1. **什么是微服务**

> The **microservice architectural** style is an approach to developing a single application as **a suite of small services, each running in its own process and communicating with lightweight mechanisms**, often an HTTP resource API. These services are ****built around business capabilities**** and **independently deployable** by fully automated deployment machinery. There is **a bare minimum of centralized management of these services** , which may be written in different programming languages and use different data storage technologies. **-- James Lewis and Martin Fowler**

官方网站: https://www.martinfowler.com/articles/microservices.html

---

2. **微服务的起源**

---

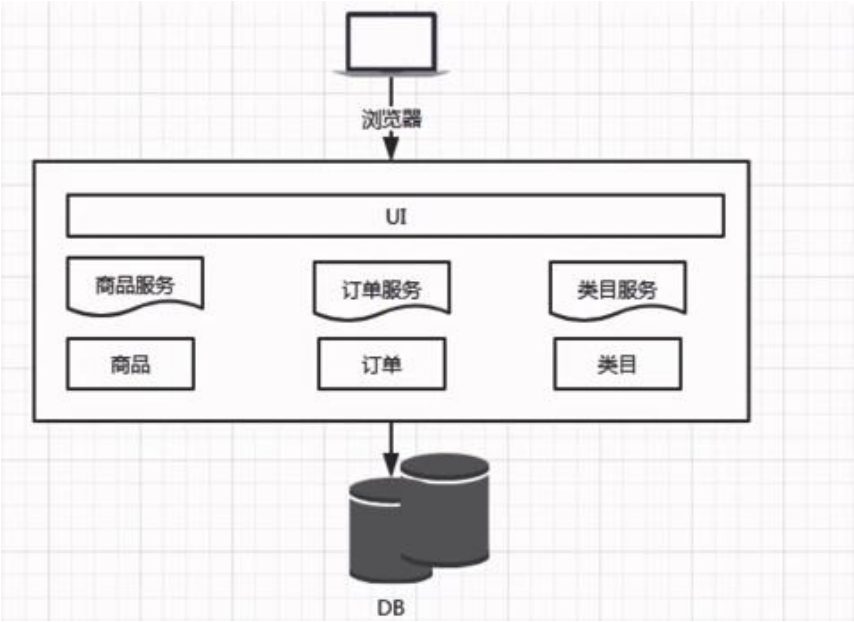
3. **微服务官方定义的解释**

> 微服务就是由一系列服务功能组成,能单独跑在自己的进程里,每个服务独立开发,独立部署,分布式的管理

---

4. **为什么会出现微服务**

5. **单体架构**



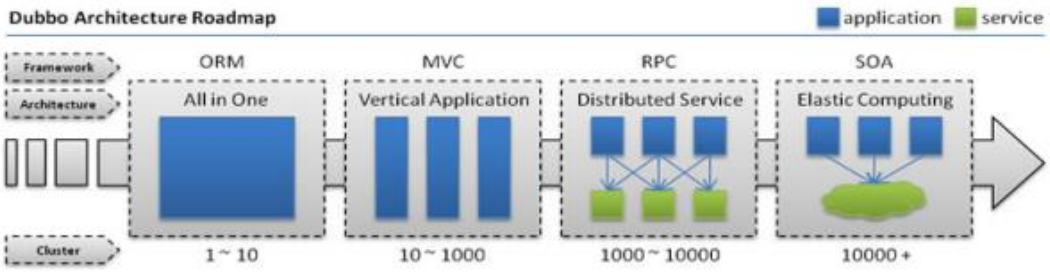
优点:

- 容易测试
- 容易部署

缺点:

- 开发效率低
- 代码难维护
- 部署不灵活
- 稳定性不高
- 扩展性不高

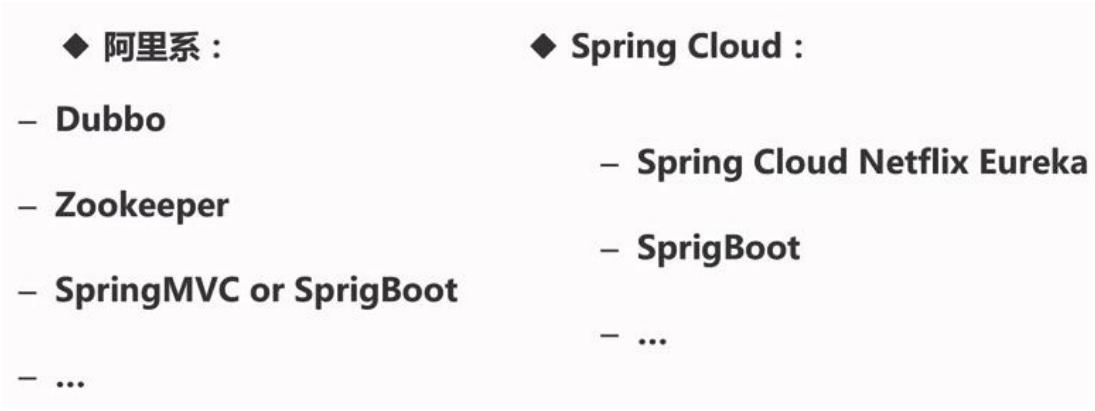
2. 架构的演变



好的架构不是设计出来的,是(**进化演变 ing**)出来的

---

5. 微服务的解决方案



---

6. 什么是 Spring Cloud

Spring Cloud 是一个含概多个子项目的开发工具集,集合了众多的开源框架,他利用了 Spring Boot 开发的便利性实现了很多功能,如服务注册,服务注册发现,负载均衡等.Spring Cloud 在整合过程中主要是针对 Netflix(耐非)开源组件的封装.

