架构势在必行。

### 1.2 SOA 架构



Service Oriented Architecture 面向服务的架构。也就是把**工程拆分成服务层、表现层两个工程**。服务层中包含业务逻辑，只需要对外提供服务即可。表现层只需要处理和页面的交互，业务逻辑都是调用服务层的服务来实现。

SOA 架构下的系统编码更加灵活，提高了代码的重用性，且易于维护，同时提升了系统的伸缩性和高可用性，降低系统之间的耦合性。

### 1.3 微服务架构

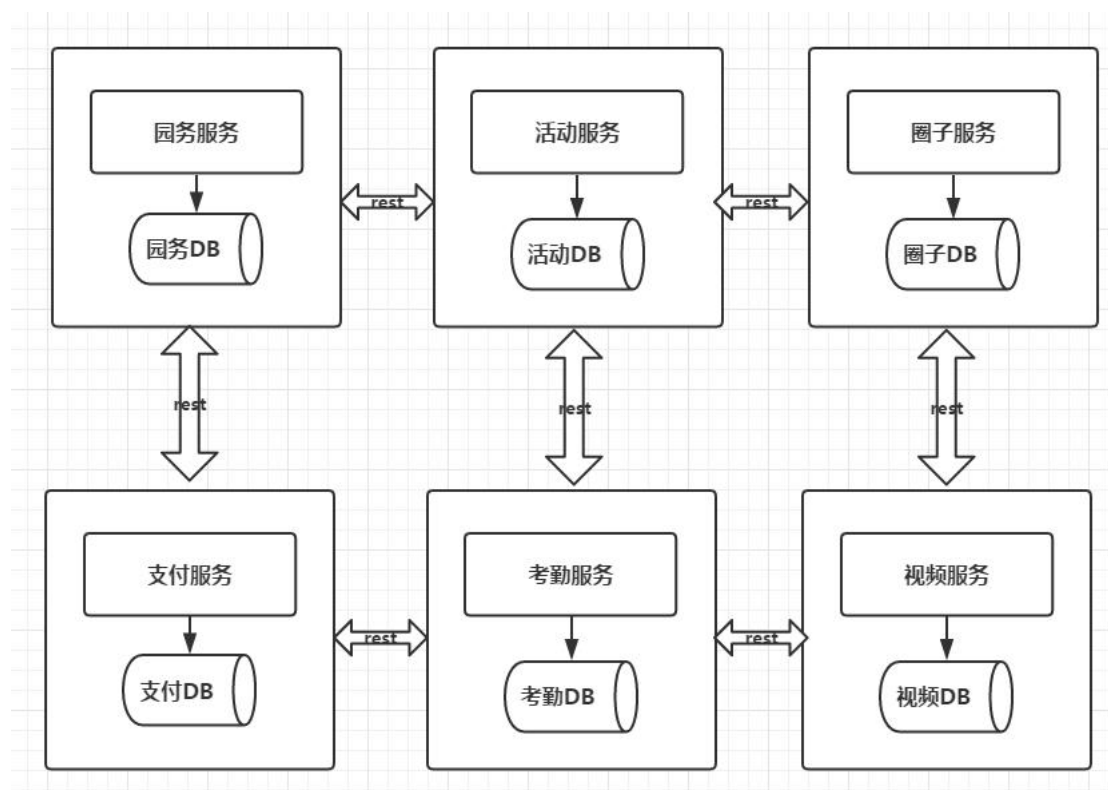
“微服务”最初是由 软件工程师 Martin Fowler 2014 年写的《MicroServices》中提出来的。

**微服务架构是将复杂的业务组件和，实际上也是一种面向服务的体现，他说 SOA 架构的一种实现，但是相比传统的 SOA 架构来讲它更为轻便、敏捷和简单。**微服务架构的风格，就是将单一程序开发成一个微服务，每个微服务运行在自己的进程中，并使用轻量级机制通信，通常是 HTTP RESTFUL API。这些服务围绕业务能力来划分构建的，并通过完全自动化部署

讲师：闫洪波

微信：769828176

机制来独立部署 这些服务可以使用不同的编程语言，以及不同数据存储技术，以保证最低限度的集中式管理。



优点：

1. 由于每个服务都是独立并且微小的,由单独的团队负责,仍然可以采用敏捷开发模式,自由的选择合适的技术,甚至可以重写老服务,当然都要遵守统一的 API 约定。
2. 每一个微服务都是独立部署的,可以进行快速迭代部署,根据各自服务需求选择合适的虚拟机和使用最匹配的服务资源要求的硬件。
3. 降低系统之间的耦合性,因为采用 HTTP 方式交互,各个微服务可以使用不同的编程语言实现。

缺点：

1. 微服务应用作为分布式系统带来了复杂性。各个微服务进行分布式独立部署,当进行

模块调用的时候，分布式将会变得更加麻烦。

2. 微服务架构一般使用各个独立数据库，分布式事务的实现更具挑战性。
3. 测试微服务变得复杂，当一个服务依赖另外一个服务时，测试时候需要另外一个服务的支持。
4. 部署基于微服务的应用也很复杂，独立微服务的部署不但变得复杂，而且需要更高级别的自动化。

## 二. SpringCloud 概述

Spring Cloud 的首要目标就是通过提供 系列开发组件和框架，帮助开发者迅速搭建分布式的微服务系统。Spring Cloud 是通过包装其他技术框架来实现的，例如包装开源的 **Netflix oss 组件**，实现了一套通过基于注解、Java 配置和基于模版开发的微服务框架。Spring Cloud 框架来自于 Spring Resources 社区，由 **Pivotal 和 Netflix** 两大公司和一些其他的开发者提供技术上的更新迭代。

Spring Cloud 是一系列框架的有序集合。它利用 Spring Boot 的开发便利性巧妙地简化了分布式系统基础设施的开发，如服务发现注册、配置中心、消息总线、负载均衡、断路器、数据监控等，都可以用 Spring Boot 的开发风格做到一键启动和部署。Spring Cloud 是基于 Spring Boot 。

