# Hystrix介绍

<https://projects.spring.io/spring-cloud/spring-cloud.html#_circuit_breaker_hystrix_clients>

# Eureka 环境

1 eurekaServer: **eureka**

2 eurekaService: **eurekaservice**

3 ribbon: **eurekaribbon**

4 feign: **eurekafeign**

## eureka项目

### 核心依赖

<dependency>

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

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

</dependency>

### 启动类配置

@SpringBootApplication

@EnableEurekaServer

### application.properties

server.port=9000

eureka.instance.hostname=localhost

eureka.client.register-with-eureka=false

eureka.client.fetch-registry=false

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

### 访问地址

http://localhost:9000

## eurekaservice项目

### 核心依赖

<dependency>

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

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

</dependency>

### 启动类配置

@SpringBootApplication

@EnableEurekaClient

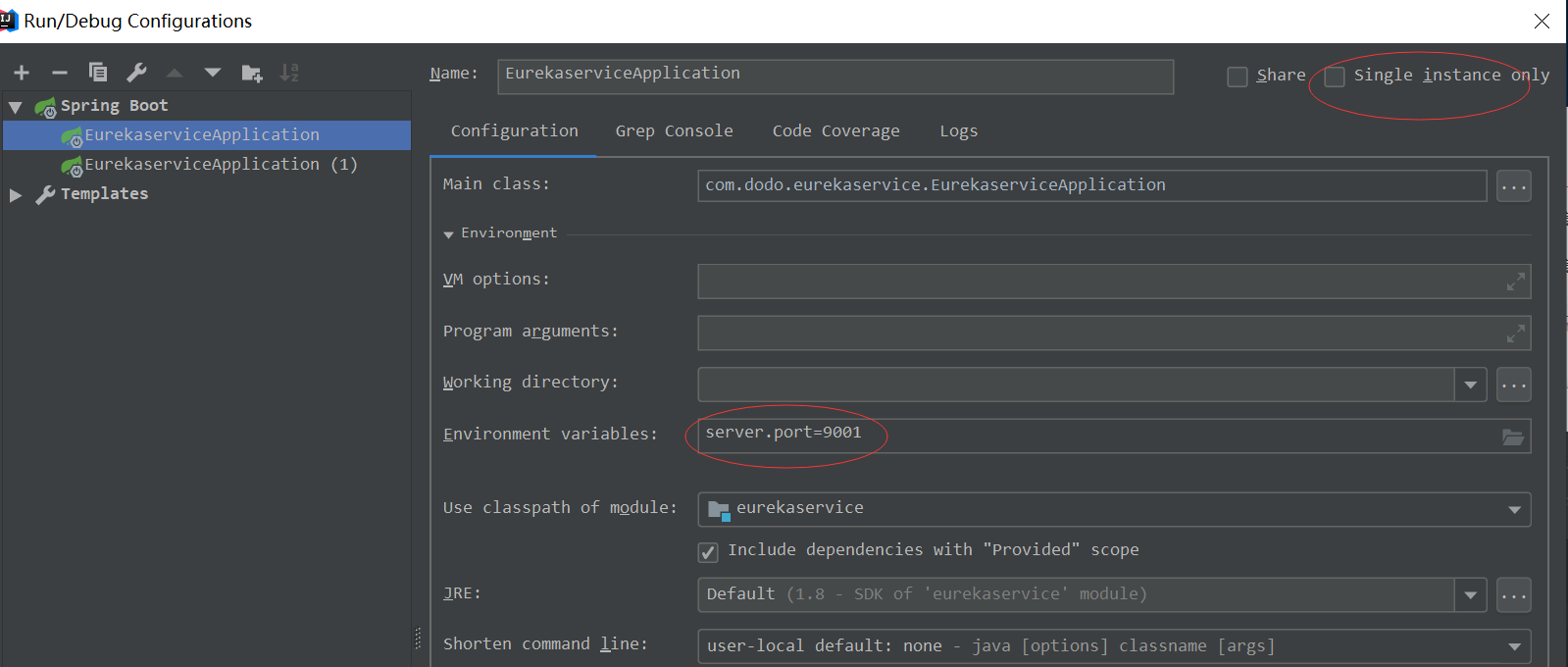
### application.properties

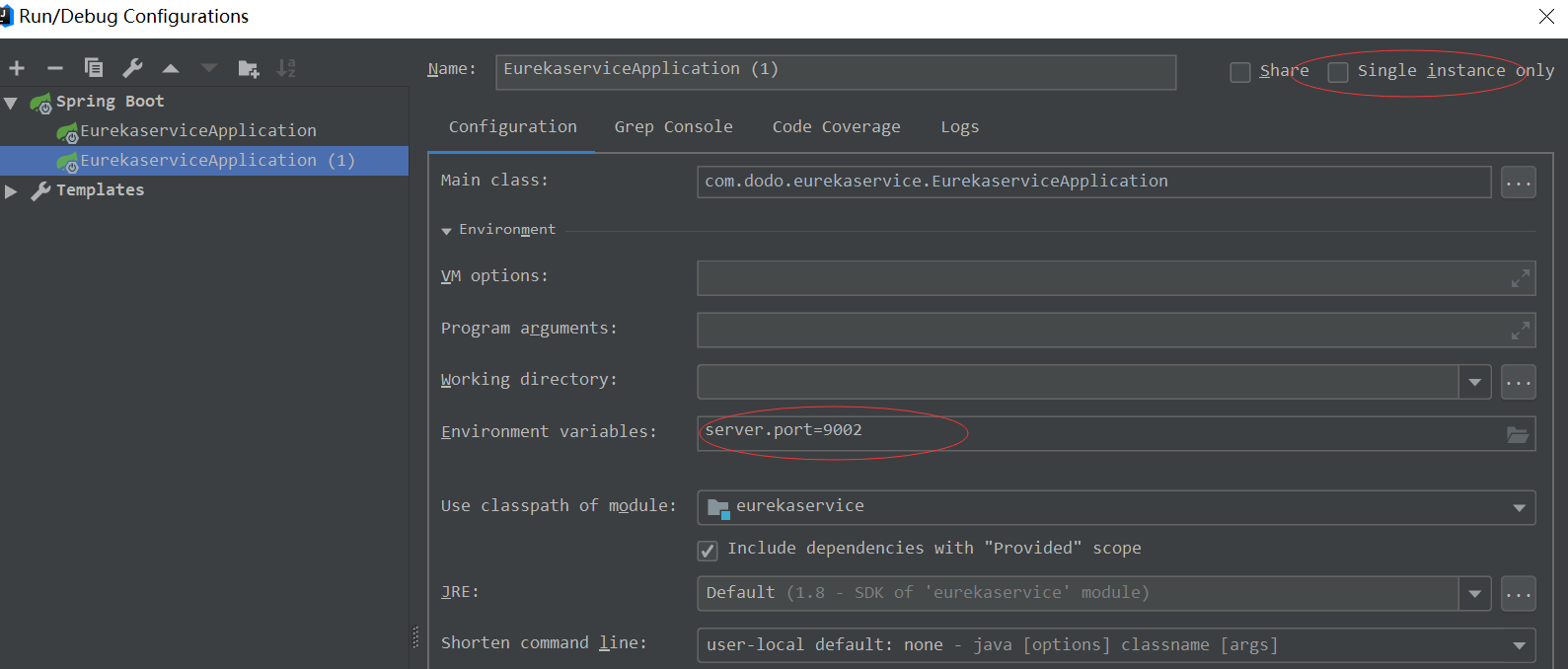
eureka.client.service-url.defaultZone=http://localhost:9000/eureka/

spring.application.name=welcome

some.words=Welcome to Spring Cloud

### 双service配置





### 访问地址

<http://localhost:9001/welcome?name=zhangsan>

http://localhost:9002/welcome?name=zhangsan

## eurekaribbon项目

### ribbon介绍回顾

Spring Cloud的服务有两种消费者，Ribbon是其中之一，它是一个负载均衡客户端，可以很好的控制http和tcp的一些行为。

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，随即连接等）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。

