# 1 Spring Boot

## 打包方式

Spring Boot的项目打包成jar包的形式，因为web模块依赖包含了嵌入式的Tomcat。

## 模块**依赖**

### spring-boot-starter-web

全栈Web开发模块，包含了嵌入式的Tomcat和Spring MVC。

### spring-boot-starter-test

通用测试模块，包含了JUnit，Hamcrest，Mockito。

### spring-boot-starter-**actuator**

端点监控模块，用来监控端点，比如”health”端点。

### spring-cloud-starter-**eureka**-server

用来搭建Eureka服务端。

### spring-cloud-starter-eureka

用来搭建Eureka客户端。

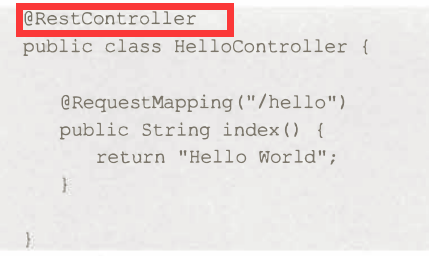
### spring-cloud-starter-**ribbon**

eureka是发现服务，ribbon是提供了对多个服务实例的选择策略。

## **注解**

### @RestController

图1



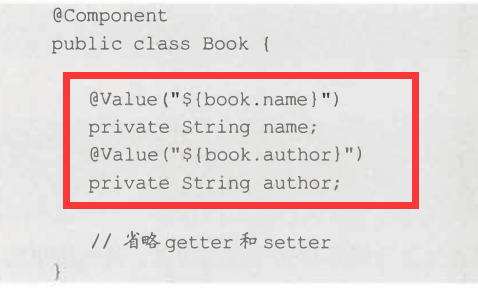
### @Value 加载自定义参数

@Value可以通过PlaceHolder方式，格式为${...}；也可以通过SpEL（Spring Expression Language），格式为#{...}加载参数。

图1（application.properties中的参数）



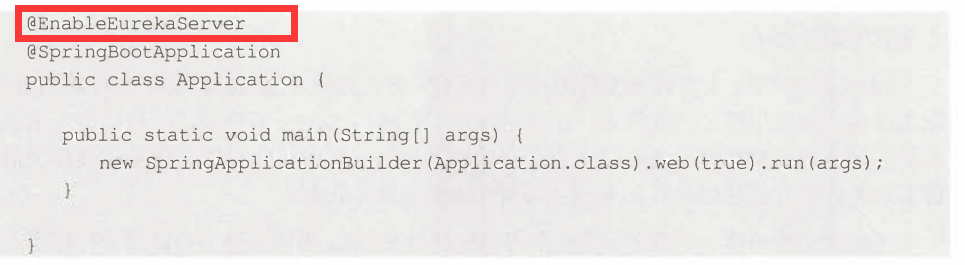
图2（加载application.properties中的参数）



### @**EnableEurekaServer**

@EnableEurekaServer启动一个Eureka服务端。

图1



## 配置文件

### properties和yaml的区别

yaml：

图1



yaml的优点：

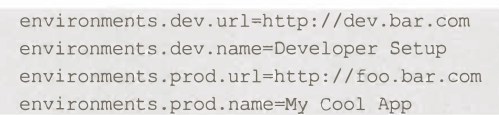
1.有缩进，可读性强

yaml的缺点：

1.无法通过@PropertySource注解来加载配置

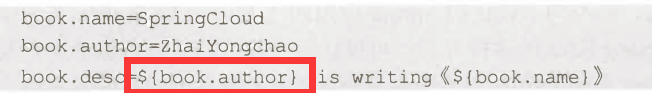
properties：

图2（与yaml等价的properties）



### ${...} 读取上下文参数值

图1



### ${ramdom}**使用**随机数

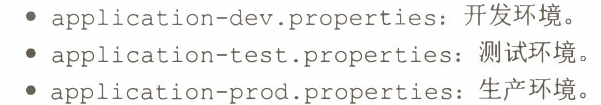
图1



### 多环境**配置**

配置文件的文件名为： application-{profile}.properties，其中profile是环境标识。然后通过设置spring.profiles.active = dev或test或prod，就会加载对应properties文件的内容，spring.profiles.active表示默认加载的环境标识。