### 2.1 SpringCloud 的特性

Spring Cloud 专注于提供良好的**开箱即用**经验的典型用例和可**扩展性机制**覆盖。

分布式/版本化配置

服务注册和发现

路由

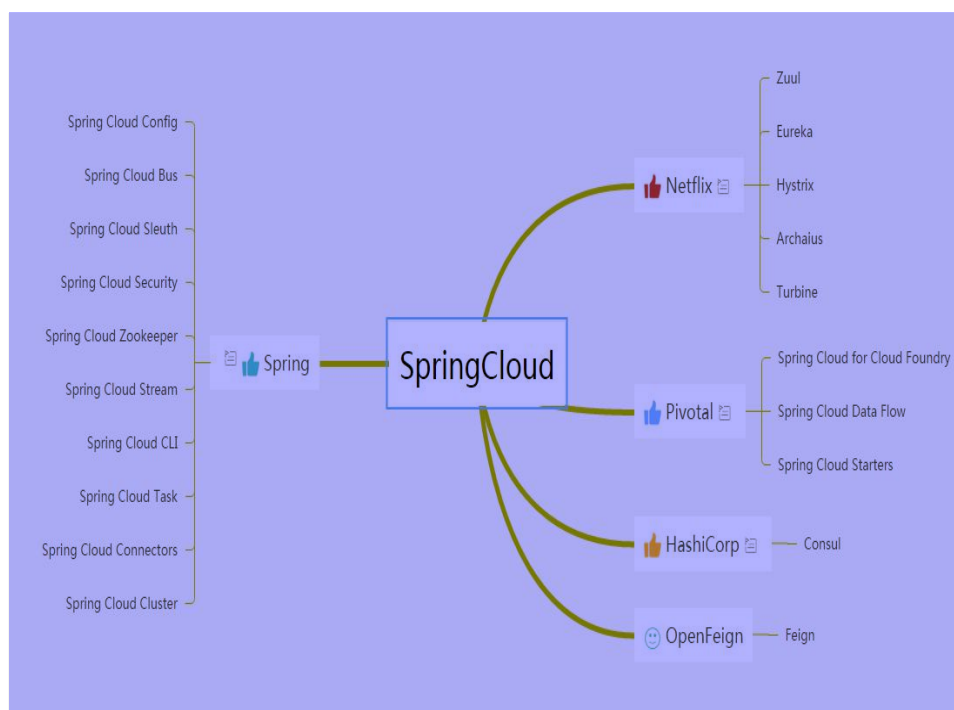
service - to - service 调用

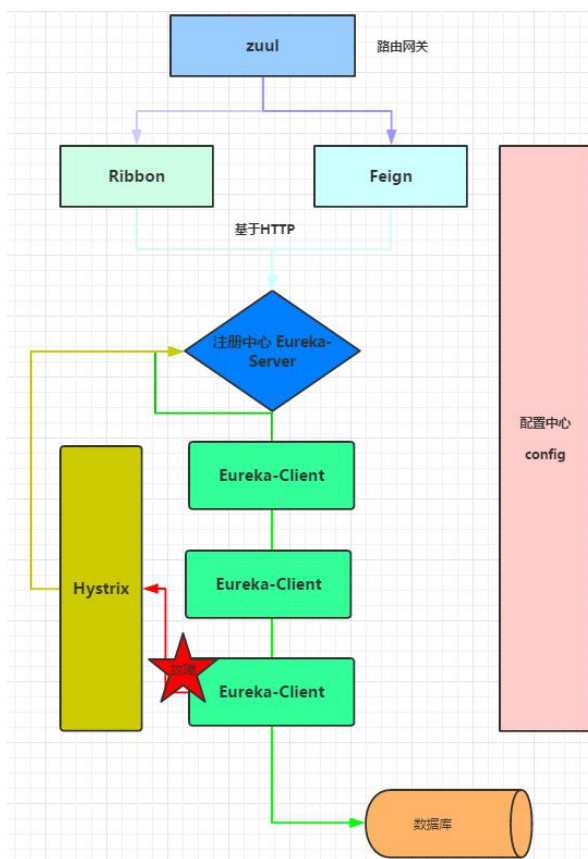
负载均衡

断路器

分布式消息传递

## 2.2 SpringCloud 的组成





### 1. 服务注册和发现组件 Eureka

利用 Eureka 组件可以很轻松地实现服务的注册和发现的功能。

### 2. 熔断组件 Hystrix

它提供了熔断器功能，能够阻止分布式系统中出现联动故障。

### 3. 负载均衡组件 Ribbon

Ribbon 是将负载均衡逻辑封装在客户端中，并且运行在客户端的进程里。

### 4. 声明式调用 Feign

Feign 使得服务调用变得更简单优雅。

### 5. 路由网关 Zuul

Zuul 作为微服务系统的网关组件，用于构建边界服务，致力于动态路由、过滤、监控、

弹性伸缩和安全。

## 6. Spring Cloud Config

Config Server 可以从本地仓库读取配置文件，也可以从远处 Git 仓库读取。是 SpringCloud 分布式开发下的配置中心组件。

## 2.3 SpringCloud 的版本

### Documentation

Each **Spring project** has its own; it explains in great details how you can use **project features** and what you can achieve with them.

Greenwich <b>CURRENT</b> <b>GA</b>	Reference Doc.	API Doc.
Greenwich <b>SNAPSHOT</b>	Reference Doc.	API Doc.
<b>Finchley SR2</b> <b>GA</b>	Reference Doc.	API Doc.
Finchley <b>SNAPSHOT</b>	Reference Doc.	API Doc.
Edgware SR5 <b>GA</b>	Reference Doc.	API Doc.
Edgware <b>SNAPSHOT</b>	Reference Doc.	API Doc.
Dalston SR5 <b>GA</b>	Reference Doc.	API Doc.

BUILD-XXX	开发版	开发团队内部使用，不是很稳定
GA	稳定版	相比于开发版，基本上可以使用了
PRE (M1、M2)	里程碑版	主要是修复了一些 BUG 的版本，一个 GA 后通常有多个里程碑版
RC	候选发布版	该阶段的软件类似于最终版的一个发行观察期，基本只修复比较严重的 BUG



SR	正式发布版	
----	-------	--

注意：

Finchley 与 Spring Boot 2.0.x，兼容，不支持 Spring Boot 1.5.x.

Dalston 和 Edgware 与 Spring Boot 1.5.x，兼容，不支持 Spring Boot 2.0.x.

## 2.4 SpringCloud 和 Dubbo 的比较

表 2-1 从微服务关注点比较 Spring Cloud 和 Dubbo

微服务关注点	Spring Cloud	Dubbo
配置管理	Config	—
服务发现	Eureka、Consul、Zookeeper	Zookeeper
负载均衡	Ribbon	自带
网关	Zuul	—
分布式追踪	Spring Cloud Sleuth	—
容错	Hystrix	不完善
通信方式	HTTP、Message	RPC
安全模块	Spring Cloud Security	—

Spring Cloud 有很多的项目模块，包含了微服务系统的方方面面，Dubbo 是一个非常优秀的服务治理和服务调用框架，但缺少很多功能模块，例如网关、链路追踪等。在项目模块上，Spring Cloud 占据更大的优势。从开发速度上讲，Spring Cloud 基于 SpringBoot 注解无配置的开发模式，具有更高的开发和部署速度。

Spring Cloud 的通信方式大多数是基于 HTTP Restful 风格的，服务与服务之无关、无耦合。由于采用的是 HTTP Rest，此服务无关乎语言和平台，只需要提供 API 接口就可实现相互调用。Dubbo 的通信方式基于远程调用，对接口、平台和语言有较强依。如果需要实现跨平台调用服务，需要写额外的中间件。

基于 RPC 协议的 Dubbo 速率上要比基于 Http 的 Spring Cloud 速率高很多。



使用一个 POJO 对象包含 10 个属性，请求 10 万次，Dubbo 和 Spring Cloud 在不同的线程数量下，每次请求耗时（ms）如下：

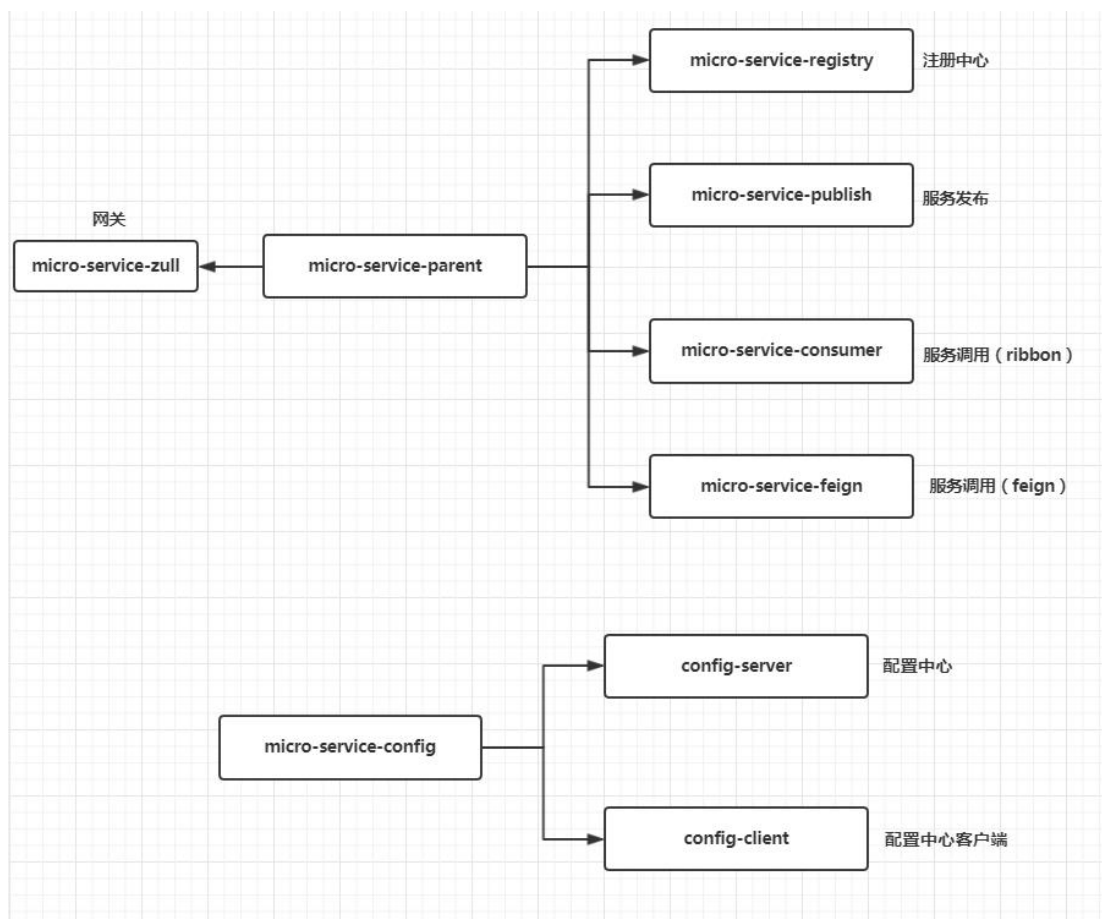
线程数	Dubbo	Spring Cloud
10线程	2.75	6.52
20线程	4.18	10.03
50线程	10.3	28.14
100线程	20.13	55.23
200线程	42	110.21

## 三. SpringCloud 实战

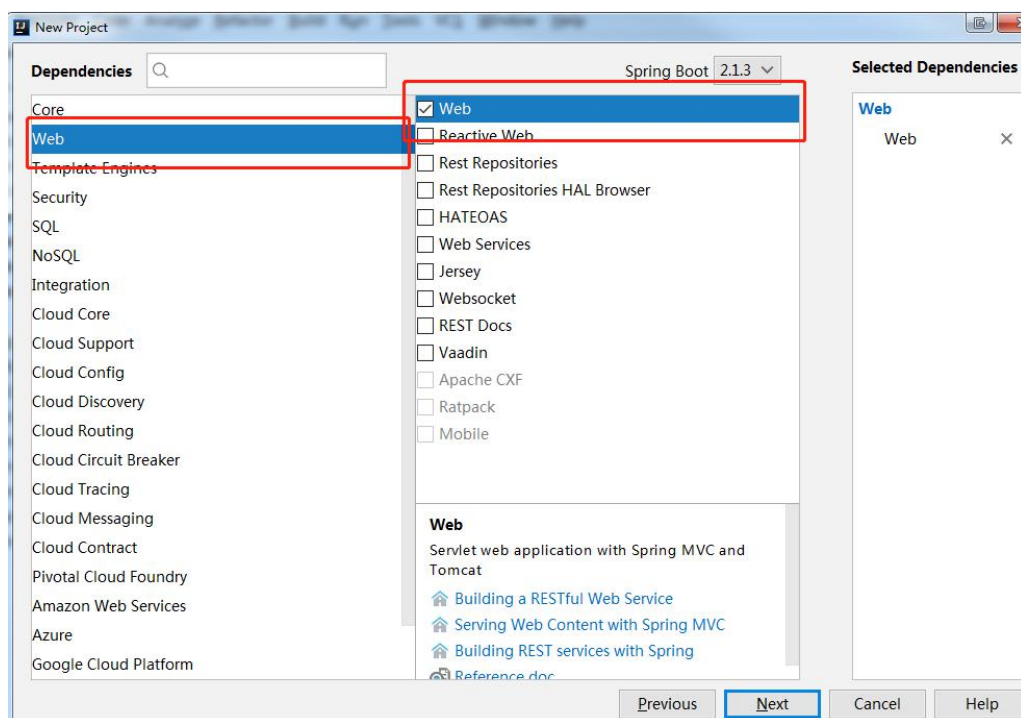
### 3.1 创建父工程

创建父工程: micro-service-parent, 通过父工程统一管理 SpringCloud 组件的版本号。

我使用 jdk 为 1.8 版本，**springBoot 为 2.0.6.RELEASE** 版本，**SpringCloud 为 Finchley.RELEASE** 版本。