NetFlix 是美国的一个在线视频网站,微服务业的翘楚,他是公认的大规模生产级微服务的杰出实践者,NetFlix 的开源组件已经在他大规模分布式微服务环境中经过多年的生产实战验证,因此 spring cloud 中很多组件都是基于 NetFlix 组件的封装

Spring Cloud 的出现真正的简化了分布式架构的开发

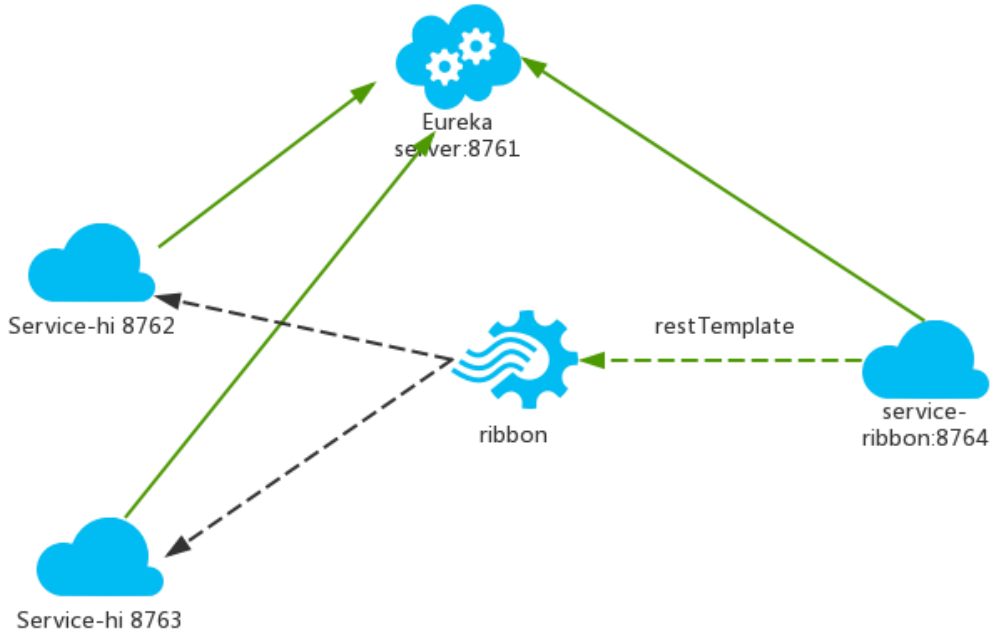
---

7. Spring Cloud 的特性

a. 服务注册和发现
b. 路由
c. service - to - service 调用
d. 负载均衡
e. 断路器

---

8. Spring Cloud 的服务架构图



- 一个服务注册中心，eureka server,端口为 8080
- 多个服务的提供者 端口为 8989.....
- 一个服务的消费者 端口为 9090

---

9. 开发注册中心

1. 引入 spring cloud 的依赖

```
<?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>


    <groupId>com.baizhi</groupId>
    <artifactId>springcloud_regist</artifactId>
```

```
<version>1.0-SNAPSHOT</version>
<name>springcloud_regist</name>
<url>http://www.example.com</url>

<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>
</properties>

<!--引入 springboot 的父项目-->
<parent>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-parent</artifactId>
    <version>1.5.7.RELEASE</version>
</parent>
<dependencies>
    <!--eureka server-->
    <dependency>
        <groupId>org.springframework.cloud</groupId>
        <artifactId>spring-cloud-starter-eureka-server</artifactId>
    </dependency>
    <!-- spring boot test-->
    <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>Dalston.RC1</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>
<repositories>
    <repository>
        <id>spring-milestones</id>
        <name>Spring Milestones</name>
        <url>https://repo.spring.io/milestone</url>
        <snapshots>
            <enabled>false</enabled>
        </snapshots>
    </repository>
</repositories>
```

```
</project>
```

2. 创建服务的入口类

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

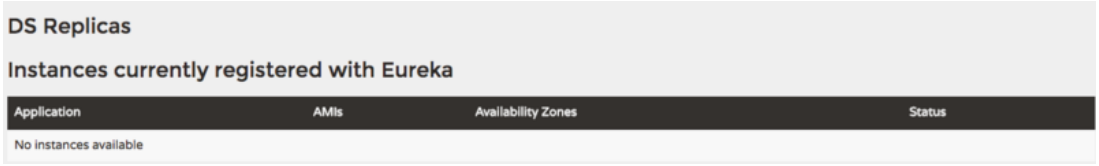
3. 编辑 springboot 的配置文件

```
server:
  port: 8080
eureka:
  instance:
    hostname: localhost
  client:
    register-with-eureka: false
    fetch-registry: false
    service-url:
      defaultZone:
http://${eureka.instance.hostname}:${server.port}/eureka/
```

注意:通过 eureka.client.registerWithEureka：false 和 fetchRegistry：false 来表明自己是一个 eureka server.