图1



## 命令行**参数**

连续的2个-号表示对application.properties中的属性进行赋值。

图1



## 原生端点

原生端点分为3大类：

1. 应用配置类：用来获取配置类信息，比如：应用配置，环境变量，自动化配置报告。
2. 度量指标类：用来获取用于监控的度量指标，比如：内存信息，线程池信息，HTTP请求统计。
3. 操作控制类：用来对应用进行关闭等操作。

# 2 Spring Cloud Eureka

Spring Cloud Eureka使用Netflix Eureka来实现服务的注册和发现。Spring Cloud Eureka分为Eureka服务端和Eureka客户端。

（注意：很多时候Eureka客户端既是服务提供者，也是服务消费者。）

## 2.1 Eureka服务端

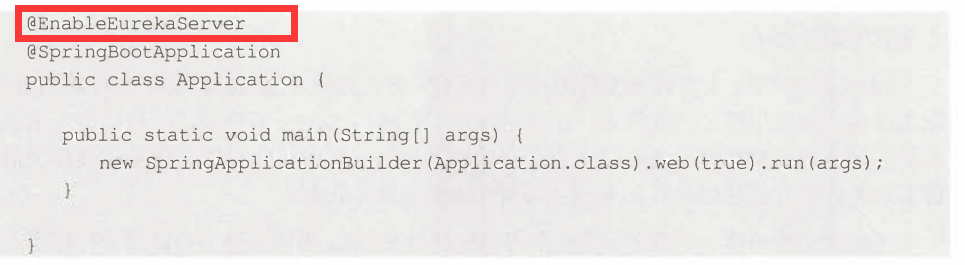
服务端用来给客户端注册和发现服务。

（注意：Eureka服务端类似于Dubbo的注册中心）

### 2.1.1 @**EnableEurekaServer**

@EnableEurekaServer启动一个Eureka服务端。

图1



### 2.1.4 服务同步

当2个注册中心互相注册为服务时，当服务提供者发送一个注册请求到其中一台注册中心，这个注册中心会将请求转发给集群中的其他注册中心，保证每台注册中心都有一样的服务注册信息。

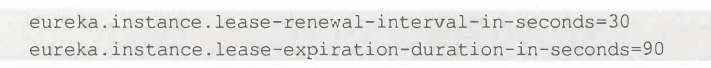
### 2.1.5 服务续约

服务提供者注册完服务后，会维护一个心跳来告诉Eureka Server它还活着，防止Eureka Server把它这个服务实例从服务列表中删除，这个操作成为服务续约。

eureka.instance.lease-renewal-interval-in-seconds：表示客户端发送心跳给Eureka Server的频率，图1中是30秒。

eureka.instance.lease-expiration-duration-in-seconds：表示Eureka Server上一次收到客户端的心跳后，在这个时间内如果没有收到下一次心跳，则移除该服务实例，图1中是90秒。

图1



### 2.1.6 服务下线

当服务下线时，会发送一条下线的REST请求给Eureka Server。Eureka Server收到下线请求后，会将该服务状态标识为下线（DOWN）。

### 2.1.7 失效剔除

当服务不是正常下线，而是因为内存溢出或网络故障等原因使服务不能正常工作时，Eureka Server会每隔一段时间（默认60秒）去剔除那些超时没有续约的服务。

### 2.1.8 自我保护

Eureka Server在运行时会统计心跳失败的比例在15分钟内是否低于85%，如果低于（通常由于网络不稳定造成），则进入自我保护保护，将当前已注册的服务实例保护起来，让这些服务实例不会过期。

（注意：保护起来的这些服务实例可能在这期间会出问题，比如我关掉了某个服务实例，但由于保护机制不会剔除这个实例，当客户端拿到这个服务实例就是不可用的，所以调试模式下会引发自我保护，要关闭自我保护模式防止拿到不可用的服务实例）