使用 IDEA 创建 spring 实例工程, 如图:



讲师: 闫洪波

微信: 769828176

修改 Pom 文件如下:

```
<?xml version="1.0" encoding="UTF-8"?>

<project xmlns="http://maven.apache.org/POM/4.0.0"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/xsd/maven-4.0.0.xsd">

  <modelVersion>4.0.0</modelVersion>

  <parent>

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

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

    <version>2.0.6.RELEASE</version>

    <relativePath/> <!-- lookup parent from repository -->

  </parent>

  <groupId>com.spring.microservice</groupId>

  <artifactId>micro-service-parent</artifactId>

  <version>0.0.1-SNAPSHOT</version>

  <name>micro-service-parent</name>

  <description>Demo project for Spring Boot</description>

  <modules>

    <module>micro-service-registry</module>

    <module>micro-service-publish</module>

    <module>micro-service-consumer</module>

    <module>micro-service-feign</module>

    <module>micro-service-zull</module>

  </modules>

  <!-- 版本信息 -->
```

```
<properties>

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

    <spring-cloud.version>Finchley.RELEASE</spring-cloud.version>

</properties>

<dependencies>

    <dependency>

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

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

    </dependency>

    <dependency>

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

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

        <scope>test</scope>

    </dependency>

</dependencies>

<dependencyManagement>

    <dependencies>

        <dependency>

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

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

            <version>${spring-cloud.version}</version>

            <type>pom</type>

            <scope>import</scope>

        </dependency>

    </dependencies>

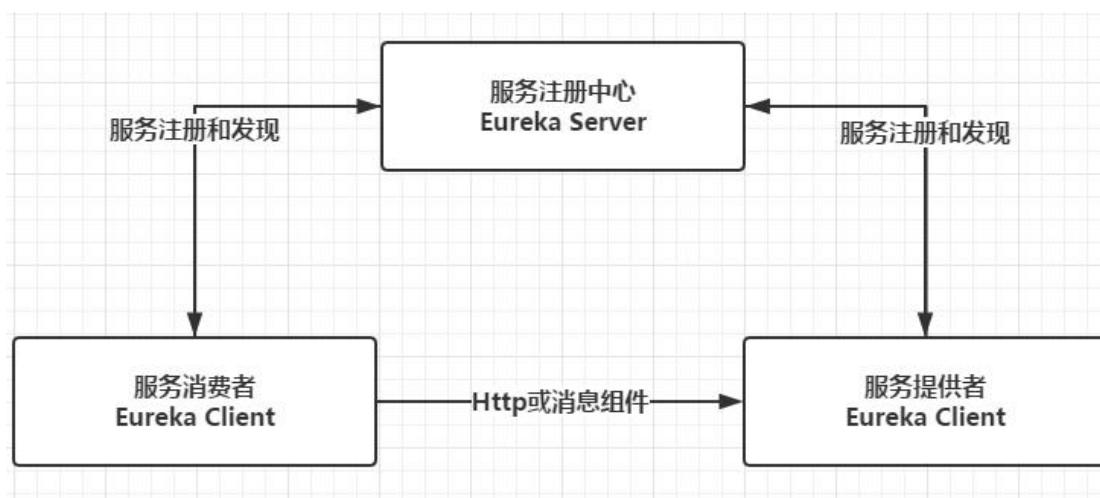
</dependencyManagement>
```

```
<build>
  <plugins>
    <plugin>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-maven-plugin</artifactId>
    </plugin>
  </plugins>
</build>
</project>
```

## 3.2 Eureka

### 3.2.1 Eureka 概述

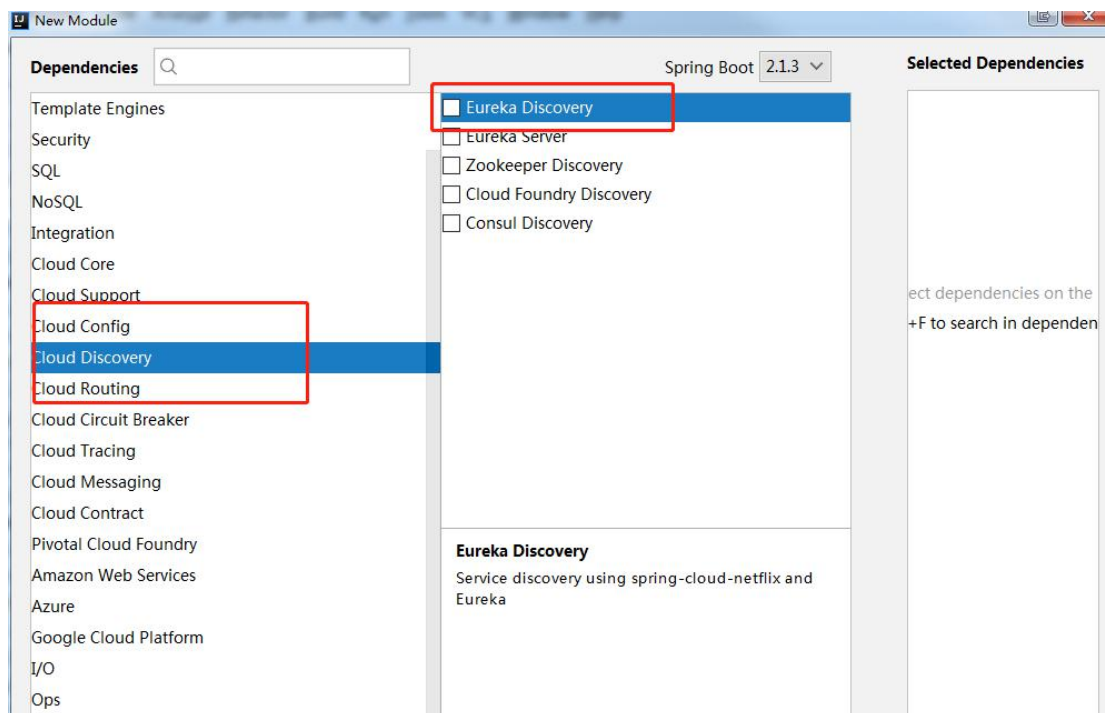
和 Zookeeper 类似，Eureka 是一个用于服务注册和发现的组件。Eureka 分为 Eureka Server 和 Eureka Client，Eureka Server 为 Eureka 服务注册中心，Eureka Client 为 Eureka 客户端。



服务注册是指向服务注册中心注册一个服务实例，服务提供者将自己的服务信息（如服务名、IP 地址等）告知服务注册中心。服务发现是指当服务消费者需要消费另外一个服务时，服务注册中心能够告知服务消费者它所消费服务的实例信息（如服务名、地址等）。通常情况下一个服务既是服务提供者，也是服务消费者。服务消费者一般使用 HTTP 协议或者消息组件这种轻量级的通信机制来进行服务消费。

### 3.2.2 注册中心 Eureka Server

创建 micro-service-parent 子工程 micro-service-registry，并依赖 eureka-server。



修改工程依赖 pom 文件：

```
<?xml version="1.0" encoding="UTF-8"?>

<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.spring.microservice</groupId>

  <artifactId>micro-service-parent</artifactId>

  <version>0.0.1-SNAPSHOT</version>

</parent>

<groupId>com.spring.microservice</groupId>

<artifactId>micro-service-registry</artifactId>

<version>0.0.1-SNAPSHOT</version>

<name>micro-service-registry</name>

<packaging>jar</packaging>

<description>Demo project for Spring Boot</description>

<dependencies>

  <dependency>

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

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

  </dependency>

  <dependency>

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

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

  </dependency>

  <dependency>

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

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

    <scope>test</scope>

  </dependency>

</dependencies>
```



```
</dependencies>

</project>
```

修改 application.properties, 添加 eureka 配置信息:

```
#注册中心端口

server.port=8081

#设置当前实例的主机名称

eureka.instance.hostname=localhost

#默认是启动服务注册的 此处关闭服务注册 表示不向注册中心注册自己

eureka.client.registerWithEureka=false

#由于注册中心的职责是维护服务实例, 不需要检索服务, 所以设置为 false

eureka.client.fetch-registry=false

#注册中心服务 URL

eureka.client.service-url.defaultZone=http://${eureka.instance.hostname}:${server.port}/eureka/

#注册中心服务名

spring.application.name=eureka-server
```

