# 一站式微服务架构Spring Cloud

主讲：Cat 老师

**北京动力节点教育科技有限公司**  
**2019 • 北京**

**动力节点版权所有，禁止私自传播**

# 快速回顾

1. 分布式与微服务架构的理论梳理；
2. 什么是Spring Cloud？
3. Spring Cloud的整体架构 （与Dubbo比较）
4. 服务消费者Controller直连调用服务提供者Controller（http协议的restful）
5. Spring Cloud的注册中心Eureka
6. Spring Cloud Eureka与Zookeeper比较
7. Spring Cloud Eureka高可用集群
8. Spring Cloud Eureka自我保护机制
9. Spring Cloud Ribbon负载均衡
10. Spring Cloud Feign声明式服务调用（与Dubbo接口层比较）
11. Spring Cloud Hystrix 服务熔断降级、服务限流
12. Spring Cloud Hystrix DashBoard仪表盘监控
13. Spring Cloud Hystrix Turbine聚合监控
14. Spring Cloud Zuul网关（路由、过滤、异常、降级）
15. Spring Cloud Config配置中心（用途、使用、加解密）

# 本次课主题大纲

1. Spring Cloud Config 配置信息局部刷新
2. Spring Cloud Bus + RabbitMQ配置信息全局刷新
3. Spring Cloud Config 高可用
4. Spring Cloud Config 安全认证
5. Spring Cloud Eureka 安全认证

# Spring Cloud Config配置动态刷新

很多场景下，需要在运行期间动态调整配置，如果配置发生了修改，微服务要如何实现配置的动态刷新呢？

## 2.1 配置信息局部刷新

Spring Boot 的actuator提供了一个刷新端点/refresh，添加依赖spring-boot-starter-actuator，可用于配置的刷新；

1、添加依赖：

*<!--springboot的一个监控actuator-->*<**dependency**>  
 <**groupId**>org.springframework.boot</**groupId**>  
 <**artifactId**>spring-boot-starter-actuator</**artifactId**>  
</**dependency**>

2、在Controller上添加注解@RefreshScope，添加这个注解的类会在配置更新时得到特殊的处理；

1. 打开web访问端点：management.endpoints.web.exposure.include=\*
2. 访问<http://localhost:8080/actuator/refresh> 进行手动刷新配置；

