# 一、版本说明

## 1、版本号命名规则

采用伦敦地铁站命名，简写为A-Z依次类推迭代版本。如Angel、Brixton等，重要发行版"service releases"，简写SR，如Greenwich.SR2, 为G版第二个发行版。

## 2、版本选择

### Ⅰ SpringBoot版本选择

<https://github.com/spring-projects/spring-boot/releases/>

### Ⅱ SpringCloud版本选择

<https://spring.io/projects/spring-cloud>

### Ⅲ 详细版本依赖

<https://start.spring.io/actuator/info>

# 二、相关组件的的停更/升级/替换

## 1、服务注册中心

### Ⅰ Eureka(停更)

### Ⅱ Zookeeper

### Ⅲ Consul

### Ⅳ Nacos

## 2、服务调用

### Ⅰ Ribbon

### Ⅱ LoadBalancer

### Ⅲ Feign(停更)

### Ⅳ OpenFeign

## 3、服务降级

### Ⅰ HyStrix(停更)

### Ⅱ resilience4j

### Ⅲ sentienl

## 4、服务网关

### Ⅰ Zuul(停更)

### Ⅱ Zuul2

### Ⅲ gateway

## 5、服务配置

### Ⅰ Config(停更)

### Ⅱ Nacos

## 6、服务总线

### Ⅰ Bus(停更)

### Ⅱ Nacos

# 三、微服务架构编码构建(IDEA)

约定 > 配置 > 编码

## 1、微服务cloud整体聚合父工程Project

### Ⅰ 新建Maven工程项目

选择maven-archetype，**org.apache.maven.archetypes:maven-archetype-site**。

Maven不同archetype介绍：[Maven 的41种骨架功能介绍](https://www.cnblogs.com/iusmile/archive/2012/11/14/2770118.html)

### Ⅱ 字符编码

Settings -> Editor ->**File Encodings** -> Global Encoding、Project Encoding、Properties Files设置为UTF-8；勾选 "Transparent native-to-ascii conversion".

### Ⅲ 注解生效激活

Settings -> Build,Execution,Deployment -> Complier -> **Annotation Processors** -> Default -> 勾选:Enable annotation processing

### Ⅳ Java编译版本选择

Settings -> Build,Execution,Deployment -> Complier -> **Java Complier**

### Ⅴ File Type过滤

过滤项目中不常用文件显示(个人习惯)

Settings -> Editor -> File Types

## 2、父工程pom文件

1. 整合：利用标签，约定子项目 中依赖版本号(父项目只声明，不实现引入)
2. 发布：mvn install，父工程创建完成执行mvn:install将父工程发布到仓库方便子工程继承

<properties>  
 <maven.compiler.source>8</maven.compiler.source>  
 <maven.compiler.target>8</maven.compiler.target>  
 <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>  
 <mysql.version>8.0.28</mysql.version>  
 <druid.version>1.1.22</druid.version>  
 <mybatis.spring.boot.version>2.1.3</mybatis.spring.boot.version>  
 <junit.version>4.12</junit.version>  
 <log4j.version>1.2.17</log4j.version>  
 <lombok.version>1.16.20</lombok.version>  
 </properties>  
  
 <dependencyManagement>  
 <dependencies>  
 <dependency>  
 <groupId>org.springframework.boot</groupId>  
 <artifactId>spring-boot-dependencies</artifactId>  
 <version>2.2.2.RELEASE</version>  
 <type>pom</type>  
 <scope>import</scope>  
 </dependency>  
  
 <dependency>  
 <groupId>org.springframework.cloud</groupId>  
 <artifactId>spring-cloud-dependencies</artifactId>  
 <version>Hoxton.SR1</version>  
 <type>pom</type>  
 <scope>import</scope>  
 </dependency>  
  
 <dependency>  
 <groupId>com.alibaba.cloud</groupId>  
 <artifactId>spring-cloud-alibaba-dependencies</artifactId>  
 <version>2.1.0.RELEASE</version>  
 </dependency>  
  