## 2.2 Rureka客户端

Rureka客户端用来向Rureka服务端注册服务和发现服务。

（注意：Eureka客户端类似于Dubbo的服务提供者，以及服务消费者，这两者在Eureka中都是Eureka客户端）

### 2.2.1 @**EnableDiscoverClient和@EnableEurekaClient的区别**

@EnableDiscoverClient和@EnableEurekaClient都用来启动一个注册中心的客户端，当注册中心时使用eureka时使用@EnableEurekaClient，使用其他注册中心时使用@EnableDiscoverClient。

### 2.2.2 spring.application.name

spring.application.name指定服务的名字。

图1



### 2.2.3 eureka.client.serviceUrl.defaultZone

eureka.client.serviceUrl.defaultZone指定注册中心的地址。

图2



### 2.2.4 restTemplate

在客户端主类中注入restTemplate，然后使用restTemplate调用其他已注册的服务。

图1

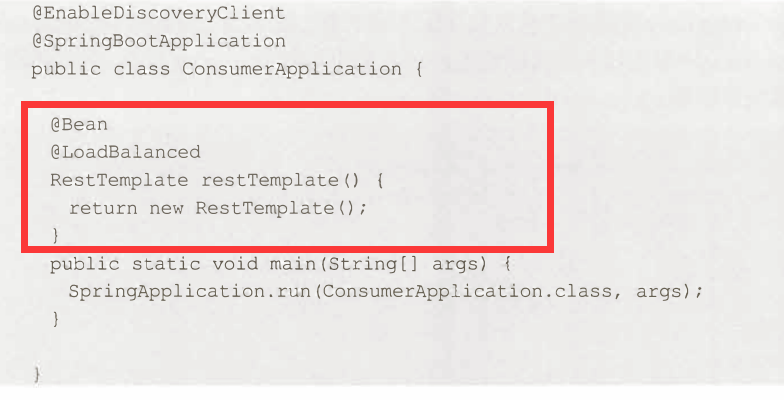
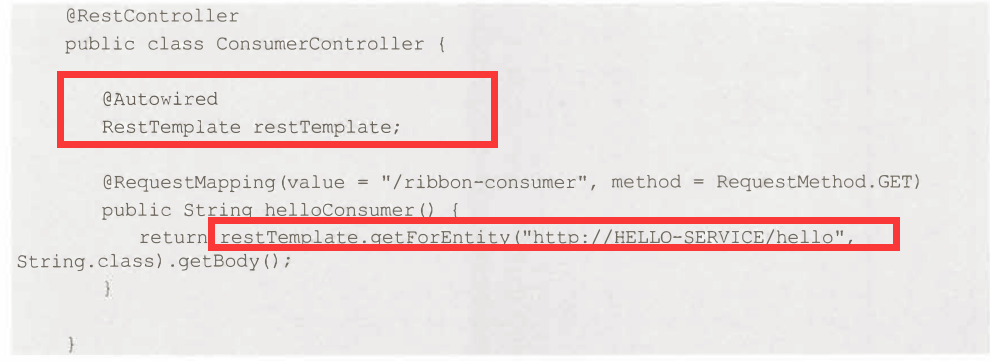


图2



### 2.2.6 eureka.client.registry-fetch-interval-seconds

eureka.client.registry-fetch-interval-seconds表示客户端隔多长时间向Eureka服务端获取服务列表。

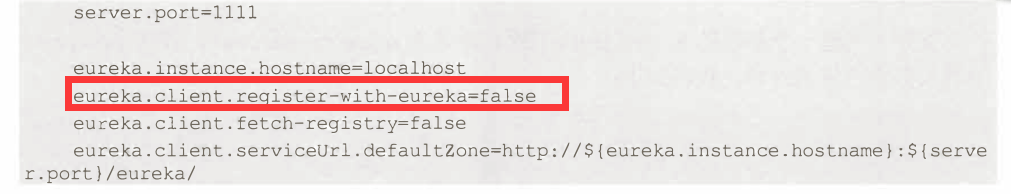
### **2.2.7 eureka**.client.register-with-eureka

eureka.client.register-with-eureka表示是否向eureka注册自己。如果是只有一个eureka服务器则设置成fasle，如果有eureka集群则设置为true。