在启动类上添加注解@EnableEurekaServer 开启注册中心功能:

```
@EnableEurekaServer

@SpringBootApplication

public class MicroServiceRegistryApplication {

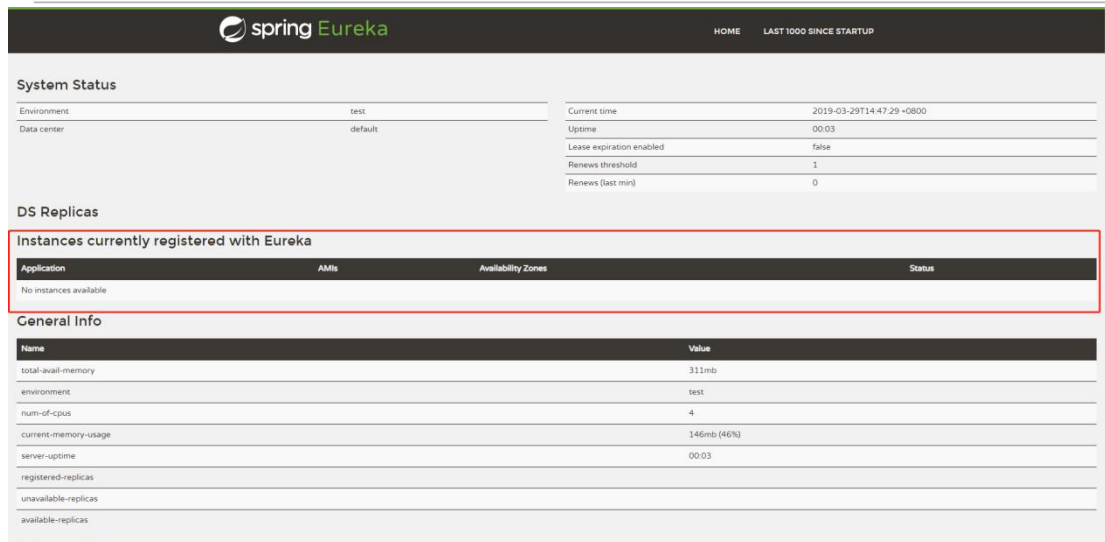
    public static void main(String[] args) {

        SpringApplication.run(MicroServiceRegistryApplication.class, args);

    }

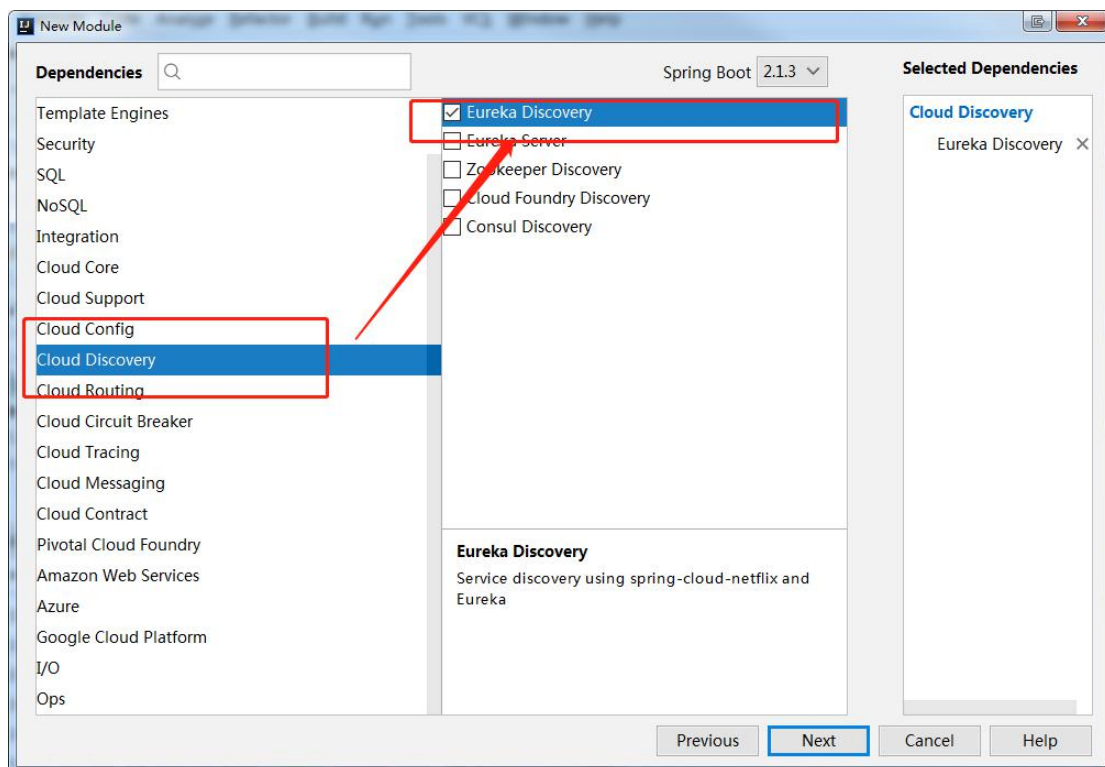
}
```

启动成功后访问: <http://localhost:8081/>, 显示如图:



### 3.1.3 服务发布 Eureka Clinet

创建 micro-service-parent 子工程 micro-service-publish, 并依赖 eureka-client。



修改 pom 文件依赖信息：

```
<?xml version="1.0" encoding="UTF-8"?>

<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.spring.microservice</groupId>

        <artifactId>micro-service-parent</artifactId>

        <version>0.0.1-SNAPSHOT</version>

    </parent>

    <groupId>com.spring.microservice</groupId>

    <artifactId>micro-service-publish</artifactId>

    <version>0.0.1-SNAPSHOT</version>

    <name>micro-service-publish</name>

    <description>Demo project for Spring Boot</description>

    <dependencies>

        <dependency>

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

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

        </dependency>

        <dependency>

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

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

        </dependency>

        <dependency>

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

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

            <scope>test</scope>

        </dependency>

    </dependencies>
```

```
</project>
```

修改 application.properties 文件，添加相关配置

```
#指定服务端口

server.port=8082

#指定服务名称

spring.application.name=service-hi

#指定服务注册中心地址

eureka.client.service-url.defaultZone=http://localhost:8081/eureka/
```

启动类添加@EnableEurekaClient，表明这是一个服务提供方

```
@SpringBootApplication
@EnableEurekaClient

public class MicroServicePublishApplication {

    public static void main(String[] args) {

        SpringApplication.run(MicroServicePublishApplication.class, args);

    }

}
```

创建 Controller，添加 hi 接口

```
@RestController

public class EurekaPublishController {

    @Value("${server.port}")

    private String port;

    @RequestMapping("/hi")

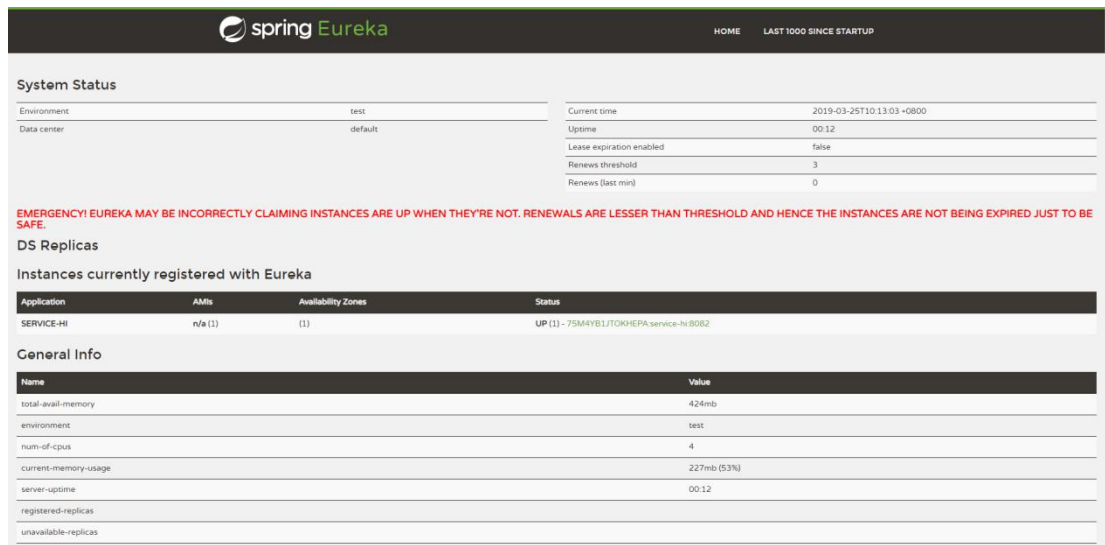
    public String publish(@RequestParam(value = "name") String name){

        return name+"-----port:"+port;

    }

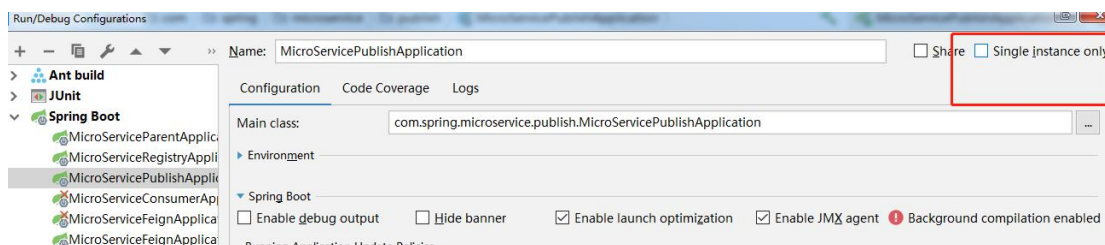
}
```