 <dependency>  
 <groupId>mysql</groupId>  
 <artifactId>mysql-connector-java</artifactId>  
 <version>${mysql.version}</version>  
 </dependency>  
  
 <dependency>  
 <groupId>com.alibaba</groupId>  
 <artifactId>druid</artifactId>  
 <version>${druid.version}</version>  
 </dependency>  
  
 <dependency>  
 <groupId>org.mybatis.spring.boot</groupId>  
 <artifactId>mybatis-spring-boot-starter</artifactId>  
 <version>${mybatis.spring.boot.version}</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>  
  
 <dependency>  
 <groupId>org.projectlombok</groupId>  
 <artifactId>lombok</artifactId>  
 <version>${lombok.version}</version>  
 </dependency>  
  
 </dependencies>  
 </dependencyManagement>

## 3、构建Rest微服务工程模块

### Ⅰ 步骤

1. 新建module
2. 配置子工程pom

* <dependencies>  
   <dependency>  
   <groupId>org.springframework.boot</groupId>  
   <artifactId>spring-boot-starter-web</artifactId>  
   </dependency>  
    
   <!--用于检测系统的健康情况、当前的Beans、系统的缓存等-->  
   <dependency>  
   <groupId>org.springframework.boot</groupId>  
   <artifactId>spring-boot-starter-actuator</artifactId>  
   </dependency>  
    
   <dependency>  
   <groupId>org.springframework.boot</groupId>  
   <artifactId>spring-boot-starter-jdbc</artifactId>  
   </dependency>  
    
   <dependency>  
   <groupId>org.springframework.boot</groupId>  
   <artifactId>spring-boot-starter-test</artifactId>  
   <scope>test</scope>  
   </dependency>  
    
   <dependency>  
   <groupId>org.mybatis.spring.boot</groupId>  
   <artifactId>mybatis-spring-boot-starter</artifactId>  
   </dependency>  
    
   <dependency>  
   <groupId>org.projectlombok</groupId>  
   <artifactId>lombok</artifactId>  
   <optional>true</optional>  
   </dependency>  
   </dependencies>

1. 写yml配置文件

* server:  
   port: 80  
    
  spring:  
   application:  
   name: cloud-payment-service  
   datasource:  
   type: com.alibaba.druid.pool.DruidDataSource  
   driver-class-name: com.mysql.cj.jdbc.Driver # mysql驱动包  
   url: jdbc:mysql://localhost:3306/spring\_cloud?useUnicode=true&characterEncoding-utf-8&useSSL=false  
   username: root  
   password: 123456  
    
  mybatis:  
   mapper-locations: classpath:mapper/\*.xml  
   type-aliases-package: com.senmu.springcloud.entities # 所有Entity别名类所在包

1. 主启动类

* @SpringBootApplication  
  public class xxxxMain80 {  
   public static void main(String[] args) {  
   SpringApplication.run(xxxxMain80.class, args);  
   }  
    
  }

1. 业务类
   1. mapper接口
   2. mapper.xml
   3. service接口和service实现类
   4. controller

### Ⅱ 热部署Devtools

spring-boot-devtools是一个为开发者服务的一个模块，其中最重要的功能就是自动应用代码更改到最新的App上面去。原理是在发现代码有更改之后，重新启动应用，但是速度比手动停止后再启动还要更快，更快指的不是节省出来的手工操作的时间。

深层原理是使用了两个ClassLoader，一个Classloader加载那些不会改变的类（第三方Jar包），另一个ClassLoader加载会更改的类，称为restart ClassLoader，这样在有代码更改的时候，原来的restart ClassLoader 被丢弃，重新创建一个restart ClassLoader，由于需要加载的类相比较少，所以实现了较快的重启时间。