4. 启动 Eureka 服务,打开浏览器访问 Eureka 浏览器访问:

a. 访问: http://localhost:8080



---

10. 开发 Eureka 的客户端(服务提供)

1. 引入 spring cloud 的依赖

```
<?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>

    <groupId>com.baizhi</groupId>
    <artifactId>springcloud_regist</artifactId>
    <version>1.0-SNAPSHOT</version>
    <name>springcloud_regist</name>
    <url>http://www.example.com</url>
```

```
    <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>
    </properties>

    <!--引入 springboot 的父项目-->
    <parent>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-parent</artifactId>
        <version>1.5.7.RELEASE</version>
    </parent>
    <dependencies>
        <!--eureka server-->
        <dependency>
            <groupId>org.springframework.cloud</groupId>
            <artifactId>spring-cloud-starter-eureka-server</artifactId>
        </dependency>
        <!-- spring boot test-->
        <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>Dalston.RC1</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>
    <repositories>
        <repository>
            <id>spring-milestones</id>
            <name>Spring Milestones</name>
            <url>https://repo.spring.io/milestone</url>
            <snapshots>
                <enabled>false</enabled>
            </snapshots>
        </repository>
    </repositories>
</project>
```

2. 开发服务类

```
@RestController
@RequestMapping("/user")
public class UserController {
    @Value("${server.port}")
    private String port;
    @GetMapping("/findAll")
    public String findAll(@RequestParam  String name){
        return "Hello :"+name+",你好!我是"+port;
    }
```

```
}
```

3. 开发入口类

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

注意:仅仅@EnableEurekaClient 是不够的，还需要在配置文件中注明自己的服务注册中心的地址

4. 编辑配置文件

```
eureka:
  client:
    serviceUrl:
      defaultZone: http://localhost:8080/eureka/
server:
  port: 8989
spring:
  application:
    name: service-hi
```

5. 启动测试,并查看注册中心

a. 访问地址:http://localhost:8989/user/findAll?name=xiaohei

Hello :xiaohei,你好!我是8989

b. 查看注册中心:http://localhost:8080

---

11. HTTP VS RPC

应用间通信方式主要是 HTTP 和 RPC,在微服务架构中两大配方的主角分别是:

Dubbo　RPC 框架

基于 dubbo 开发的应用还是要依赖周边的平台生态, 相比其它的 RPC 框架, dubbo 在服务治理与服务集成上可谓是非常完善，不仅提供了服务注册,发现还提供了负载均衡,集群容错等基础能力同时,还提供了面向开发测试节点的 Mock 和泛化调用等机制。 在 spring cloud 出现之前 dubbo 在国内应用十分广泛，但 dubbo 定位始终是一个 RPC 框架。

SpringCloud 微服务框架（HTTP 通信）

Spring Cloud 的目标是微服务架构下的一栈式解决方案，自 dubbo 复活后 dubbo 官方表示要积极适配到 spring cloud 的生态方式，比如作为 springcloud 的二进制通信方案来发挥 dubbo 的性能优势，或者通过 dubbo 的模块化以及对 http 的支持适配到 Spring Cloud，但是到目前为止 dubbo 与 spring cloud 还是不怎么兼容，spring cloud 微服务架构下微服务之间使用 http 的 RestFul 方式进行通信，Http RestFul 本身轻量易用适用性强，可以很容易跨语言，跨平台，或者与已有的系统集成。

12. 服务消费者(restTemplate+ribbon)

在微服务架构中，业务都会被拆分成一个独立的服务，服务与服务的通讯是基于 http restful 的。Spring cloud 有两种服务调用方式，一种是 ribbon+restTemplate，另一种是 feign。这里优先使用第一个种方式,后续讲解第二种方式。

1. ribbon 简介

Ribbon is a client side load balancer which gives you a lot of control over the behaviour of HTTP and TCP clients. Feign already uses Ribbon, so if you are using @FeignClient then this section also applies.——摘自官网

ribbon 是一个负载均衡客户端，可以很好的控制 htt 和 tcp 的一些行为。Feign 默认集成了 ribbon。

2. **开发服务消费者(service customer)**

> 启动注册中心和服务两个工程,这里注册中心的端口号是 8080,服务的端口号 8989,启动之后创建服务消费者并引入一下依赖 spring-cloud-starter-eureka、spring-cloud-starter-ribbon、spring-boot-starter-web。

1. pom.xml 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>
  <groupId>com.baizhi</groupId>
  <artifactId>springcloud_client</artifactId>
  <version>1.0-SNAPSHOT</version>
  <name>springcloud_client</name>
  <url>http://www.example.com</url>
  <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>
  </properties>
    <!--引入 springboot 的父项目-->
    <parent>
        <groupId>org.springframework.boot</groupId>
```

```
    <artifactId>spring-boot-starter-parent</artifactId>
    <version>1.5.7.RELEASE</version>
  </parent>
<dependencies>
  <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.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>Dalston.RC1</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>
  <repositories>
    <repository>
```

```
            <id>spring-milestones</id>
            <name>Spring Milestones</name>
            <url>https://repo.spring.io/milestone</url>
            <snapshots>
                <enabled>false</enabled>
            </snapshots>
        </repository>
    </repositories>
</project>
```

2. 配置注册中心地址

```
eureka:
  client:
    serviceUrl:
      defaultZone: http://192.168.8.100:8080/eureka/
server:
  port: 9090
spring:
  application:
    name: service-ribbon
```

3. 配置入口类在类中注入一个 RestTemplate