访问注册中心可以看到服务列表中新增 SERVICE-HI 服务信息：



### 3.1.4 Eureka Server 的高可用

修改 tomcat 启动配置为非单实例方式。



修改 application.properties 文件，启动一个实例

```
#注册中心端口

server.port=8181

#设置当前实例的主机名称

eureka.instance.hostname=server

#默认是启动服务注册的 此处关闭服务注册 表示不向注册中心注册自己

eureka.client.registerWithEureka=false

#由于注册中心的职责是维护服务实例，不需要检索服务，所以设置为 false
```

讲师：闫洪波

微信：769828176

```
eureka.client.fetch-registry=false

#注册中心服务 URL

eureka.client.service-url.defaultZone=http://localhost:8081/eureka/

#注册中心服务名

spring.application.name=eurka-server
```

修改 application.properties 文件，启动第二个实例

```
#注册中心端口

server.port=8081

#设置当前实例的主机名称

eureka.instance.hostname=server2

#默认是启动服务注册的 此处关闭服务注册 表示不向注册中心注册自己

eureka.client.registerWithEureka=false

#由于注册中心的职责是维护服务实例，不需要检索服务，所以设置为 false

eureka.client.fetch-registry=false

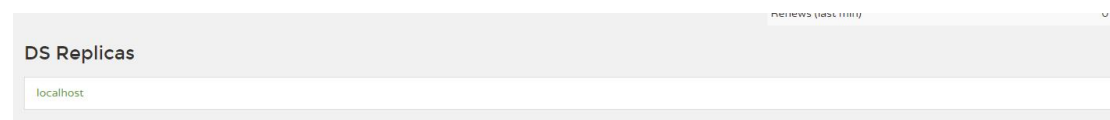
#注册中心服务 URL 此处写其他注册中心服务 URL

eureka.client.service-url.defaultZone=http://localhost:8181/eureka/

#注册中心服务名

spring.application.name=eurka-server
```

分别访问：<http://localhost:8181> 和 <http://localhost:8081>，可以看到：



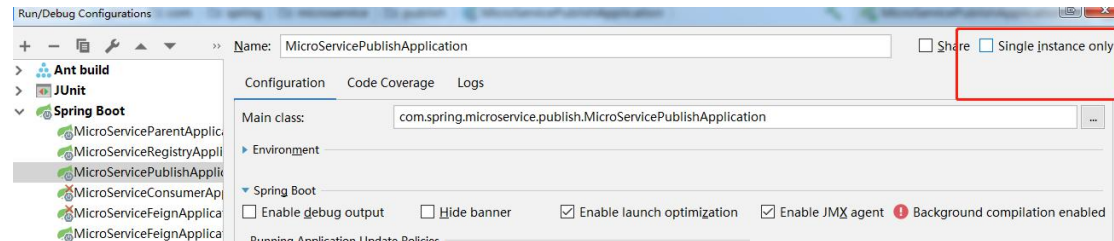
启动服务，继续访问我们可以看到虽然服务只注册了一个注册中心，但是两个注册中心都可以查看到服务列表中已存在该服务。

#### Instances currently registered with Eureka

Application	AMIs	Availability Zones	Status
SERVICE-HI	n/a (1)	(1)	UP (1) - 75M4YB1JTOKHEPA:service-hi:8089

### 3.1.5 Eureka Client 的高可用

修改 tomcat 启动配置为非单实例方式。



修改 application.properties 文件，端口为 8089：

```
#指定服务端点
server.port=8089

#指定服务名称
spring.application.name=service-hi

#指定服务注册中心地址
eureka.client.service-url.defaultZone=http://localhost:8081/eureka/
```

再次启动服务，查看服务注册中心，可以看到同一个服务有两个实例了。

SERVICE-HI	n/a (2)	(2)	UP (2)	75M4YB1JTOKHEPA:service-hi:8089	75M4YB1JTOKHEPA:service-hi:8082
------------	---------	-----	--------	---------------------------------	---------------------------------

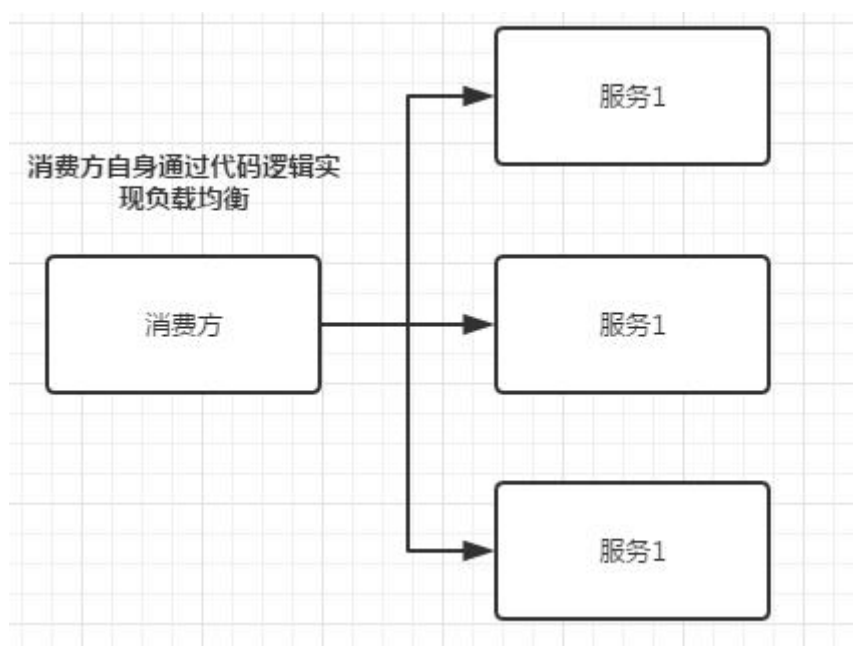
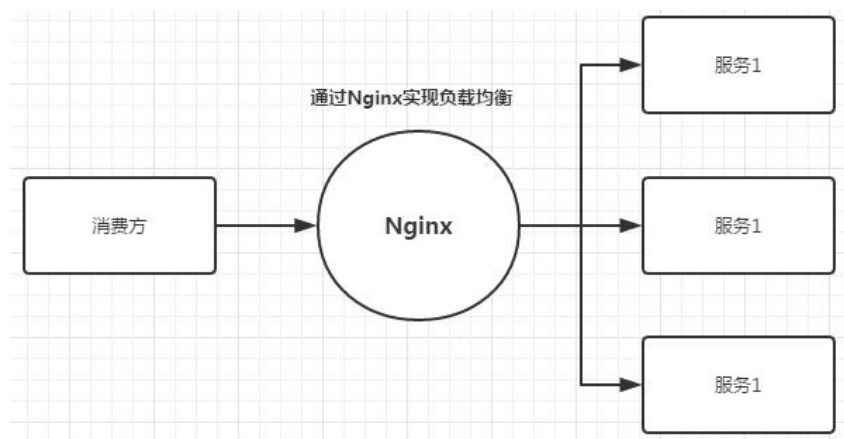
## 3.3 Ribbon

### 3.2.1 Ribbo 概述

负载均衡是指将负载分摊到多个执行单元上，常见的负载均衡有两种方式。一种是独立进程单元，通过负载均衡策略，将请求转发到不同的执行单元上，例如 Ngnix。另一种是



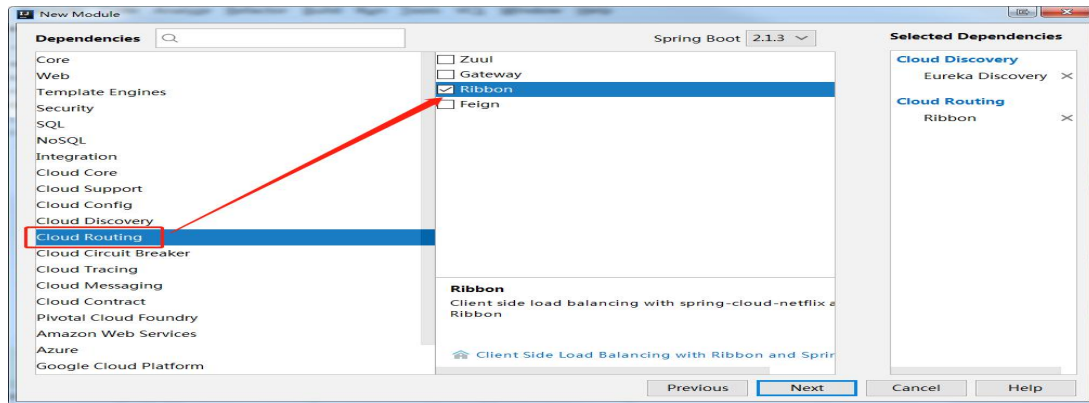
将负载均衡逻辑以代码的形式封装到服务消费者的客户端上,服务消费者客户端维护了一份服务提供的信息列表,有了信息列表,通过负载均衡策略将请求分摊给多个服务提供者,从而达到负载均衡的目的。



Ribbon 是 Netflix 公司开源的一个负载均衡的组件,它属于上述的第二种方式,是将负载均衡逻辑封装在客户端中,并且运行在客户端的进程里。Ribbon 作为服务消费者的负载均衡器,有两种使用方式一种是和 RestTemplate 相结合,另一种是和 Feign 相结合。Feign 已经默认集成了 Ribbon。