eureka收到客户端的注册服务请求（包含该服务的元数据）后，会将元数据信息保存在一个双层结构的Map中，第一层的key是服务名，第二层的key是服务的实例名。

（注意：一个服务有多个服务实例）

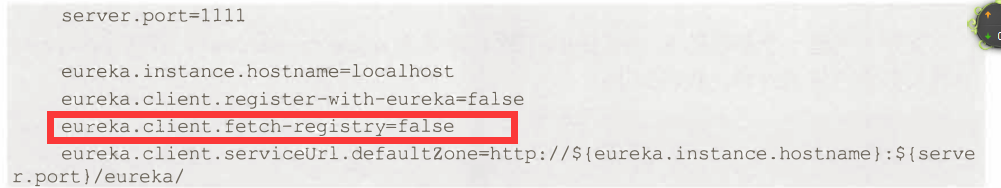
图1



### 2.2.8 eureka.client.fetch-registry

eureka.client.fetch-registry表示是否向eureka检索已注册的服务。如果是服务消费者，则需要开启这项配置，如果是服务提供者可以不开。

图1



### 2.2.9 服务的元数据

元数据存储在InstanceInfo类型的对象中。服务注册时会带上元数据，然后保存在注册中心的一个双层结构的ConcurrentHashMap中，第一层key是服务名，即InstanceInfo的appName，第二层key是实例名，即InstanceInfo中的instanceId。

自定义元数据：

图1



### 2.2.10 实例名的命名规则

因为命名规则中有端口号，所以一个主机可以跑多个服务实例，但端口号不能相同。

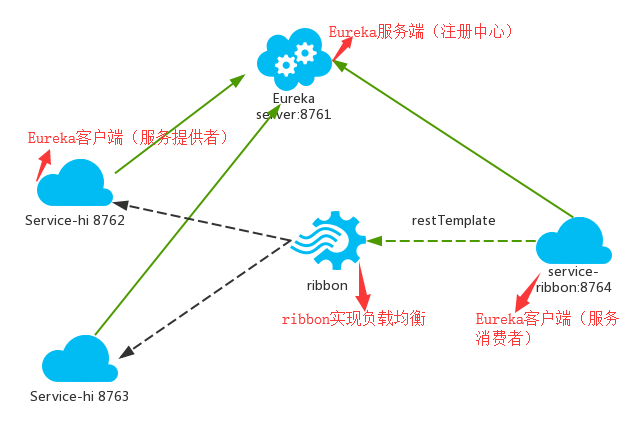
图1



# 3 Spring Cloud Ribbon

Spring Cloud Ribbon用来实现客户端的负载均衡。

图1



## 客户端负载均衡和服务端负载均衡的区别

在客户端负载均衡中，所有客户端节点都要维护着自己要访问的服务端列表，这个服务列表是定时从服务端拿下来的。

## RestTemplate

### @LoadBalanced

@LoadBalanced用来开启客户端的负载均衡。

## Ribbon实现客户端负载均衡的原理

通过LoadBalancerInterceptor拦截器对RestTemplate的请求进行拦截，然后用Spring Cloud的负载均衡器LoadBalancerClient（继承ILoadBalancer接口）对uri中的host主机名通过负载均衡策略（IRule接口的实现）转换成具体的服务实例的地址。

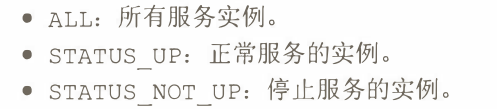
## 负载均衡器

负载均衡器实现了ILoadBalancer接口。

### Abstractload Balancer

Abstractload Balancer是ILoadBalancer接口的抽象实现。它将服务实例分为3种不同的类型组：

图1



## 负载均衡策略

负载均衡策略实现了IRule接口。

## 重试机制

因为Spring Cloud Eureka存在保护机制，但保护机制开启时服务实例列表中可能有故障节点，当遇到故障节点时，重试机制会再次访问当前实例（当前实例重试次数由MaxAutoRetries配置），如果失败，则换一个实例重试，从而增强容错效果。