```
@EnableDiscoveryClient
@SpringBootApplication
public class Application {
```

```
    public static void main(String[] args) {
        SpringApplication.run(Application.class,args);
    }
    @Bean
    @LoadBalanced
    RestTemplate getRestTemplate(){
        return new RestTemplate();
    }
}
```

4. 在业务类中注入 restTemplate 进行调用

```
@Service
public class UserServiceImpl implements UserService{
    @Autowired
    private RestTemplate restTemplate;
    @Override
    public String hiService(String name) {
        return restTemplate.getForObject("http://SERVICE-
HI/hi?name="+name,String.class);
    }
}
```

5. 开发 Controller

```
@RestController
@RequestMapping("/user")
```

```
public class UserController {
    @Resource
    private UserService userService;
    @RequestMapping("/findAll")
    public String findAll(String name){
        String content = userService.hiService(name);
        return content;
    }
}
```

6. 启动项目,并进行测试

---

13. 服务消费者(Feigin)

> 服务的消费的两种方式:一种是使用:rest+ribbon、一种是 Feign

1. Feign 简介

> Feign 是一个声明式的伪 Http 客户端，它使得写 Http 客户端变得更简单。使用 Feign，只需要创建一个接口并注解。它具有可插拔的注解特性，可使用 Feign 注解和 JAX-RS 注解。Feign 支持可插拔的编码器和解码器。Feign 默认集成了 Ribbon，并和 Eureka 结合，默认实现了负载均衡的效果。

简而言之:

- Feign 采用的是基于接口的注解
- Feign 整合了 ribbon

2. 使用 Feign 调用服务

1. 启动 eureka 注册中心,启动服务提供者

2. 创建一个新的工程,并引入如下依赖

```
<?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>
  <groupId>com.baizhi</groupId>
  <artifactId>springcloud_feign</artifactId>
  <version>1.5.7.RELEASE</version>
  <name>springcloud_feign</name>
  <!-- FIXME change it to the project's website -->
  <url>http://www.example.com</url>
  <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>
  </properties>
  <parent>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-parent</artifactId>
    <version>1.5.7.RELEASE</version>
```

```
</parent>
<dependencies>
  <dependency>
    <groupId>org.springframework.cloud</groupId>
    <artifactId>spring-cloud-starter-eureka</artifactId>
  </dependency>
  <dependency>
    <groupId>org.springframework.cloud</groupId>
    <artifactId>spring-cloud-starter-feign</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>
    <scope>test</scope>
  </dependency>
</dependencies>

<dependencyManagement>
```

```
  <dependencies>
    <dependency>
      <groupId>org.springframework.cloud</groupId>
      <artifactId>spring-cloud-dependencies</artifactId>
      <version>Dalston.RC1</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>
<repositories>
  <repository>
    <id>spring-milestones</id>
    <name>Spring Milestones</name>
```

```
        <url>https://repo.spring.io/milestone</url>
        <snapshots>
          <enabled>false</enabled>
        </snapshots>
      </repository>
    </repositories>
</project>
```

3. 再配置文件中设置注册中心

```
eureka:
  client:
    serviceUrl:
      defaultZone: http://192.168.8.100:8080/eureka/
server:
  port: 8765
spring:
  application:
    name: service-feign
```

4. 编写入口类

```
@SpringBootApplication
@EnableDiscoveryClient
@EnableFeignClients
public class ServiceFeignApplication {
    public static void main(String[] args) {
```

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

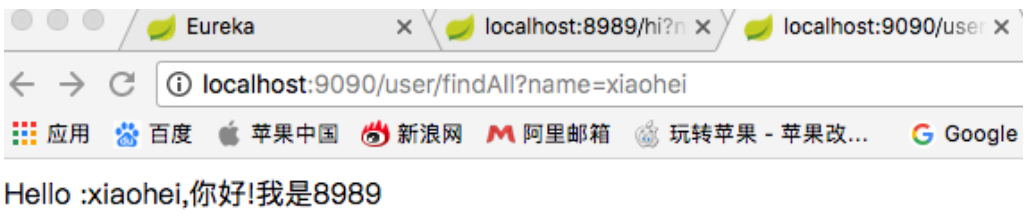
5. 编写 Fiegn 接口

```
@FeignClient("service-hi")
public interface ServiceFeignInterfaces {
    @GetMapping("/hi")
    public String  sayContent(@RequestParam("name") String name);
}
```

6. 编写 Controller