### 3.2.2 RestTemplate 消费案例

创建 micro-service-parent 子工程 micro-service-consumer，并依赖 eureka-client 和 ribbon。



```
<?xml version="1.0" encoding="UTF-8"?>

<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.spring.microservice</groupId>

    <artifactId>micro-service-parent</artifactId>

    <version>0.0.1-SNAPSHOT</version>

  </parent>

  <groupId>com.spring.microservice</groupId>

  <artifactId>micro-service-consumer</artifactId>

  <version>0.0.1-SNAPSHOT</version>

  <name>micro-service-consumer</name>

  <description>Demo project for Spring Boot</description>
```

```
<dependencies>

  <dependency>

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

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

  </dependency>

  <dependency>

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

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

  </dependency>

  <dependency>

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

    <artifactId>spring-cloud-starter-netflix-ribbon</artifactId>

  </dependency>

  <dependency>

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

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

    <scope>test</scope>

  </dependency>

</dependencies>

<build>

  <plugins>

    <plugin>

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

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

    </plugin>

  </plugins>

</build>

</project>
```

修改 application.properties 配置文件，指定相关信息：

```
#指定 ribbon 服务名称
spring.application.name=eureka-ribbon-client

#设定当前服务端口
server.port=8083

#指定服务注册中心地址
eureka.client.service-url.defaultZone=http://localhost:8081/eureka
```

添加@EnableEurekaClient 注解，标识为 Eureka，用于服务调用。

```
@SpringBootApplication
@EnableEurekaClient

public class MicroServiceConsumerApplication {

    public static void main(String[] args) {

        SpringApplication.run(MicroServiceConsumerApplication.class, args);

    }

}
```

创建配置类，获取 RestTemplate 对象 Bean，LoadBalanced 注解使 restTemplate 在 http 请求时候具备了负载均衡的能力。

```
package com.spring.micro.service.consumer.config;

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;

/**
```

```
* @Author: zhangjian
* @Date: 2019/3/25 10:56
* @Version 1.0
*/

@Configuration
public class RibbonConfig {

    @Bean

    @LoadBalanced

    RestTemplate restTemplate() {

        return new RestTemplate();

    }

}
```

创建 service 类, 通过 RestTemplate 调用服务:

```
package com.spring.micro.service.consumer.service;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;
import org.springframework.web.client.RestTemplate;

/**
 * @Author: zhangjian
 * @Date: 2019/3/25 10:57
 * @Version 1.0
 */
@Service
public class RibbonService {

    @Autowired

    RestTemplate restTemplate;
```

```
public String hi(String name){  
  
    return  
  
restTemplate.getForObject("http://service-hi/hi?name="+name,String.class);  
  
}  
  
}
```

### 创建 Controller 接口，调用服务

```
package com.spring.micro.service.consumer.controller;  
  
import com.spring.micro.service.consumer.service.RibbonService;  
  
import org.springframework.beans.factory.annotation.Autowired;  
  
import org.springframework.web.bind.annotation.RequestMapping;  
  
import org.springframework.web.bind.annotation.RequestParam;  
  
import org.springframework.web.bind.annotation.RestController;  
  
/**  
 * @Author: zhangjian  
 * @Date: 2019/3/25 10:59  
 * @Version 1.0  
 */  
  
@RestController  
  
public class RibbonController {  
  
    @Autowired  
  
    RibbonService ribbonService;  
  
    @RequestMapping("/hi")  
  
    public String hi(@RequestParam(required = false,defaultValue = "") String  
name) {  
  
        return ribbonService.hi(name);  
  
    }  
  
}
```

启动服务，访问 <http://localhost:8083/hi>



查看注册中心，可以看到服务消费方也是服务注册方：

Application	AMIs	Availability Zones	Status
EUREKA-RIBBON-CLIENT	n/a (1)	(1)	UP (1) - 75M4YB1JTOKHEPA:eureka-ribbon-client:8083
SERVICE-HI	n/a (1)	(1)	UP (1) - 75M4YB1JTOKHEPA:service-hi:8082

### 3.2.3 负载均衡案例

创建高可用 Eureka 的服务实例，并启动。目前 eureka 服务有两个集群构成，那么我们继续访问 <http://localhost:8083/hi>，会发现 port 在 8082 和 8089 之间不断的切换，那么我们也发现 ribbon 的负载均衡策略默认是轮流访问的方式进行的。



## 3.4 Feign

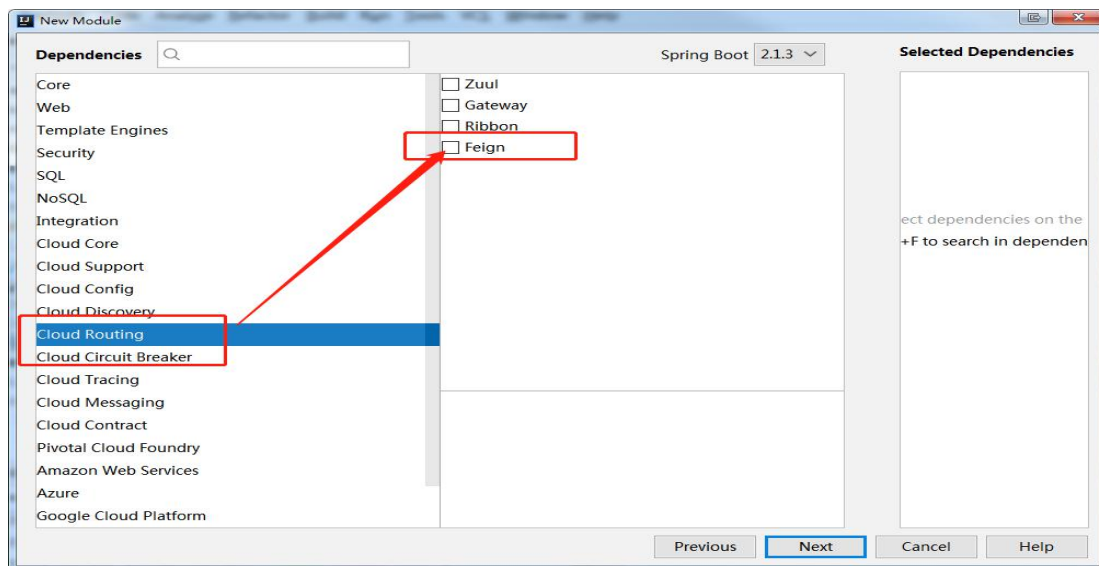
### 3.3.1 Feign 概述

Feign 受 Retrofit、JAXRS-2.0 和 WebSocket 影响，采用了声明式的 API 接口风格，将 Java Http 客户端绑定到它的内部。Feign 的首要目的是将 Java Http 客户端调用过程变得简单。



### 3.3.2 Feign 案例

创建 micro-service-parent 子工程 micro-service-feign，并依赖 eureka-client 和 openfeign。



修改 pom 文件相关信息：

```
<?xml version="1.0" encoding="UTF-8"?>

<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.spring.microservice</groupId>

    <artifactId>micro-service-parent</artifactId>

    <version>0.0.1-SNAPSHOT</version>

  </parent>

  <groupId>com.spring.microservice</groupId>

  <artifactId>micro-service-feign</artifactId>
```

```
<version>0.0.1-SNAPSHOT</version>

<name>micro-service-feign</name>

<description>Demo project for Spring Boot</description>

<dependencies>

  <dependency>

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

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

  </dependency>

  <dependency>

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

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

  </dependency>

  <dependency>

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

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

  </dependency>

  <dependency>

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

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

    <scope>test</scope>

  </dependency>

</dependencies>

<build>

  <plugins>

    <plugin>

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

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

    </plugin>

  </plugins>

</build>
```

```
</plugins>

</build>

</project>
```

修改 application.properties 文件:

```
#指定服务名称

spring.application.name=eureka-feign-client

#指定服务端口

server.port=8084

#指定注册中心 url

eureka.client.service-url.defaultZone=http://localhost:8081/eureka/
```

添加注解@EnableEurekaClient 和@EnableFeignClients 在启动类:

```
package com.spring.microservice.feign;

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

@SpringBootApplication
@EnableEurekaClient
@EnableFeignClients

public class MicroServiceFeignApplication {

    public static void main(String[] args) {
```

```
        SpringApplication.run(MicroServiceFeignApplication.class, args);  
    }  
  
}
```

创建 Feign 声明式调用接口,使用@FeignClient 声明调用的服务名称,使用 requestMapping 声明调用的具体服务。

```
package com.spring.microservice.feign.client;  
  
import com.spring.microservice.feign.config.FeignConfig;  
import org.springframework.cloud.openfeign.FeignClient;  
import org.springframework.web.bind.annotation.RequestMapping;  
import org.springframework.web.bind.annotation.RequestMethod;  
import org.springframework.web.bind.annotation.RequestParam;  
  
/**  
 * @Author: zhangjian  
 * @Date: 2019/3/25 14:31  
 * @Version 1.0  
 */  
  
@FeignClient(value = "service-hi", configuration = FeignConfig.class)  
public interface EurekaClientFeign {  
  
    @RequestMapping(value = "hi", method = RequestMethod.GET)  
    String sayHi(@RequestParam(value = "name") String name);  
}
```

```
}
```

创建 Controller 使用 Feign 接口调用服务。

```
package com.spring.microservice.feign.controller;

import com.spring.microservice.feign.client.EurekaClientFeign;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.RequestParam;
import org.springframework.web.bind.annotation.RestController;

/**
 * @Author: zhangjian
 * @Date: 2019/3/25 14:39
 * @Version 1.0
 */
@RestController
public class FeignController {

    @Autowired
    EurekaClientFeign eurekaClientFeign;

    @GetMapping("/hi")
    public String sayHi(@RequestParam(defaultValue = "", required = false) String
name) {

        return eurekaClientFeign.sayHi(name);

    }

}
```