### Ⅲ 工程重构

创建通用模块，集合工具类、配置类、常量等

# 四、Eureka服务注册与发现(停更)

## 1、基础知识

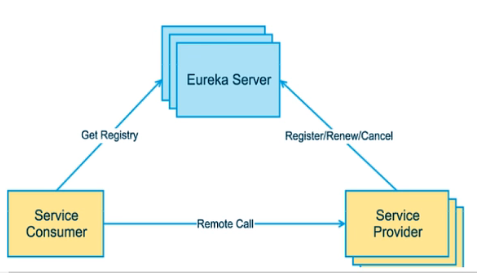
### Ⅰ 什么是服务治理

在传统的rpc远程调用框架中，管理每个服务与服务之间依赖关系比较复杂，因此需要服务治理来实现服务调用、负载均衡、容错等，实现服务发现与注册。

### Ⅱ 什么是服务注册

Eureka采用了CS的设计架构，Eureka Server走位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，是由 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。

在服务注册与发现中，有一个注册中心。当服务启动时，会把自身服务器信息，如通讯地址等以别名方式注册到注册中心上。另一方，以此别名去注册中心获取实际通讯地址，然后再实现本地PRC调用RPC远程调用框架核心设计思想：在于注册中心，因为是由注册中心管理每个服务与服务之间的一个依赖关系（服务治理概念）。任何RPC远程框架中，都会有一个注册中心(存分服务地址相关信息(接口地址))



### Ⅲ Eureka两组件

* **Eureka Server** 提供服务注册服务
* 各个微服务节点通过配置启动后，会在 Eureka Server 中进行注册，这样EurekaServer中的服务注册表中会存储所有可用服务节点的信息。
* **Eureka Client** 通过注册中心进行访问
* 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(roud-robin)负载均衡器。在应用启动后，将会向 Eureka Server 发送心跳(默认周期为30s)。如果Eureka Server在多个心跳周期内没有接收到某个节点的心跳，Eureka Server将会从服务注册表中把这个服务节点移除(默认90s)。

## 2、单机Eureka构建步骤

### Ⅰ IDEA生成Eureka Server端服务注册中心

1. 建立Eureka Server模块
2. pom文件

* <!--Eureka Server-->  
  <dependency>  
   <groupId>org.springframework.cloud</groupId>  
   <artifactId>spring-cloud-starter-netflix-eureka-server</artifactId>  
  </dependency>

1. yml配置

* server:  
   port: 7001  
    
  eureka:  
   instance:  
   hostname: localhost # eureka服务端的实例名称  
   client:  
   # false 表示不向注册中心注册自己  
   register-with-eureka: false  
   # false 表示自己就是注册中心，不需要检索服务  
   fetch-registry: false  
   service-url:  
   # 设置与Eureka Server交互的地址用于查询服务和注册服务  
   defaultZone: http://${eureka.instance.hostname}:${server.port}/eureka/

1. 主启动类

* @EnableEurekaServer

### Ⅱ Eureka Client端注册provider和consumer

1. pom引入eureka

* <!--Eureka Client-->  
  <dependency>  
   <groupId>org.springframework.cloud</groupId>  
   <artifactId>spring-cloud-starter-netflix-eureka-client</artifactId>  
  </dependency>

1. yml配置

* eureka:  
   client:  
   # true 表示向注册中心注册自己  
   register-with-eureka: true  
   # true 是否向Eureka Server检索已有注册信息， 默认true。单节点不影响，集群模式下必须设置true才能配合ribbon使用负载均衡  
   fetch-registry: true  
   service-url:  
   # 设置与Eureka Server交互的地址用于查询服务和注册服务  
   defaultZone: http://localhost:7001/eureka