```
@RestController
public class UserController {
    @Value("${server.port}")
    private String port;
    @RequestMapping("/hi")
    public String findAll(@RequestParam  String name){
        return "Hello :"+name+",你好!我是"+port;
    }
}
```

7. 启动浏览器测试

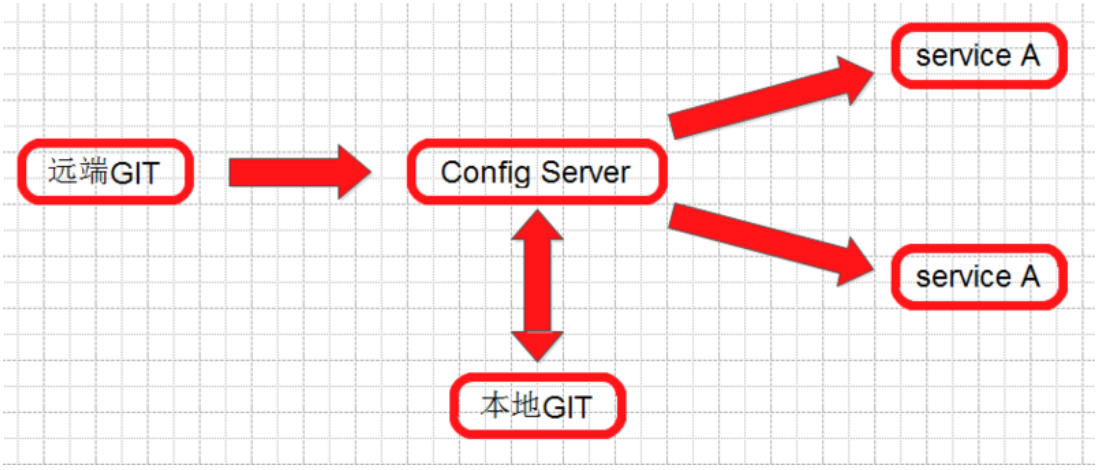


14. 统一配置中心

为什么需要统一配置中心?

> 统一配置中心顾名思义,就是将配置统一管理,配置统一管理的好处是在日后大规模集群部署服务应用时相同的服务配置一致,日后再修改配置只需要统一修改全部同步,不需要一个一个服务手动维护

统一配置中心的架构图:



1. 开发统一配置服务中心:
   a. 新建统一开发中心服务并引入一下依赖

```
<!--引入 springboot 的父项目-->
<parent>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-parent</artifactId>
    <version>1.5.7.RELEASE</version>
</parent>
<dependencies>
    <!--eureka server-->
    <dependency>
        <groupId>org.springframework.cloud</groupId>
        <artifactId>spring-cloud-starter-eureka-server</artifactId>
```

```
    </dependency>
    <!-- spring boot test-->
    <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>Dalston.RC1</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>
<repositories>
    <repository>
        <id>spring-milestones</id>
        <name>Spring Milestones</name>
        <url>https://repo.spring.io/milestone</url>
        <snapshots>
            <enabled>false</enabled>
        </snapshots>
    </repository>
</repositories>
```

b. 创建入口类

```
@SpringBootApplication
@EnableConfigServer
@EnableEurekaClient
```

```
public class ConfigServerApplication {
    public static void main(String[] args) {
        SpringApplication.run(ConfigServerApplication.class,args);
    }
}
```

c. 在远程的仓库创建 yml 配置文件

d. 配置配置中心的项目的 yml 中 git 仓库的地址

```
server:
  port: 9099
eureka:
  client:
    service-url:
      defaultZone: http://localhost:8761/eureka
spring:
  application:
    name: config
  cloud:
```

```
config:
  server:
    git:
      uri: https://gitee.com/chenyn-java/spring-cloud-config.git
      default-label: master
```

e. 启动配置中心服务

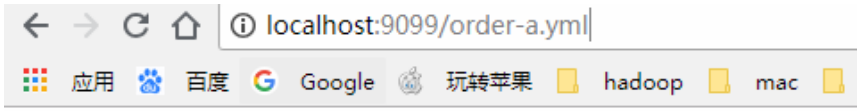
http://localhost:9099/order-a.yml

http://localhost:9099/order-a.yaml

http://localhost:9099/order-a.properties

http://localhost:9099/order-a.json

以上四种访问方式都可以




```
eureka:
  client:
    service-url:
      defaultZone: http://localhost:8761/eureka
mybatis:
  mapper-locations: classpath:com/baizhi/mapper/*.xml
server:
  jsp-servlet:
    init-parameters:
      development: true
  port: 8990
spring:
  application:
    name: order-service
  datasource:
    driver-class-name: com.mysql.jdbc.Driver
    password: root
    type: com.alibaba.druid.pool.DruidDataSource
    url: jdbc:mysql://localhost:3306/shop
    username: root
  jackson:
    date-format: yyyy-MM-dd
```

f. 服务的配置的访问规则

{name}-{profiles}.yml

{name}/{profiles:.*[^-].*}

{name}-{profiles}.json

{label}/{name}-{profiles}.yml

{name}/{profiles}/{label:.*}

{label}/{name}-{profiles}.properties

{label}/{name}-{profiles}.json

{name}/{profile}/{label}/**

{name}/{profile}/{label}/**

说明:

label: 分支名称 默认是 master 分支

name:文件的服务的名称(自定义的名称)

profiles:不同环境

2. 统一配置中心客户端

a. 在现有的服务中加入如下依赖

```
<dependency>
    <groupId>org.springframework.cloud</groupId>
    <artifactId>spring-cloud-config-client</artifactId>
</dependency>
```

b. 修改 yml,删除原有配置,设置配置文件的注册中心

```
spring:
  cloud:
    config:
      name: order
      profile: aa
      discovery:
        enabled: true
        service-id: config(eureka 注册服务名)
```

3. 启动服务发现报错

```
 java.lang.Object.wait(Native Method)
 java.lang.ref.ReferenceQueue.remove(ReferenceQueue.java:143)
 com.mysql.jdbc.AbandonedConnectionCleanupThread.run(AbandonedConnectionCleanu
2018-07-08 13:38:29.335  INFO 10884 --- [  restartedMain] utoConfigurationRepo

Error starting ApplicationContext. To display the auto-configuration report re
2018-07-08 13:38:29.341 ERROR 10884 --- [  restartedMain] o.s.b.d.LoggingFailu

