# 基本概念

1、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。

2、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。（有点类似Nginx）

# 负载平衡的意义

1、简单来说： 先说集群，集群就是把一个的事情交给多个人去做，假如要做1000个产品给一个人做要10天，我叫10个人做就是一天，这就是集群，负载均衡的话就是用来控制集群，他把做的最多的人让他慢慢做休息会，把做的最少的人让他加量让他做多点。

2、在计算中，负载平衡可以改善跨计算机，计算机集群，网络链接，中央处理单元或磁盘驱动器等多种计算资源的工作负载分布。负载平衡旨在优化资源使用，最大化吞吐量，最小化响应时间并避免任何单一资源的过载。使用多个组件进行负载平衡而不是单个组件，可能会通过冗余来提高可靠性和可用性。负载平衡通常涉及专用软件或硬件，例如多层交换机或域名系统服务器进程。

# 负载均衡的作用

1、提高应用的效率。这个是负载均衡的作用之一，如果客户端的流量过多的话，有可能会导致服务器的反应速度减慢，所以需要将客户端的流量分摊到其他各种服务器上。

2、监测数据安全。负载均衡还能够监测数据安全，通过其内置功能自动隔离有异常情况的主机，以此来保护数据的安全，如果出现了问题就会直接将问题主机直接替换掉。

3、均衡流量。如果应用场景较为复杂，就会使用默认的端口。负载均衡能够为不同域名的网站提供不同的导出IP地址，从而实现流量的均衡。

# 流程

负载均衡，会帮你在每次请求时选择一台机器，均匀的把请求分发到各个机器，Ribbon的负载均衡默认使用的最经典的Round Robin轮询算法。简单来说，就是如果订单服务对库存服务发起10次请求，那就先让你请求第1台机器、然后是第2台机器、第3台机器、第4台机器、第5台机器，接着再来—个循环，第1台机器、第2台机器。。。以此类推。

图示

描述已自动生成

# 原理

图示

描述已自动生成

# 负载均衡方式

1、服务端负载均衡

2、客户端负载均衡

# 服务端负载均衡

## 基本概念

服务端负载均衡是最常见的负载均衡方式，其工作原理如下图。

图示

描述已自动生成

服务端负载均衡是在客户端和服务端之间建立一个独立的负载均衡服务器，该服务器既可以是硬件设备（例如 F5），也可以是软件（例如 Nginx）。这个负载均衡服务器维护了一份可用服务端清单，然后通过心跳机制来删除故障的服务端节点，以保证清单中的所有服务节点都是可以正常访问的。当客户端发送请求时，该请求不会直接发送到服务端进行处理，而是全部交给负载均衡服务器，由负载均衡服务器按照某种算法（例如轮询、随机等），从其维护的可用服务清单中选择一个服务端，然后进行转发。

## 特点

1、需要建立一个独立的负载均衡服务器。

2、负载均衡是在客户端发送请求后进行的，因此客户端并不知道到底是哪个服务端提供的服务。

3、可用服务端清单存储在负载均衡服务器上。

# 客户端负载均衡

## 基本概念

相较于服务端负载均衡，客户端负载均衡则是一个比较小众的概念。客户端负载均衡的工作原理如下图。

图示

描述已自动生成

客户端负载均衡是将负载均衡逻辑以代码的形式封装到客户端上，即负载均衡器位于客户端。客户端通过服务注册中心（例如 Eureka Server）获取到一份服务端提供的可用服务清单。有了服务清单后，负载均衡器会在客户端发送请求前通过负载均衡算法选择一个服务端实例再进行访问，以达到负载均衡的目的。客户端负载均衡也需要心跳机制去维护服务端清单的有效性，这个过程需要配合服务注册中心一起完成。

## 特点

1、负载均衡器位于客户端，不需要单独搭建一个负载均衡服务器。

2、负载均衡是在客户端发送请求前进行的，因此客户端清楚地知道是哪个服务端提供的服务。

3、客户端都维护了一份可用服务清单，而这份清单都是从服务注册中心获取的。

Ribbon 就是一个基于 HTTP 和 TCP 的客户端负载均衡器，当我们将 Ribbon 和 Eureka 一起使用时，Ribbon 会从 Eureka Server（服务注册中心）中获取服务端列表，然后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。