1. 修改主启动类，加入注解 **@EnableEurekaClient**

## 3、集群Eureka构建步骤

### Ⅰ Eureka 集群原理说明

多台Eureka Server互相注册，实现负载均衡和故障容错

### Ⅱ Eureka Server 集群环境构建步骤

1. 新建Eureka Server模块，POM同上，若在同一台设备上测试，可修改hosts域名映射配置，以标识不同的Server如下

* # C:\Windows\System32\drivers\etc\hosts  
    
  127.0.0.1 eureka7001.com  
  127.0.0.1 eureka7002.com

1. 修改YML，同一集群下的Server需相互注册，如下

* server:  
   port: 7001  
    
  eureka:  
   instance:  
   hostname: eureka7001.com  
   # hostname: localhost # eureka服务端的实例名称  
   client:  
   # false 表示不向注册中心注册自己  
   register-with-eureka: false  
   # false 表示自己就是注册中心，不需要检索服务  
   fetch-registry: false  
   service-url:  
   # 设置与Eureka Server交互的地址用于查询服务和注册服务  
   # defaultZone: http://${eureka.instance.hostname}:${server.port}/eureka/  
   defaultZone: http://eureka7002.com:7002/eureka/

1. 主启动类，同上

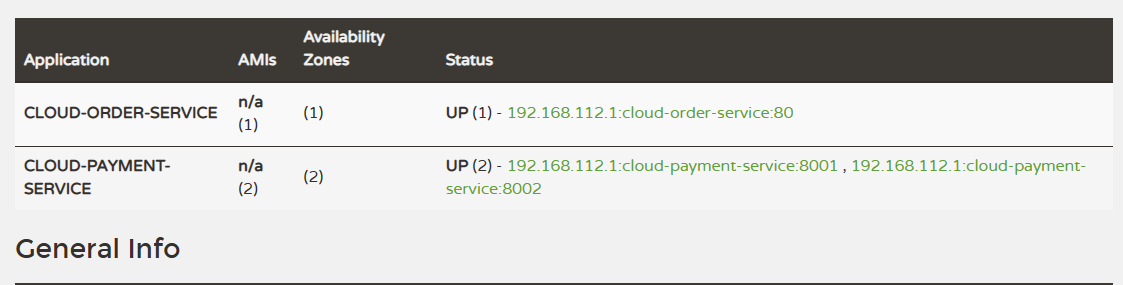
### Ⅲ 服务发布到集群

服务模块修改yml

service-url:  
 # 设置与Eureka Server交互的地址用于查询服务和注册服务  
 # defaultZone: http://localhost:7001/eureka  
 defaultZone: http://eureka7001:com:7001/eureka/,http://eureka7002:com:7002/eureka/

### Ⅳ 构建provider服务的集群

1. 创建别名相同的provider服务模块，并注册

* 

1. consumer服务端使用 @LoadBalanced 注解服务RestTemplate负载均衡的能力。

* package com.senmu.springcloud.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;  
    
  @Configuration  
  public class ApplicationContextConfig {  
    
   @Bean  
   @LoadBalanced // 使用 @LoadBalanced 注解服务RestTemplate负载均衡的能力。  
   public RestTemplate getRestTemplate() {  
   return new RestTemplate();  
   }  
  }

1. consumer服务端controller调用URL前缀

* public static final String PAYMENT\_URL = "http://CLOUD-PAYMENT-SERVICE";

1. 实现负载均衡：默认算法，轮询

* consumer服务controller每次交替调用同别名集群中的provider服务

## 4、actuator微服务信息完善

instance:  
 # 重置Eureka注册服务实例列表status列中，ip:服务名称:端口的默认命名  
 instance-id: payment8001  
 # 访问路径显示ip地址  
 prefer-ip-address: true

## 5、服务发现Discovery

通过Eureka中注册的微服务，获取注册中心中的服务信息

1. 需要使用服务发现的主启动类，配置**@EnableDiscoveryClient**注解
2. 调用org.springframework.cloud.client.discovery.**DiscoveryClient**类中的方法

## 6、Eureka自我保护

### Ⅰ 故障现象

保护模式主要用于一组客户端和Eureka Server之间存在网络分区场景下的保护，一旦进入保护模式，**Eureka Server 将会尝试保护其服务注册表中的信息，不再删除其服务注册表中的数据，也就是不会注销任何微服务。**