使用@Configuration 可以指定 FeignClient 的重试次数, 重试间隔为 100ms, 最大重试时间为 1s, 重试次数为 5 次

```
package com.spring.microservice.feign.config;

import feign.Retryer;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import static java.util.concurrent.TimeUnit.SECONDS;

/**
 * @Author: zhangjian
 * @Date: 2019/3/25 14:36
 * @Version 1.0
 */
@Configuration
public class FeignConfig {

    @Bean
    public Retryer feignRetryer() {
        return new Retryer.Default(100, SECONDS.toMillis(1), 3);
    }
}
```

### 3.3.3 负载均衡案例

创建高可用 Eureka 的服务实例，并启动。目前 eureka 服务有两个集群构成，那么我们继续访问 <http://localhost:8084/hi>，会发现 port 在 8082 和 8089 之间不断的切换，那么我们也发现 ribbon 的负载均衡策略默认是轮流访问的方式进行的。



## 3.5 Hystrix

### 3.4.1 Hystrix 概述

在分布式系统中，服务与服务之间的依赖错综复杂，一种不可避免的情况就是某些服务出现故障，导致依赖于它们的其他服务出现远程调度的线程阻塞 Hystrix 是 Netflix 公司开源的一个项目，它提供了熔断器功能，能够阻止分布式系统中出现联动故障。Hystrix 是通过隔离服务的访问点阻止联动故障的，并提供了故障的解决方案，从而提升了整个分布式系统的弹性。

### 3.4.1 Hystrix 案例

修改 micro-service-feign 工程的 application.properties 文件：开启 hystrix

```
feign.hystrix.enabled=true
```

修改启动类，添加注解：EnableDiscoveryClient

```
@SpringBootApplication
@EnableEurekaClient
```



```
@EnableFeignClients

@EnableDiscoveryClient

public class MicroServiceFeignApplication {

    public static void main(String[] args) {

        SpringApplication.run(MicroServiceFeignApplication.class, args);

    }

}
```

创建 EurekaClientFeign 的实现类：EurekaClientHystric

```
@Component

public class EurekaClientHystric implements EurekaClientFeign {

    @Override

    public String sayHi(String name) {

        return "sorry:"+name;

    }

}
```

在@FeignClient 注解中添加 fallback=EurekaClientHystric.class,指定当所调用服务故障时候, 调用 Hystric。

```
@FeignClient(value = "service-hi", configuration = FeignConfig.class, fallback =
EurekaClientHystric.class)

public interface EurekaClientFeign {

    @RequestMapping(value="hi", method = RequestMethod.GET)

    String sayHi(@RequestParam(value="name")String name);

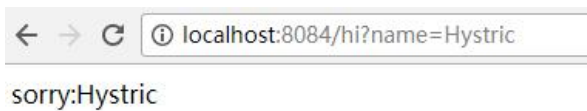
}
```

```
}
```

此时请求 `http://localhost:8084/hi?Name=Hystric`, 可以看到服务被调用:



关闭 eureka 服务, 此时再次请求 `http://localhost:8084/hi?Name=Hystric`, 可以看到返回熔断器接口实现:



## 3.6 Zuul

### 3.6.1 Zuul 概述

**Zuul** 作为微服务系统的网关组件, 用于构建边界服务 ( `dgService` ), 致力于动态路由、过滤、监控、弹性伸缩和安全。

### 3.6.2 Zuul 案例

创建 `micro-service-parent` 子工程 `micro-service-zuul`, 并依赖 `eureka-client` 和 `openfeign`。

`pom` 文件依赖如下:

```
<?xml version="1.0" encoding="UTF-8"?>
<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.spring.microservice</groupId>

        <artifactId>micro-service-parent</artifactId>

        <version>0.0.1-SNAPSHOT</version>

    </parent>

    <groupId>com.spring.microservice</groupId>

    <artifactId>micro-service-zull</artifactId>

    <version>0.0.1-SNAPSHOT</version>

    <name>micro-service-zull</name>

    <description>Demo project for Spring Boot</description>

    <dependencies>

        <dependency>

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

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

        </dependency>

        <dependency>

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

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

        </dependency>

        <dependency>

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

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

        </dependency>
```

```
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-test</artifactId>
    <scope>test</scope>
</dependency>
</dependencies>
</project>
```

修改 application.properties 文件

```
#指定服务端口

server.port=8085

#指定服务名称

spring.application.name=service-zuul

#指定注册中心地址

eureka.client.service-url.defaultZone=http://localhost:8081/eureka/

#指定 api-a 请求调用的服务

zuul.routes.api-a.path=/api-a/**

zuul.routes.api-a.service-id=eureka-ribbon-client

#指定 api-b 请求调用的服务

zuul.routes.api-b.path=/api-b/**

zuul.routes.api-b.service-id=eureka-feign-client
```

修改启动类添加相关注解：

```
EnableZuulProxy、EnableEurekaClient 和 EnableDiscoveryClient

@SpringBootApplication
@EnableZuulProxy
@EnableEurekaClient
@EnableDiscoveryClient

public class MicroServiceZullApplication {

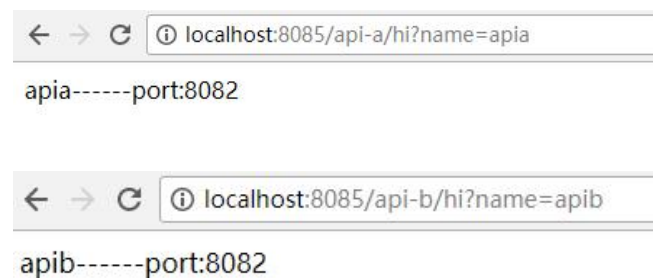
    public static void main(String[] args) {

        SpringApplication.run(MicroServiceZullApplication.class, args);

    }

}
```

分别请求/api-a/hi?name=apia 和/api-b/hi?name=apib, 可以看到被 zuul 分发到不同的服务。



← → ↻ ⓘ localhost:8085/api-a/hi?name=apia  
apia-----port:8082

← → ↻ ⓘ localhost:8085/api-b/hi?name=apib  
apib-----port:8082

## 3.7 Config

### 3.7.1 Config 概述

Spring Cloud Config 是 springCloud 组件中的分布式配置中心。该组件分为服务端和客户端, 服务端可以读取管理远程仓库中的配置文件, 客户端可以获取到服务端的配置信息。

创建一个 micro-service-config 父工程项目, 指定依赖的版本, springBoot 为 2.0.6,  
讲师: 闫洪波 微信: 769828176

spring cloud 为 Finchley.RELEASE 版本。

Pom 文件如下：

```
<?xml version="1.0" encoding="UTF-8"?>

<project xmlns="http://maven.apache.org/POM/4.0.0"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
    http://maven.apache.org/xsd/maven-4.0.0.xsd">

  <modelVersion>4.0.0</modelVersion>

  <parent>

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

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

    <version>2.0.6.RELEASE</version>

    <relativePath/> <!-- lookup parent from repository -->

  </parent>

  <groupId>com.microservice</groupId>

  <artifactId>micro-service-config</artifactId>

  <version>0.0.1-SNAPSHOT</version>

  <name>micro-service-config</name>

  <description>Demo project for Spring Boot</description>

  <packaging>pom</packaging>

  <modules>

    <module>config-server</module>

    <module>config-client</module>

  </modules>

  <properties>

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

```
<spring-cloud.version>Finchley.RELEASE</spring-cloud.version>

</properties>

<dependencies>

  <dependency>

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

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

  </dependency>

  <dependency>

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

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

    <scope>test</scope>

  </dependency>

</dependencies>

<dependencyManagement>

  <dependencies>

    <dependency>

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

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

      <version>${spring-cloud.version}</version>

      <type>pom</type>

      <scope>import</scope>

    </dependency>

  </dependencies>

</dependencyManagement>

<build>
```

```
<plugins>

  <plugin>

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

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

  </plugin>

</plugins>

</build>

</project>
```

### 3.7.2 Config-Server 案例

创建 config-server 子工程，负责加载远程仓库的配置文件信息。需要依赖 spring-cloud-config-server。Pom 文件依赖信息如下：

```
<?xml version="1.0" encoding="UTF-8"?>

<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.microservice</groupId>

    <artifactId>micro-service-config</artifactId>

    <version>0.0.1-SNAPSHOT</version>

  </parent>

  <groupId>com.microservice</groupId>
```



```
<artifactId>config-server</artifactId>

<version>0.0.1-SNAPSHOT</version>

<name>config-server</name>

<description>Demo project for Spring Boot</description>

<dependencies>

  <dependency>

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

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

  </dependency>

  <dependency>

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

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

    <scope>test</scope>

  </dependency>

</dependencies>

<build>

  <plugins>

    <plugin>

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

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

    </plugin>

  </plugins>

</build>

</project>
```

修改 application.properties 文件,指定远程仓库信息:

```
spring.application.name=config-server

server.port=8086

#git 仓库地址

spring.cloud.config.server.git.uri=https://github.com/soulstare89/SpringcloudConfig

#仓库路径

spring.cloud.config.server.git.searchPaths=resp

#配置仓库的分支

spring.cloud.config.label=master

#访问 git 仓库的用户名

spring.cloud.config.server.git.username=

#访问 git 仓库的密码

spring.cloud.config.server.git.password=

#指定服务注册地址

eureka.client.serviceUrl.defaultZone=http://localhost:8081/eureka/

#开发环境 dev

spring.profiles.active=dev
```

修改启动类,添加注解@EnableConfigServer:

```
package com.micro.service.config;

import org.springframework.boot.SpringApplication;

import org.springframework.boot.autoconfigure.SpringBootApplication;

import org.springframework.cloud.config.server.EnableConfigServer;
```

```
@SpringBootApplication
@EnableConfigServer

public class ConfigServerApplication {

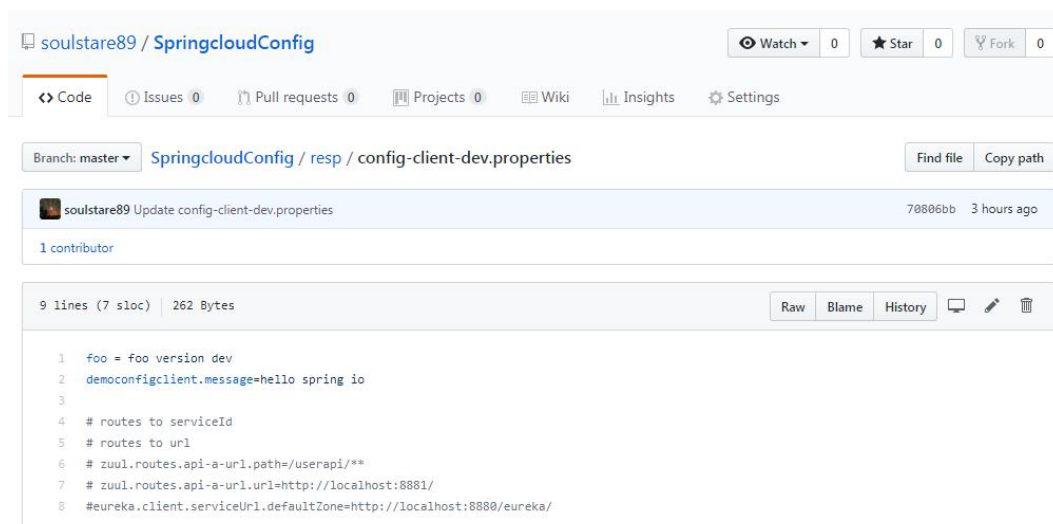
    public static void main(String[] args) {

        SpringApplication.run(ConfigServerApplication.class, args);

    }

}
```

在 gitHub 上创建自己的配置文件仓库中心，如图：



### 3.7.3 Config-Client 案例

创建 config-client 项目，依赖 spring-cloud-starter-config。该项目负责获取配置中心服务端配置信息。也可以再其他项目中添加依赖来获取配置信息进行调用。

```
<?xml version="1.0" encoding="UTF-8"?>

<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.microservice</groupId>

    <artifactId>micro-service-config</artifactId>

    <version>0.0.1-SNAPSHOT</version>

  </parent>

  <groupId>com.microservice</groupId>

  <artifactId>config-client</artifactId>

  <version>0.0.1-SNAPSHOT</version>

  <name>config-client</name>

  <description>Demo project for Spring Boot</description>

  <dependencies>

    <dependency>

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

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

    </dependency>

    <dependency>

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

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

```

        <scope>test</scope>

    </dependency>

    <dependency>

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

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

    </dependency>

</dependencies>

<build>

    <plugins>

        <plugin>

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

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

        </plugin>

    </plugins>

</build>

</project>

```

修改 bootstrap.properties 配置文件：

```

#指明配置文件
spring.application.name=config-client

#指明远程仓库的分支
spring.cloud.config.label=master

#开发环境配置文件
spring.cloud.config.profile=dev

#指明配置服务中心的网址。
spring.cloud.config.uri= http://localhost:8086/

server.port=8087

```

```
#eureka.client.serviceUrl.defaultZone=http://localhost:8081/eureka/  
  
#是从配置中心读取文件。  
  
#spring.cloud.config.discovery.enabled=true  
  
#配置中心的 servieId, 即服务名。  
  
#spring.cloud.config.discovery.serviceId=config-server
```

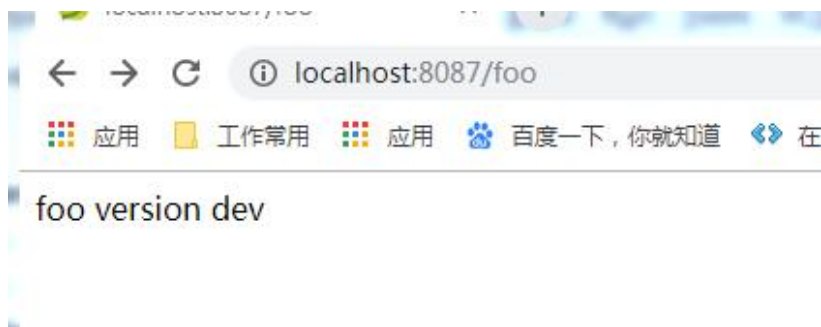
修改启动类, 添加注解: @EnableDiscoveryClient, @RefreshScope

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

编写控制层代码测试:

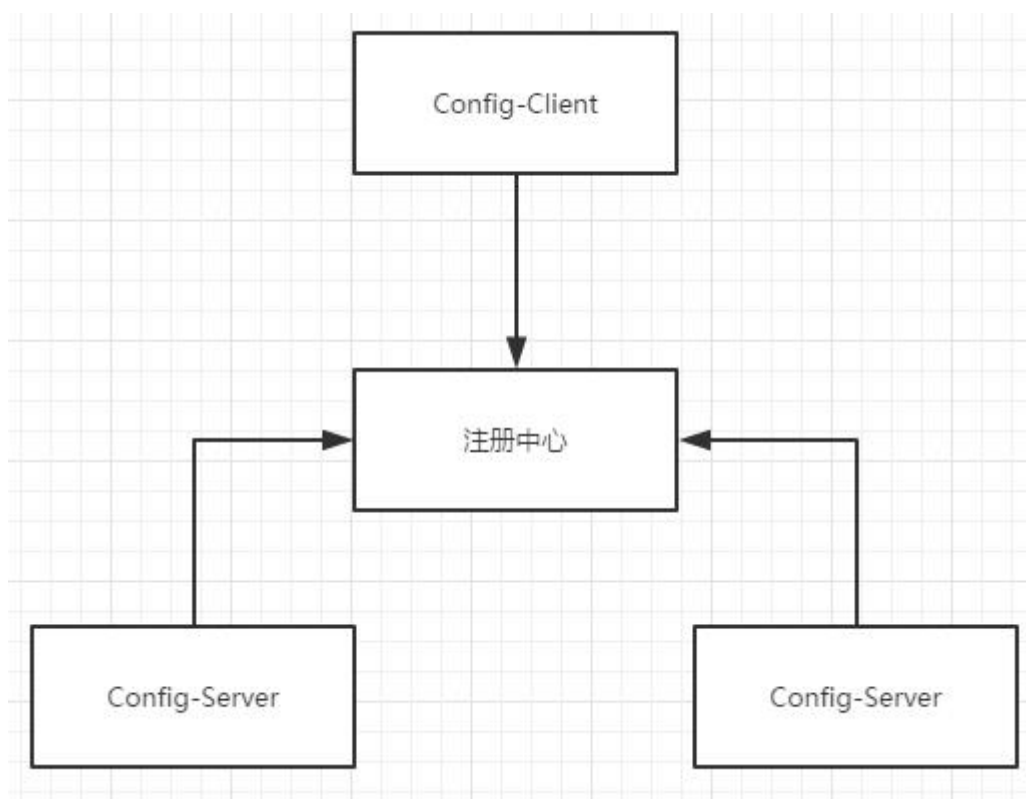
```
@RestController  
  
public class ConfigClientController {  
  
    @Value("${foo}")  
  
    String foo;  
  
    @RequestMapping(value="/foo")  
  
    public String hi() {  
  
        return foo;  
  
    }  
  
}
```

访问：<http://localhost:8087/foo>，可以看到获取到远程仓库中的配置信息：



### 3.7.4 Config 高可用案例

将配置中心 Config Server 做成一个微服务，并且将其集群化，从而达到高可用。配置中心 Config Server 高可用的架构图如图。ConfigServer 和 Config Client 向 Eureka Server 注册，且将 Config Server 多实例集群部署。



修改 config-server 的 pom 依赖信息，添加 eureka-client 依赖：

```
<dependency>

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

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

</dependency>
```

修改 application.properties 文件:

```
#指定服务注册地址

eureka.client.serviceUrl.defaultZone=http://localhost:8081/eureka/
```

修改启动类注解, 添加注解@EnableEurekaClient

```
@EnableEurekaClient

@SpringBootApplication

@EnableConfigServer

public class ConfigServerApplication
```

分别启动注册中心以及两个 config-server 实例:

CONFIG-SERVER	n/a (2)	(2)	UP (2) - 75M4YB1JTOKHEPA:config-server:8088 , 75M4YB1JTOKHEPA:config-server:8086
---------------	---------	-----	--

修改 config-client 的依赖信息, 添加 eureka-client:

```
<dependency>

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

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

</dependency>
```

修改 bootstrap.properties 文件, 指定从注册中心获取配置信息:

```
#指明配置文件

spring.application.name=config-client

#指明远程仓库的分支
```



```
spring.cloud.config.label=master

#开发环境配置文件

spring.cloud.config.profile=dev

#指明配置服务中心的网址。

#spring.cloud.config.uri= http://localhost:8088/

server.port=8087

#指定注册中心地址

eureka.client.serviceUrl.defaultZone=http://localhost:8081/eureka/

#是从配置中心读取文件。

spring.cloud.config.discovery.enabled=true

#配置中心的 servieId, 即服务名。

spring.cloud.config.discovery.serviceId=CONFIG-SERVER
```

启动 config-client, 就实现了配置中心的高可用。

## 四. 学习资料

中文官网: <https://springcloud.cc/>

参考书籍: 《深入理解 Spring Cloud 与微服务构建》