### 核心依赖

<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>

### 启动类配置

@SpringBootApplication

@EnableDiscoveryClient

### rest模板（负载均衡）

@Bean

@LoadBalanced

RestTemplate restTemplate() {

return new RestTemplate();

}

### Service

@Service

public class WelcomeService {

@Autowired

RestTemplate restTemplate;

public String welcomeService(String name) {

return restTemplate.getForObject("http://welcome/welcome?name="+name,String.class);

}

}

### Controller

@RestController

public class WelcomeController {

@Autowired

WelcomeService welcomeService;

@RequestMapping(value = "/welcome")

public String welcome(@RequestParam String name){

return welcomeService.welcomeService(name);

}

}

### application.properties

server.port=9003

eureka.client.service-url.defaultZone=http://localhost:9000/eureka/

spring.application.name=ribbon

## eurekafeign项目

### feign介绍回顾

Spring Cloud的服务有两种消费者，Feign是另一者，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。

### 核心依赖

<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>

### 启动类配置

@SpringBootApplication

@EnableDiscoveryClient

@EnableFeignClients

### Feign interface

@FeignClient(value = "welcome")

public interface WelcomeInterface {

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

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

}

### Feign controller

@RestController

public class WelcomeController {

@Autowired

WelcomeInterface welcomeInterface;