当安全模式启动时，System Status中的**Lease expiration enabled**为**false**。且主页出现如下警告：

**EMERGENCY! EUREKA MAY BE INCORRECTLY CLAIMING INSTANCES ARE UP WHEN THEY'RE NOT. RENEWALS ARE LESSER THAN THRESHOLD AND HENCE THE INSTANCES ARE NOT BEING EXPIRED JUST TO BE SAFE.**

紧急！EUREKA 可能错误地声称实例已启动，而实际上它们没有启动。 续订小于阈值，因此实例不会为了安全而过期。

### Ⅱ 导致原因

某时刻某一微服务不可用了，Eureka不会立刻清理，依旧会对该微服务的信息进行保护。为了防止因网络等原因导致无法维持心跳连接，通信超时(默认90秒)，可能健康的实例被注销。

属于分布式系统中**CAP**(Consistency（一致性）、Availability（可用性）、Partition tolerance（分区容忍性）)中的**AP**。

### Ⅲ 如何禁用

1. 注册中心：

* eureka:  
   server:  
   # 表示注册中心是否开启服务的自我保护能力。默认true  
   enable-self-preservation: false  
   # 表示 Eureka Server 清理无效节点的频率，默认 60000 毫秒（60 秒）。  
   eviction-interval-timer-in-ms: 2000

1. 客户端：

* eureka:  
   server:  
   # 表示 Eureka Client 向 Eureka Server 发送心跳的频率（默认 30 秒），如果在 lease-expiration-duration-in-seconds 指定的时间内未收到心跳，则移除该实例。  
   lease-renewal-interval-in-seconds: 30  
   # 表示 Eureka Server 在接收到上一个心跳之后等待下一个心跳的秒数（默认 90 秒），若不能在指定时间内收到心跳，则移除此实例，并禁止此实例的流量。  
   lease-expiration-duration-in-seconds: 90

# 五、Zookeeper服务注册与发现

## 1、简介

zookeeper是一个分布式协调工具，可以实现注册中心功能

下载：[Apache ZooKeeper](https://zookeeper.apache.org/releases.html#download)

## 2、服务注册

### Ⅰ Provider

1. pom：

* 服务器安装的zookeeper注册中心版本低于springcloud自带的**org:apache:zookeeper:zookeeper:x.x.x**版本时需要解决版本冲突。
* <dependency>  
   <groupId>org.springframework.cloud</groupId>  
   <artifactId>spring-cloud-starter-zookeeper-discovery</artifactId>  
   </dependency>

1. yml：

* spring:  
   cloud:  
   zookeeper:  
   connect-string: 192.168.75.131:2181

1. 主启动类：

* 配置**@EnableDiscoveryClient**注解，用于向使用consul或zookeeper走位服务注册中心时注册服务

### Ⅱ Consumer

基本配置同服务提供端上，需额外编写配置类**ApplicationContextConfig**配置**RestTemplate**

### Ⅲ 服务节点类型

主要分为**临时节点**和**持久节点**。默认服务为临时节点，注册中心心跳超时直接注销服务。

# 六、Consul服务注册与发现

## 1、简介

### Ⅰ 官网

1. 介绍：<https://www.consul.io/intro/index.html>
2. 下载：<https://www.consul.io/downloads.html>
3. 关于springcloud：<https://www.springcloud.cc/spring-cloud-consul.html>

### Ⅱ 功能

1. 服务发现：提供HTTP和DNS两种发现方式
2. 健康检查：支持多张方式，HTTP、TCP、Docker、Shell脚本定制化监控
3. KV存储：键值对方式存储
4. 支持多数据中心
5. 可视化Web界面

## 2、安装运行

安装压缩包只有一个consul.exe文件。

cmd，开发者模式启动服务

consul agent -dev

## 3、服务注册

### Ⅰ Provider

1. pom：

* <dependency>  
   <groupId>org.springframework.cloud</groupId>  
   <artifactId>spring-cloud-starter-consul-discovery</artifactId>  
   </dependency>