***************************
APPLICATION FAILED TO START
***************************

Description:

Cannot determine embedded database driver class for database type NONE

Action:

If you want an embedded database please put a supported one on the classpath.
```

上述错误的原因是 spring cloud 没有办法区分如何先去找配置中心还是直接启动,因此必须将配置文件的名字改为 bootstrap.yml 或者是 bootstrap.properties 文件重新即可

---

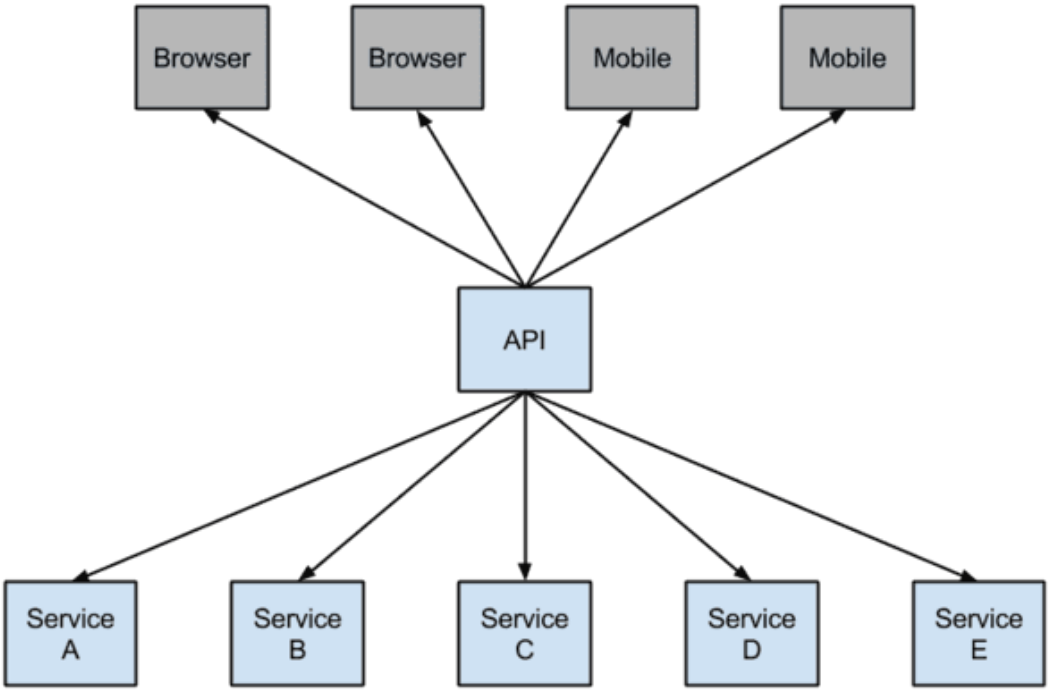
15. 断路器(Hystrix)

在微服务架构中，根据业务来拆分成一个个的服务，服务与服务之间可以相互调用（RPC），在 Spring Cloud 可以用 RestTemplate+Ribbon 和 Feign 来调用。为了保证其高可用，单个服务通常会集群部署。由于网络原因或者自身的原因，服务并不能保证 100%可用，如果单个服务出现问题，调用这个服务就会出现线程阻塞，此时若有大量的请求涌入，Servlet 容器的线程资源会被消耗完毕，导致服务瘫痪。服务与服务之间的依赖性，故障会传播，会对整个微服务系统造成灾难性的严重后果，这就是服务故障的“雪崩”效应。

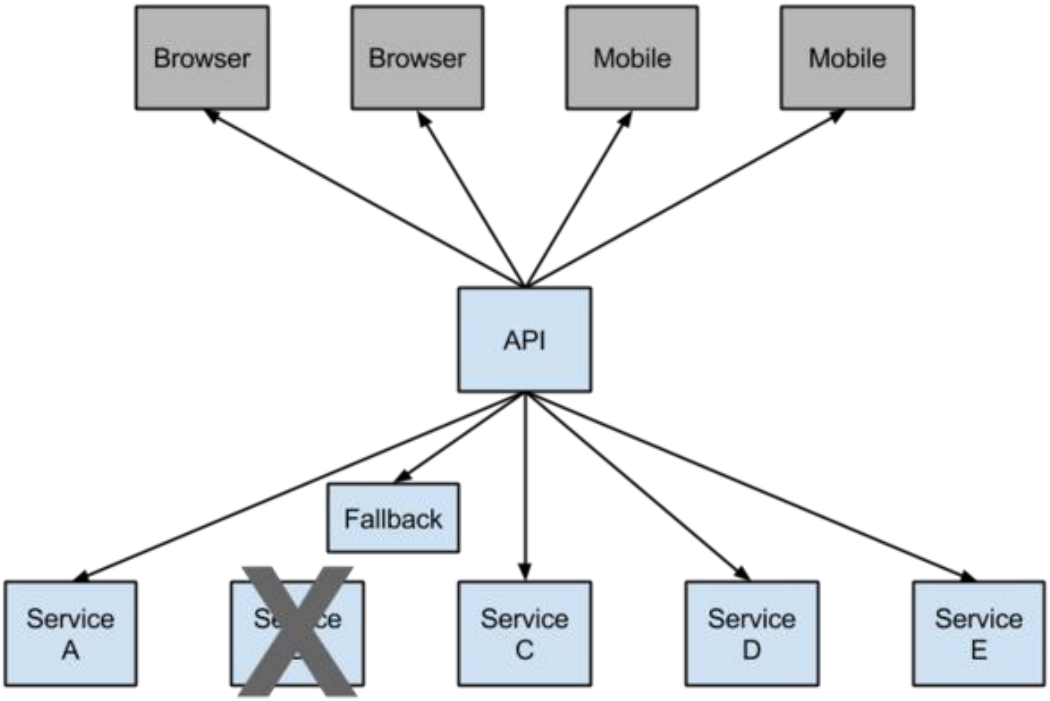
1. 断路器简介

Netflix has created a library called Hystrix that implements the circuit breaker pattern. In a microservice architecture it is common to have multiple layers of service calls.—-摘自官网

Netflix 开源了 Hystrix 组件，实现了断路器模式，SpringCloud 对这一组件进行了整合。 在微服务架构中，一个请求需要调用多个服务是非常常见的,如下架构所展示一样:



如果较底层的服务如果出现故障，会导致连锁故障。当对特定的服务的调用的不可用达到一个阀值（Hystric 是 5 秒 20 次） 断路器将会被打开。如下图所示:



注意:此时断路器打开后，可用避免连锁故障，fallback 方法可以直接返回一个固定值。

2. 使用 Rest+Ribbon 的断路器实现

1. 改造 serice-ribbon 工程的代码，首先在 pox.xml 文件中加入 spring-cloud-starter-hystrix 的起步依赖:

```
<!--加入断路器-->
<dependency>
    <groupId>org.springframework.cloud</groupId>
    <artifactId>spring-cloud-starter-hystrix</artifactId>
</dependency>
```

2. 在程序的启动类 ServiceRibbonApplication 加@EnableHystrix 注解开启 Hystrix：

```
@EnableDiscoveryClient
```

@SpringBootApplication

@EnableHystrix

public class Application

{

public static void main(String[] args)

{

SpringApplication.run(Application.class,args);

}

@Bean

@LoadBalanced

RestTemplate

getRestTemplate(){

return new RestTemplate();

}

}

3.改造服务调用类，在 hiService 方法上加上@HystrixCommand 注解。该注解对该方法创建了熔断器的功能，并指定了 fallbackMethod 熔断方法，熔断方法直接返回了一个字符串，字符串为”hi,”+name+”,sorry,error!”，代码如下：

```
@Service
public class UserServiceImpl implements UserService{
    @Autowired
    private RestTemplate restTemplate;

    @Override
    @HystrixCommand(fallbackMethod = "hiError")//失败的方法名称
    public String hiService(String name) {
        return restTemplate.getForObject("http://SERVICE-
HI/hi?name="+name,String.class);
    }
    public String  hiError(String name){
        return  "hiError==================>:"+name;
    }
}
```

4.启动测试 注册中心,启动服务提供者,启动客户端进行调用发现结果是正常的:

5.关闭服务提供者测试在访问测试:

这就说明当服务工程不可用的时候，service-ribbon 调用服务的 API 接口时，会执行快速失败，直接返回一组字符串，而不是等待响应超时，这很好的控制了容器的线程阻塞。

3. Feign 中使用断路器

1. Feign 是自带断路器的，在低版本的 Spring Cloud 中，它没有默认打开。需要在配置文件中配置打开它，在配置文件加以下代码：

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

2. 基于 Feign 工程进行的改造,只需要在 FeignClient 的服务的接口加入如下实现类：

```
@Component
public class HiServiceFeignImpl implements  ServiceFeignInterfaces{
    @Override
    public String sayContent(String name) {
        return "error:========================>"+name;
    }
}
```

注意:在实现类中执行当服务发生错误时的,处理错误的实现

3. 将定义的实现类指定是为 failbackmethod 的类:

```
@Component
public class HiServiceFeignImpl implements  ServiceFeignInterfaces{
    @Override
    public String sayContent(String name) {
        return "error:========================>"+name;
    }
}
```

4. 启动服务注册中心,服务提供者,启动 Feign 调用者,使用 Feign 调用发现此时是正常的

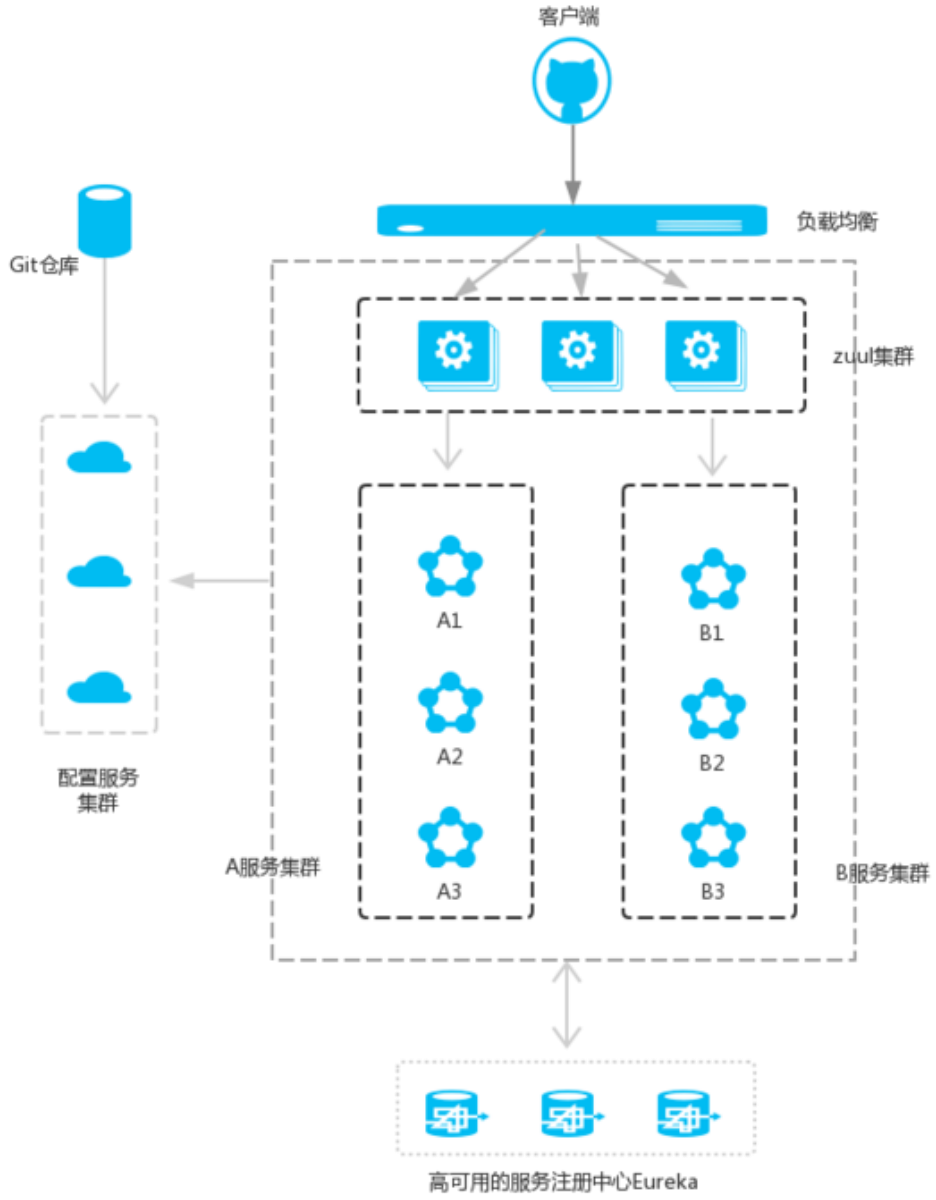
5. 此时停止服务提供者,在此调用发现:,断路器已经生效了:

error:======================>xiaohei

---

16. 路由网关(zuul)

在微服务架构中，需要几个基础的服务治理组件，包括服务注册与发现、服务消费、负载均衡、断路器、智能路由、配置管理等，由这几个基础组件相互协作，共同组建了一个简单的微服务系统。一个简答的微服务系统如下图：



注意：A 服务和 B 服务是可以相互调用的，并且配置服务也是注册到服务注册中心的。

总结:在 Spring Cloud 微服务系统中，一种常见的负载均衡方式是，客户端的请求首先经过负载均衡（zuul、Ngnix），再到达服务网关（zuul 集群），然后再到具体的服。，服务统一注册到高可用的服务注册中心集群，服务的所有的配置文件由配置服务管理（下一篇文章讲述），配置服务的配置文件放在 git 仓库，方便开发人员随时改配置。

1. Zuul 简介

> Zuul 的主要功能是路由转发和过滤器。路由功能是微服务的一部分，比如 / api/user 转发到到 user 服务，/api/shop 转发到到 shop 服务。zuul 默认和 Ribbon 结合实现了负载均衡的功能

2. 搭建 zuul

1. 创建一个新的工程

引入如下依赖:

```
<?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>
  <groupId>com.baizhi</groupId>
  <artifactId>springcloud_zuul</artifactId>
  <version>1.5.7.RELEASE</version>
  <name>springcloud_zuul</name>
  <!-- FIXME change it to the project's website -->
  <url>http://www.example.com</url>
  <properties>
    <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
    <maven.compiler.source>1.7</maven.compiler.source>
    <maven.compiler.target>1.7</maven.compiler.target>
  </properties>
  <parent>
    <groupId>org.springframework.boot</groupId>
```

```
  <artifactId>spring-boot-starter-parent</artifactId>
  <version>1.5.7.RELEASE</version>
</parent>
  <dependencies>
    <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>
    <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>Dalston.RC1</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>
```

```
    <repositories>
        <repository>
            <id>spring-milestones</id>
            <name>Spring Milestones</name>
            <url>https://repo.spring.io/milestone</url>
            <snapshots>
                <enabled>false</enabled>
            </snapshots>
        </repository>
    </repositories>
</project>
```

2. 编写 Zuul 的入口类

```
@EnableZuulProxy
@EnableEurekaClient
@SpringBootApplication
public class ServiceZuulApplication {

    public static void main(String[] args) {
        SpringApplication.run(ServiceZuulApplication.class, args);
    }
}
```

3. 编写 application.yml

```
eureka:
  client:
    serviceUrl:
      defaultZone: http://localhost:8080/eureka/
server:
  port: 8769
spring:
  application:
    name: service-zuul
zuul:
  routes:
    api-a:
      path: /api-a/**
      serviceId: service-ribbon
    api-b:
      path: /api-b/**
      serviceId: service-feign
```

注意:上面 serviceId 书写的是之前的 ribbon 与 feign 的应用名称