@RequestMapping(value = "/welcome")

public String welcome(@RequestParam String name){

return welcomeInterface.welcomeClientOne(name);

}

}

### application.properties

server.port=9004

eureka.client.service-url.defaultZone=http://localhost:9000/eureka/

spring.application.name=feign

# Hystrix配置

## 基于Ribbon的Hystrix

### 核心依赖

<dependency>

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

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

</dependency>

### 启动类配置

@SpringBootApplication

@EnableDiscoveryClient

@EnableHystrix

### Service

@HystrixCommand(fallbackMethod = "error")

public String welcomeService(String name) {

return restTemplate.getForObject("http://welcome/welcome?name="+name,String.class);

}

public String error(String name) {

return "程序出现了一个" + name + "错误, 请稍后重试!";

}

## 基于feign的Hystrix

### 核心依赖

Feign中，已经集成了Hystrix断路器，在Spring Cloud Dalston及以上的版本中，默认是不启动的

### application.properties打开断路器

feign.hystrix.enabled=true

### Feign interface

@FeignClient(value = "welcome", fallback = WelcomeError.class)

public interface WelcomeInterface {

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

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

}

## Hystrix Dashboard（仪表盘）

### 核心依赖

<dependency>

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

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

</dependency>

### 启动类配置

@SpringBootApplication

@EnableDiscoveryClient

@EnableFeignClients

@EnableHystrixDashboard

### 启动类添加bean

@Bean

public ServletRegistrationBean getServlet(){

HystrixMetricsStreamServlet streamServlet = new HystrixMetricsStreamServlet();

ServletRegistrationBean registrationBean = new ServletRegistrationBean(streamServlet);

registrationBean.setLoadOnStartup(1);

registrationBean.addUrlMappings("/dashboard");

registrationBean.setName("HystrixMetricsStreamServlet");

registrationBean.setName("HystrixMetricsStreamServlet");

return registrationBean;

}

### 访问路径

<http://localhost:9004/hystrix>

输入：

[http://localhost:9004/ dashboard](http://localhost:9004/hystrix)

# Hystrix 模拟

隔离、熔断、降级回退

实例：

使用 Hystrix 需要继承 HystrixCommand 或者 HystrixObservableCommand，实现 HystrixCommand#run 方法，HystrixCommand#getFallback 是可选实现，当你需要对异常或者熔断处理的时候需要实现它。

HystrixCommand 或者 HystrixObservableCommand 有多个构造方法，可以设置名词，组别，线程池等很多参数，com.netflix.hystrix.HystrixCommand.Setter 里面有更多的参数可以设置，具体可以看一下官方的文档。

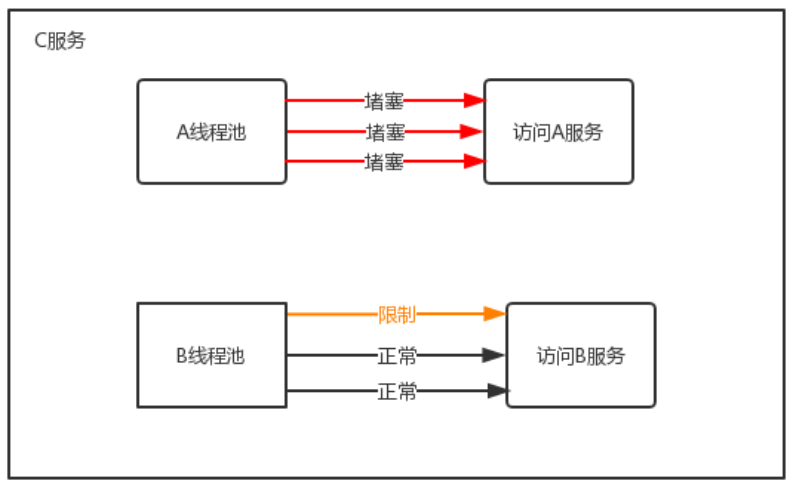
## CommandDemoFailure实现

## 基于线程池的隔离

### 模拟

HystrixThreadIsolation.java

### 图例



## 基于信号量的隔离

### 信号量介绍

场景介绍：有一个停车场只有5个车位，现在有100辆车要去抢这个5个车位，理想情况下最多只有五辆车同时可以抢到车位，那么没有抢到车位的车只能等到，其他的车让出车位，才有机会去使用该车位。