1. yml：

* spring:  
   cloud:  
   consul:  
   host: localhost  
   port: 8500  
   discovery:  
   service-name: ${spring.application.name}

1. 主启动类：

* 配置**@EnableDiscoveryClient**注解，用于向使用consul或zookeeper走位服务注册中心时注册服务

### Ⅱ Consumer

基本配置同服务提供端上，需额外编写配置类**ApplicationContextConfig**配置**RestTemplate**

## 4、与Eureka和Zookeeper区别

| 组件名 | 语言 | CAP | 服务健康检查 | 对外暴露接口 | Spring Cloud集成 |
| --- | --- | --- | --- | --- | --- |
| Eureka | Java | AP | 可配支持 | HTTP | 已集成 |
| Zookeeper | Java | CP | 支持 | 客户端 | 已集成 |
| Consul | Go | CP | 支持 | HTTP/DNS | 已集成 |

**CAP**：Consistency（一致性）、Availability（可用性）、Partition tolerance（分区容忍性）

CAP理论关注粒度是数据，而不是整体系统设计的策略。最多同时满足两个。

CA：单点集群，满足一致性，可用性的系统，通常可扩展性不强

CP：满足一致性，分区容忍性的系统，通常性能不高

AP：满足可用性，分区容忍性的系统，通常对于一致性要求低

# 七、Ribbon负载均衡服务调用

## 1、概述

### Ⅰ 简介

**Spring Cloud Ribbon**是基于Netflix Ribbon开源项目实现的一套**客户端 负载均衡的工具**。主要功能是提供**客户端的软件负载均衡算法和服务调用**。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。即再配置文件中列出Load Balancer后面所有的机器，Riboon会自动的基于某种规则(如简单轮询，随机连接等)去连接这些机器。可以很容易的通过Ribbon实现自定义的负载均衡算法。

### Ⅱ 官网

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

### Ⅲ 功能

1. LB(负载均衡)：
   * **集中式(服务端)**：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**）。
   * **进程内(本地)**：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，然后再从这些地址基于规则选一个。如**Ribbon**。
2. 基于某种规则(如简单轮询，随机连接等)去调用服务。

LB负载均衡：将用户的请求平摊的分配到多个服务上，从而达到系统的高可用。常见的负载均衡软件有Nginx，LVS，硬件F5等。

Ribbon本地负载均衡客户端 与 Nginx服务端负载均衡的区别：

Nginx是服务器负载均衡，客户端所有请求都会交给nginx，然后由nginx实现转发请求。即负载均衡由服务端实现的。

Ribbon本地负载均衡，再调用微服务接口时，会在注册中心上获取注册信息服务列表后缓存到JVM本地，从而在本地实现RPC远程服务调用技术。

## 2、使用

### Ⅰ POM

<!-- Eureka 已在 spring-cloud-starter-netflix-eureka-client 中引入 -->  
<dependency>  
 <groupId>org.springframework.cloud</groupId>  
 <artifactId>spring-cloud-starter-netflix-ribbon</artifactId>  
</dependency>

### Ⅱ RestTemplate

1. getForObject/getForEntity
2. postForObject/postForEntity

## 3、核心组件IRule

包含轮询、随机等算法。

可替换默认的负载均衡算法。需要对应模块下新建包和配置类去设置。

### Ⅱ 替换Ribbon规则

1. 指定Consumer模块新建Package：

* 自定义配置不能放在@ComponetSacn所扫描的当前包以及子包下，否则自定的配置类会被所有的Ribbon客户端所共享，达不到特殊化定制的目的。

1. 在新Package中创建自定义规则**配置类**：

* @Configuration  
  public class MyRibbonRule {  
    
   @Bean  
   public IRule myRule(){  
   return new RandomRule();  
   }  
  }

1. 该模块主启动类添加**@RibbonClient**注解：

* @RibbonClient(name = "需调用的Provider服务的${server.application.name}", configuration = MyRibbonRule.class)