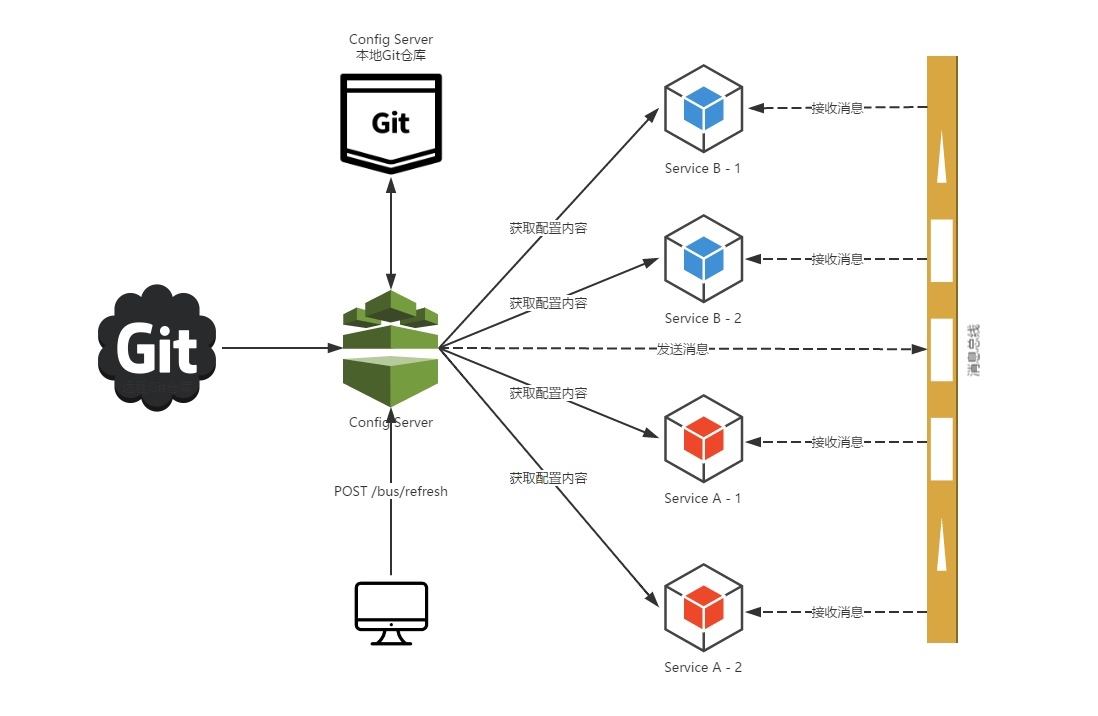
（Post访问）

这种方式的刷新，就是你对每个微服务分别进行刷新，也一个一个操作，如果你有80个微服务，那么就需要手动刷新这80个微服务；

## 2.1 配置信息全局刷新

前面使用/actuator/refresh端点手动刷新配置虽然可以实现刷新，但所有微服务节点的配置都需要手动去刷新，如果微服务非常多，其工作量非常庞大。因此，实现配置的自动刷新是志在必行，Spring Cloud Bus就可以用来实现配置的自动刷新；

Spring Cloud Bus使用轻量级的消息代理/总线（例如RabbitMQ、Kafka等）广播传播状态的更改（例如配置的更新）或者其他的管理指令，可以将Spring Cloud Bus想象成一个分布式的Spring Boot Actuator；



**Config项目添加依赖：（服务端）**

*<!-- spring-cloud-starter-bus-amqp -->*<**dependency**>  
 <**groupId**>org.springframework.cloud</**groupId**>  
 <**artifactId**>spring-cloud-starter-bus-amqp</**artifactId**>  
</**dependency**>

*<!-- spring-boot-starter-actuator -->*<**dependency**>  
 <**groupId**>org.springframework.boot</**groupId**>  
 <**artifactId**>spring-boot-starter-actuator</**artifactId**>  
</**dependency**>

配置文件增加rabbitmq的配置：

*#配置rabbitmq***spring.rabbitmq.host**=**192.168.10.128  
spring.rabbitmq.port**=**5672  
spring.rabbitmq.username**=**guest  
spring.rabbitmq.password**=**guest***#开启spring cloud bus，默认是开启的，也可以省略该配置***spring.cloud.bus.enabled**=**true**

*#打开所有的web访问端点***management.endpoints.web.exposure.include**=**\***

**各个微服务（客户端）**

其他各个微服务用于接收消息，那么也需要有spring cloud bus的依赖和RabbitMQ的连接信息；

*<!-- spring-cloud-starter-bus-amqp -->*<**dependency**>  
 <**groupId**>org.springframework.cloud</**groupId**>  
 <**artifactId**>spring-cloud-starter-bus-amqp</**artifactId**>  
</**dependency**>

*#配置rabbitmq***spring.rabbitmq.host**=**192.168.10.128  
spring.rabbitmq.port**=**5672  
spring.rabbitmq.username**=**guest  
spring.rabbitmq.password**=**guest**

然后post方式请求地址：<http://localhost:8888/actuator/bus-refresh> 如果返回成功1，则RabbitMQ将收到消息，然后微服务会消费消息，config的所有客户端的微服务配置都会动态刷新；

# 3、Spring Cloud Config高可用

有了配置中心之后，其他的微服务都是从配置中心上获取配置信息，此时配置中心就至关重要了，在真实的项目环境中，Spring Cloud Config配置中心难免会出现各种问题，此时就需要考虑Spring Cloud Config的高可用机制了；