### 模拟

HystrixSemaphoreIsolation.java

### 图例



## 线程池隔离与信号量隔离实现的比对

1是否线程切换

使用线程池隔离会产生线程切换，使用信号量没有线程切换，线程切换会有一定的系统开销

2限流方式

线程池隔离，通过固定队列实现限流；信号量隔离，通过设置信号量大小实现限流，两种方式都支持限流。

3

超时

Hystrix 的超时机制，跟使用线程策略和信号量策略没有关系，都支持超时设置

4熔断

Hystrix 是通过测量实现的，两种方式都支持熔断

5 异步处理

线程池隔离本身就异步处理过程，异步塞实现，信号量隔离处理是同步过程，不支持异步处理。

# **总结**

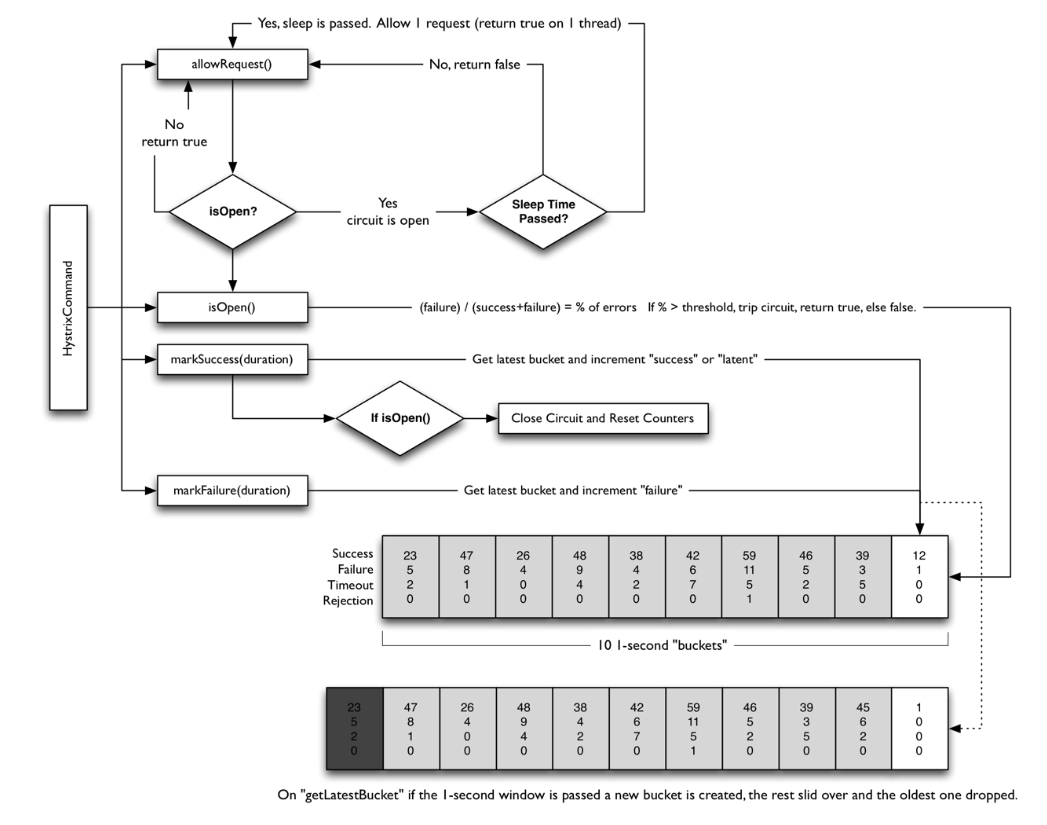
线程池隔离开销比较大，需要线程上下文切换，在一些延时要求比较低的常见不适合使用，系统资源比较紧缺的场景下也不适合使用。

信号量隔离开销小，通过信号量限流，使用信号量隔离会比线程池隔离要好一些。

# **闲聊**

**断路器设计**

**下面是官网的一个断路器流程图**



断路器使用了活动窗口来统计数据，断路器通过活动窗口的数据判断是否需要打开，统计数据包括成功、失败、拒绝、超时等次数，利用的是 RxJava 的活动窗口来实现的

设置活动窗口时间，一个窗口的时间，默认 10 秒：

设置活动窗口的桶的数量，默认为 10 个：

假设我们的窗口时长为 20s，桶数量为 10 个，也就是每两秒钟的统计次数为一个桶。

图中大的长方形有十个小的长方形，大的长方形代表一个窗口，小的长方形代表一个桶，桶里的是统计的数据，包括成功、失败、拒绝、超时次数等。

设置失败率，当失败率超过设置的失败率，断路器会打开：

比如说窗口时间是 5 秒，阈值是 20，在这个窗口时间内请求数量没有 20 即使所有的请求失败了，失败率 100% ，断路器也不会打开。