图1

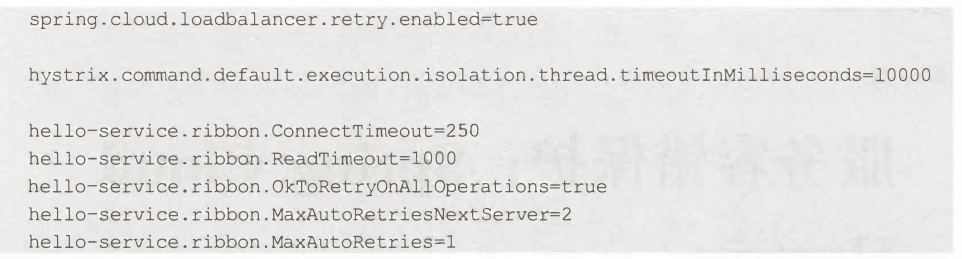


图2



### spring.cloud.loadbalancer.retry.enabled

spring.cloud.loadbalancer.retry.enabled表示开启重试机制，默认关闭。

# 4 Spring Cloud Hystrix

Spring Cloud Hystrix实现了断路器，线程隔离等功能，也是基于Netflix的开源框架Hystrix实现的。

## 4.1 断路器

在微服务架构中，如果较底层的服务出现故障，很可能会导致连锁故障（雪崩效应）。当对某个服务的不可用达到一个阈值时（Hystrix时5秒20次），断路器会被打开，fallback直接返回一个固定值。

图1（没有使用断路器）

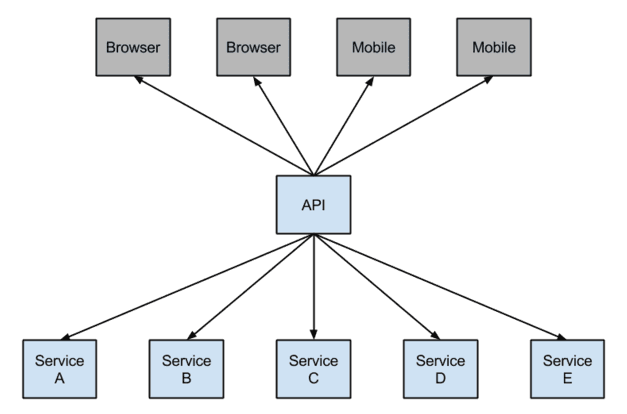
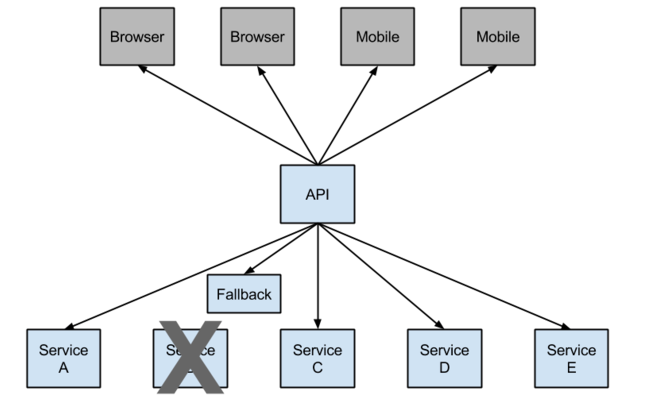


图2（使用了断路器）



### 4.1.1 开启断路器功能

@Enable­CircuitBreaker注解开启断路器功能。

在服务提供者处用fallbackMethod指定失败回调方法。当服务实例无法访问，或服务调用超时（Hystrix默认超时时间是2000毫秒），会触发fallbackMethod的方法返回结果。