Spring Cloud Config的高可用机制解决方式非常简单，把Spring Cloud Config注册到Eureka就搞定了，此时用户访问的时候不是直接从配置中心获取配置信息，而是先通过eureka中获取配置中心的地址，然后再从配置中心获取具体服务的配置信息；

我们准备三个配置中心：

application-config8887.properties

application-config8888.properties

application-config8889.properties

并向注册中心注册自己的服务：

*#每间隔10s，向服务端发送一次心跳，证明自己依然"存活"***eureka.instance.lease-renewal-interval-in-seconds**=**10***#告诉服务端，如果我60s之内没有给你发心跳，就代表我故障了，将我踢出掉***eureka.instance.lease-expiration-duration-in-seconds**=**60***#告诉服务端，服务实例以IP作为链接，而不是取机器名***eureka.instance.prefer-ip-address**=**true  
eureka.client.eureka-server-connect-timeout-seconds**=**30  
eureka.client.eureka-server-read-timeout-seconds**=**60***#告诉服务端，服务实例的id，id要是唯一的***eureka.instance.instance-id**=**34-springcloud-service-config-8887***#eureka注册中心的连接地址***eureka.client.service-url.defaultZone**=**http://192.168.10.128:8761/eureka,http://192.168.10.128:8762/eureka,http://192.168.10.128:8763/eureka**

添加eureka-client依赖：

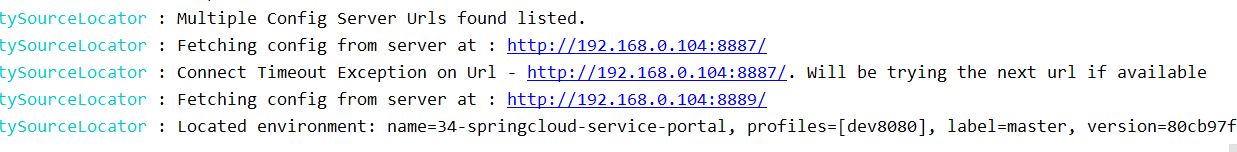
*<!--spring-cloud-starter-netflix-eureka-client-->*<**dependency**>  
 <**groupId**>org.springframework.cloud</**groupId**>  
 <**artifactId**>spring-cloud-starter-netflix-eureka-client</**artifactId**>  
</**dependency**>

在客户端增加对eureka注册中心的配置：

bootstrap.properties文件：

**spring.cloud.config.profile**=**dev8080  
spring.cloud.config.label**=**master***#配置中心的url可以通过注册中心发现***spring.cloud.config.discovery.enabled**=**true***# 在eureka中注册的服务名***spring.cloud.config.discovery.service-id**=**34-SPRINGCLOUD-SERVICE-CONFIG***#eureka注册中心的连接地址***eureka.client.service-url.defaultZone**=**http://192.168.10.128:8761/eureka,http://192.168.10.128:8762/eureka,http://192.168.10.128:8763/eureka**