## 4、负载均衡算法

### Ⅰ 静态负载均衡算法

1. 轮询（Round Robin）：服务器按照顺序循环接受请求。
2. 随机（Random）：随机选择一台服务器接受请求。
3. 权重（Weight）：给每个服务器分配一个权重值，根据权重来分发请求到不同的机器中。
4. IP哈希（IP Hash）：根据客户端IP计算Hash值取模访问对应服务器。
5. URL哈希（URL Hash）：根据请求的URL地址计算Hash值取模访问对应服务器。
6. 一致性哈希（Consistent Hash ）：采用一致性Hash算法，相同IP或URL请求总是发送到同一服务器。

### Ⅱ 动态负载均衡算法

1. 最少连接数（Least Connection）：将请求分配给最少连接处理的服务器。
2. 最快响应（Fastest Response）：将请求分配给响应时间最快的服务器。
3. 观察（Observed）：以连接数和响应时间的平衡为依据请求服务器。
4. 预测（Predictive）：收集分析当前服务器性能指标，预测下个时间段内性能最佳服务器。
5. 动态性能分配（Dynamic Ratio-APM）：收集服务器各项性能参数，动态调整流量分配。
6. 服务质量（QoS）：根据服务质量选择服务器。
7. 服务类型（ToS）: 根据服务类型选择服务器。

### Ⅲ 自定义负载均衡算法

1. 灰度发布：平滑过渡的发布方式，可以降低发布失败风险，减少影响范围，发布出现故障时可以快速回滚，不影响用户。
2. 版本隔离：为了兼容或者过度，某些应用会有多个版本，保证1.0版本不会调到1.1版本服务。
3. 故障隔离：生产出故障后将出问题的实例隔离，不影响其他用户，同时也保留故障信息便于分析。
4. 定制策略：根据业务情况定制跟业务场景最匹配的策略。

权重算法可与其他算法配合，如加权轮询、加权随机等。

### Ⅳ 中间件使用的算法

* Nginx
  1. RoundRobin：轮询。
  2. WeightedRoundRobin：加权轮询。
  3. IPHash：按访问IP的Hash选择服务器。
  4. URLHash：按请求URL的Hash选择服务器。
  5. Fair：根据后端服务器的响应时间判断负载情况，从中选出负载最轻的机器进行分流。
* Dubbo
  1. RandomLoadBalance：加权随机。
  2. RoundRobinLoadBalance：加权轮询。
  3. LeastActionLoadBalance：最少链接数。
  4. ShortestResponseLoadBalance：最短响应时间。
  5. ConsistentHashLoadBalance：一致性Hash。
* Ribbon
  1. RoundRobinRule：轮询。
  2. RandomRule：随机。
  3. WeightedResponseTimeRule：根据响应时间来分配权重的方式，响应的越快，分配的值越大。
  4. BestAvailableRule：会先过滤掉由于多次访问故障而处于断路器跳闸状态的服务，然后选择一个并发量最小的服务。
  5. RetryRule：先按照轮询策略获取服务，如果获取服务失败则在指定时间内进行重试，获取可用的服务。
  6. ZoneAvoidanceRule：根据性能和可用性选择服务。
  7. AvailabilityFilteringRule：会先过滤掉由于多次访问故障而处于断路器状态的服务，还有并发的连接数量超过阈值的服务，然后对剩余的服务列表按照轮询策略进行访问。

## 4、负载均衡算法(自定义)

1. 配置类去掉@LoadBalanced
2. 编写LoadBalancer接口和实现类，编写方法对输入的服务实例列表，根据负载均衡算法返回计算所得的最终服务实例。
3. 接口根据LoadBanlancer的方法确定当前可调用的实例去调用。

# 八、OpenFeign服务接口调用

# 九、Hystrix断路由

# 十、zuul路由网关