图1

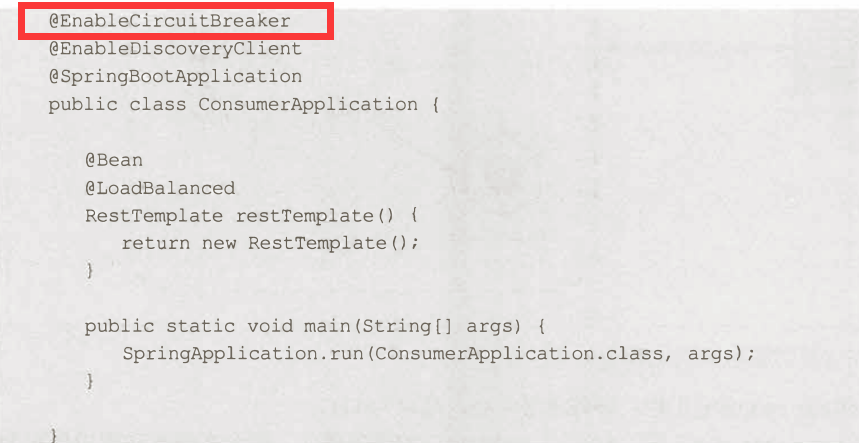


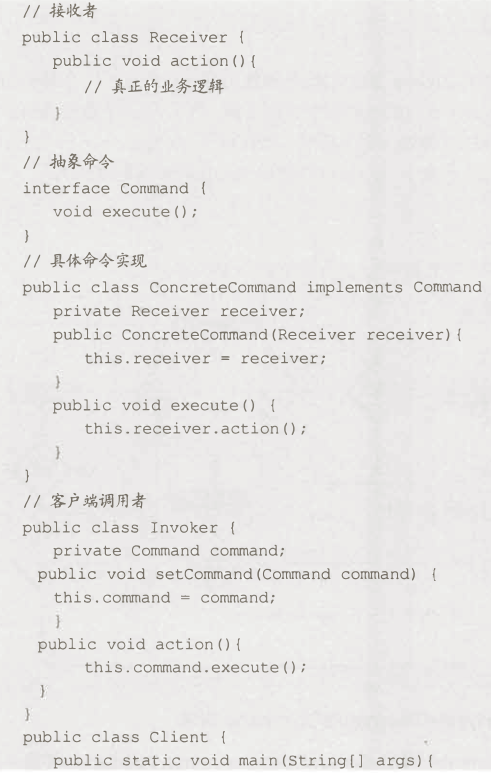
图2

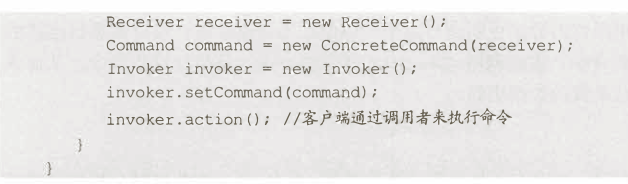


### 4.1.2 命令模式

命令模式：调用者Invoker与操作者Receiver通过命令接口Command实现了解耦。

图1





### 4.1.3 HystrixCommand的命令执行方式

图1

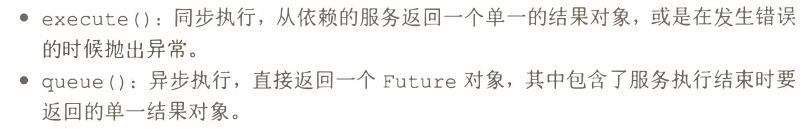


图2



### 4.1.4 HystrixObservableCommand的命令执行方式

图1

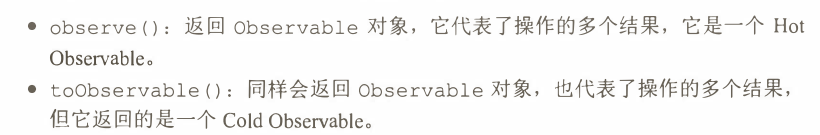


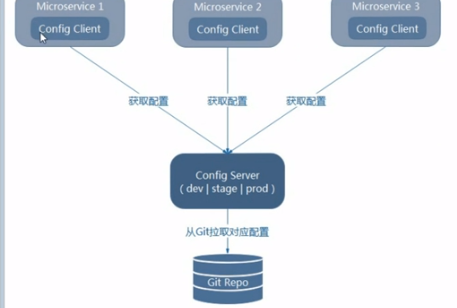
图2



# 5 Spring Cloud Config

每一个微服务会集成Config Client，然后向Config Server获取配置，而配置存储在Git或文件系统上，所以Config Server会向Git或文件系统获取配置，然后返回给Config Client。

图1



## 5.1 Spring Cloud Config的优点

1.修改某一个配置信息，不需要到每一个微服务中都修改一次。

2.修改配置信息后，不需要重启微服务。