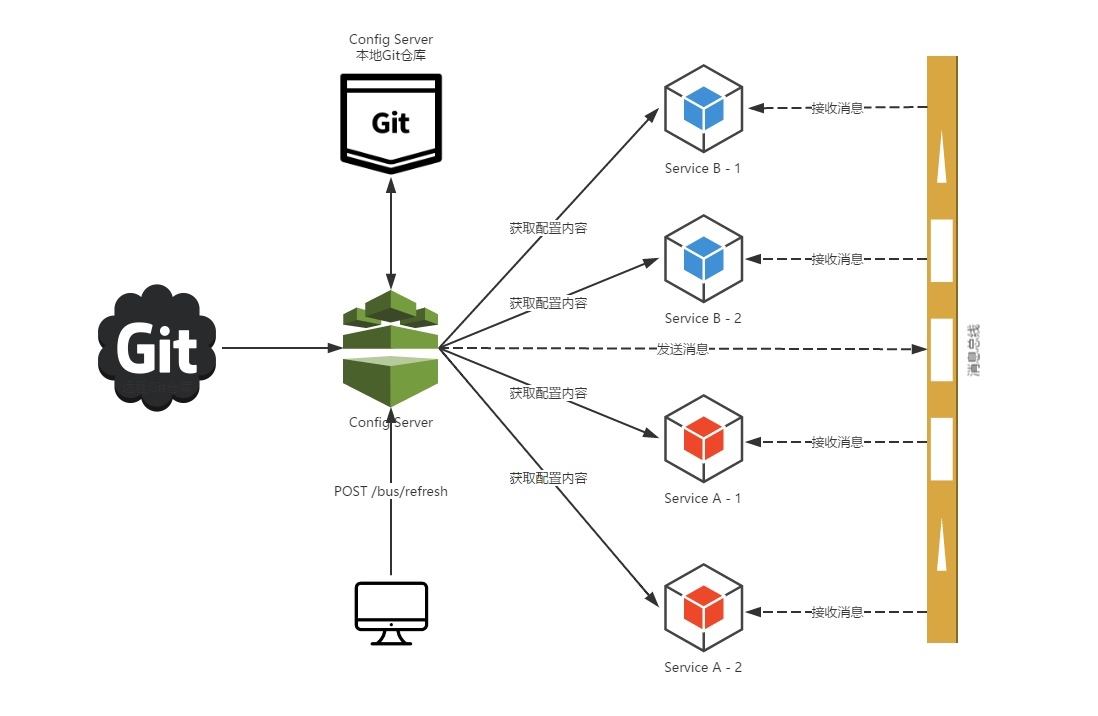
当某些配置中心宕机后，客户端的微服务会基于注册中心去尝试下一个配置中心的url；



Spring cloud config 高可用第二种方式：使用Nginx

*spring.cloud.config.uri=http://nginx的ip:nginx端口/ 这个url写成访问nginx，然后通过nginx负载均衡转发到3个config服务；*

# Spring Cloud Config安全认证



配置中心服务端：

添加依赖：

*<!-- spring-boot-starter-security -->*<**dependency**>  
 <**groupId**>org.springframework.boot</**groupId**>  
 <**artifactId**>spring-boot-starter-security</**artifactId**>  
</**dependency**>

在application.properties文件中配置账号和密码：

*#配置访问账号***spring.security.user.name**=**cat***#配置访问密码***spring.security.user.password**=**123456**

客户端：

在bootstrap.properties文件中配置访问账号和密码：

*#配置访问config配置中心的访问账号和密码***spring.cloud.config.username**=**cat  
spring.cloud.config.password**=**123456**

# 5、Spring Cloud Eureka的安全认证

一般情况下Eureka 都会在一个内网环境中，但免不了在某些项目中需要让其他外网的服务注册到Eureka，这个时候就有必要让Eureka增加一套安全认证机制了，让所有服务提供者通过安全认证后才能注册进来；

服务端：

1、添加依赖

*<!-- spring-boot-starter-security -->*<**dependency**>  
 <**groupId**>org.springframework.boot</**groupId**>  
 <**artifactId**>spring-boot-starter-security</**artifactId**>  
</**dependency**>

2、配置文件

*#配置访问账号***spring.security.user.name**=**cat***#配置访问密码***spring.security.user.password**=**123456**

3、在eureka服务端编写配置类EurekaSecurityConfig，重写configure方法，把csrf劫持置为不可用，让服务能被接收和注册；

*/\*\*  
 \* csrf置为不可用  
 \*/*@Configuration  
@EnableWebSecurity  
**public class** EurekaSecurityConfig **extends** WebSecurityConfigurerAdapter {  
 @Override  
 **protected void** configure(HttpSecurity http) **throws** Exception {  
 http.csrf().disable();  
 **super**.configure(http);  
 }  
}

客户端：

访问的时候注册中心url中添加用户名和密码：

*#eureka注册中心的连接地址***eureka.client.service-url.defaultZone**=**http://cat:123456@localhost:8761/eureka**