# 服务端负载均衡 VS 客户端负载均衡比较

图形用户界面, 文本, 应用程序, 电子邮件

描述已自动生成

# Nginx与Ribbon的区别

Nginx是反向代理同时可以实现负载均衡，nginx拦截客户端请求采用负载均衡策略，根据upstream配置进行转发，相当于请求通过nginx服务器进行转发。Ribbon是客户端负载均衡，从注册中心读取目标服务器信息，然后客户端采用轮询策略对服务直接访问，全程在客户端操作。

# ribbon涉及的maven依赖

图形用户界面, 文本, 应用程序, 电子邮件

描述已自动生成

spring-cloud-starter-netflix-ribbon

# 代码示例

## Ribbon实现服务调用

### 基本说明

Ribbon 可以与 RestTemplate（Rest 模板）配合使用，以实现微服务之间的调用。RestTemplate 是 Spring 家族中的一个用于消费第三方 REST 服务的请求框架。RestTemplate 实现了对 HTTP 请求的封装，提供了一套模板化的服务调用方法。通过它，Spring 应用可以很方便地对各种类型的 HTTP 请求进行访问。RestTemplate 针对各种类型的 HTTP 请求都提供了相应的方法进行处理，例如 HEAD、GET、POST、PUT、DELETE 等类型的 HTTP 请求，分别对应 RestTemplate 中的 headForHeaders()、getForObject()、postForObject()、put() 以及 delete() 方法。

### 创建服务消费者应用

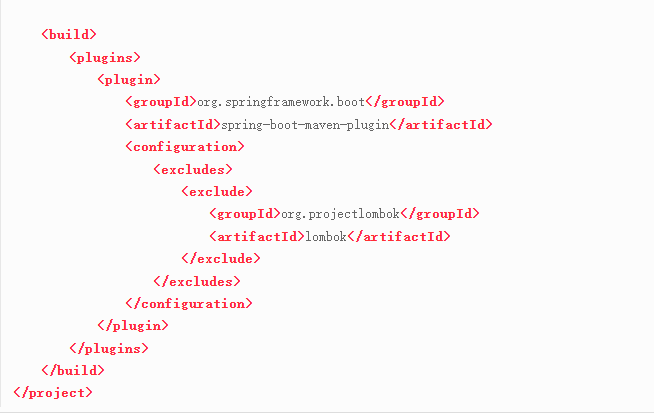
创建一个名为 micro-service-cloud-consumer-dept-80 的微服务，并在其 pom.xml 中引入所需的依赖，代码如下：

文本

描述已自动生成

图片包含 日程表

描述已自动生成



### 新建配置文件

在类路径（即 /resource 目录）下，新建一个配置文件 application.yml，配置内容如下。

图片包含 图示

描述已自动生成

### 创建配置类

在 net.biancheng.c.config 包下，创建一个名为 ConfigBean 的配置类，将 RestTemplate 注入到容器中，代码如下。

文本

描述已自动生成

### 创建Controller类

在 net.biancheng.c.controller 包下，创建一个名为 DeptController\_Consumer 的 Controller，该 Controller 中定义的请求用于调用服务端提供的服务，代码如下。

图形用户界面, 文本, 应用程序, 电子邮件

描述已自动生成

### 修改启动类

在 micro-service-cloud-consumer-dept-80 的主启动类上，使用 @EnableEurekaClient 注解来开启 Eureka 客户端功能，代码如下。

图形用户界面, 文本, 应用程序, 电子邮件

描述已自动生成

依次启动服务注册中心 micro-service-cloud-eureka-7001、服务提供者 micro-service-cloud-provider-dept-8001 以及服务消费者 micro-service-cloud-consumer-dept-80。

### 测试访问

使用浏览器访问“http://eureka7001.com:80/consumer/dept/list”，结果如下图。

文本

描述已自动生成

## Ribbon 实现负载均衡

### 基本说明

Ribbon 是一个客户端的负载均衡器，它可以与 Eureka 配合使用轻松地实现客户端的负载均衡。Ribbon 会先从 Eureka Server（服务注册中心）去获取服务端列表，然后通过负载均衡策略将请求分摊给多个服务端，从而达到负载均衡的目的。Spring Cloud Ribbon 提供了一个 IRule 接口，该接口主要用来定义负载均衡策略，它有 7 个默认实现类，每一个实现类都是一种负载均衡策略。



### 创建数据库表

在 MySQL 数据库中执行以下 SQL 语句，准备测试数据。

表格

描述已自动生成

### 创建微服务

参考 micro-service-cloud-provider-dept-8001，再创建两个微服务 Moudle ：micro-service-cloud-provider-dept-8002 和 micro-service-cloud-provider-dept-8003。

### 修改配置信息

在 micro-service-cloud-provider-dept-8002 中 application.yml 中，修改端口号、数据库连接信息以及自定义服务名称信息（eureka.instance.instance-id），修改的配置如下。

文本

描述已自动生成

在 micro-service-cloud-provider-dept-8003 中 application.yml 中，修改端口号以及数据库连接信息，修改的配置如下。

手机屏幕截图

描述已自动生成

### 启动服务

依次启动 micro-service-cloud-eureka-7001/7002/7003（服务注册中心集群）、micro-service-cloud-provider-dept-8001/8002/8003（服务提供者集群）以及 micro-service-cloud-consumer-dept-80（服务消费者）。

### 测试访问

使用浏览器连续访问“http://eureka7001.com/consumer/dept/get/1”,结果如下图。

图形用户界面, 文本, 应用程序, 电子邮件

描述已自动生成

通过图 4 中 dbSource 字段取值的变化可以看出，Spring Cloud Ribbon 默认使用轮询策略进行负载均衡。

## 切换负载均衡策略

### 基本说明

Spring Cloud Ribbon 默认使用轮询策略选取服务实例，我们也可以根据自身的需求切换负载均衡策略。切换负载均衡策略的方法很简单，我们只需要在服务消费者（客户端）的配置类中，将 IRule 的其他实现类注入到容器中即可。

### 修改配置类

在 micro-service-cloud-consumer-dept-80 的配置类 ConfigBean 中添加以下代码，将负载均衡策略切换为 RandomRule（随机）。

图形用户界面, 文本

描述已自动生成

### 测试访问

重启 micro-service-cloud-consumer-dept-80，使用浏览器访问“http://eureka7001.com/consumer/dept/get/1”，结果如下图。

图形用户界面, 文本, 应用程序, 电子邮件

描述已自动生成

通过图 5 中 dbSource 字段取值的变化可以看出，我们已经将负载均衡策略切换为 RandomRule（随机）。

## 定制负载均衡策略

### 基本说明

通常情况下，Ribbon 提供的这些默认负载均衡策略是可以满足我们的需求的，如果有特殊的要求，我们还可以根据自身需求定制负载均衡策略。

### 自定义规则类

在 micro-service-cloud-consumer-dept-80 中新建一个 net.biancheng.myrule 包，并在该包下创建一个名为 MyRandomRule 的类，代码如下。

文本

描述已自动生成

图形用户界面, 文本, 应用程序

描述已自动生成

### 创建配置类

在 net.biancheng.myrule 包下创建一个名为 MySelfRibbonRuleConfig 的配置类，将我们定制的负载均衡策略实现类注入到容器中，代码如下。

图形用户界面, 文本, 应用程序

描述已自动生成

### 修改启动类

修改位于 net.biancheng.c 包下的主启动类，在该类上使用 @RibbonClient 注解让我们定制的负载均衡策略生效，代码如下。

图形用户界面, 文本, 应用程序

描述已自动生成

### 测试访问

重启 micro-service-cloud-consumer-dept-80，使用浏览器访问“http://eureka7001.com/consumer/dept/get/1”，结果如下图。

图形用户界面, 文本, 应用程序, 电子邮件

描述已自动生成

通过图 6 中 dbSource 字段取值的变化可以看出，我们定制的负载均衡策略已经生效。

# 涉及的接口

## IRule

### 基本概念

### 源码

图形用户界面, 文本

描述已自动生成

### 方法

## ILoadBalancer

### 基本概念

### 源码

文本

描述已自动生成

# 涉及的类

## AbstractLoadBalancerRule

### 基本概念

### 源码

文本

描述已自动生成

# 涉及的注解

## @LoadBalanced

### 源码

图形用户界面, 文本

描述已自动生成

### 作用

 开启客户端负载均衡。