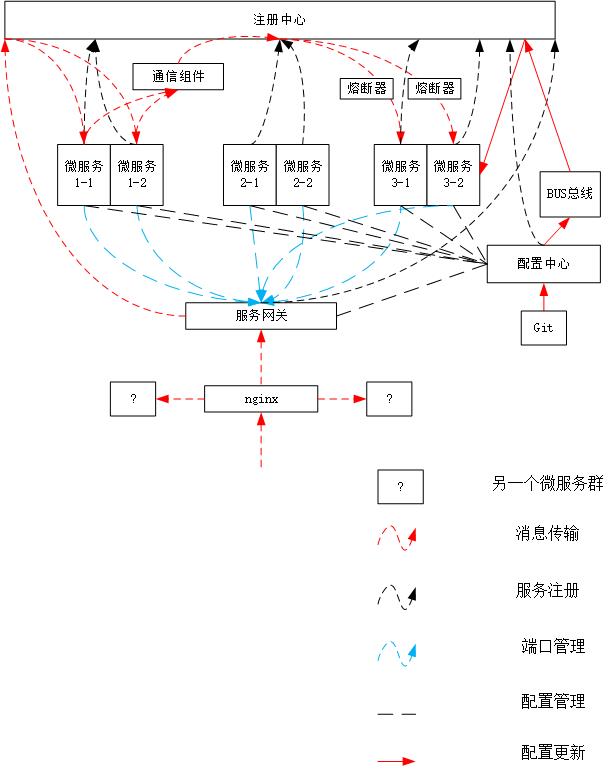
SpringCLoud笔记整理

1. 概述
2. 定义：考虑到高并发、可维护的前提下，基于业务，分割成多个独立的服务，各个服务之间通过暴露的api连接，来组成庞杂的服务系统，服务之间不一定在同一个虚拟机、服务器中。
3. 微服务的核心问题：服务的治理和监控，市面上解决微服务的有dubbo和Springcloud, dubbo通信效率比springCLoud更高，因为RPC通信比http通信效率高，但是和spring集成不是很好。
4. springcloud是一个组件集或者工具集，提供各类组件来帮助为服务的管理，通信等。SpringCloud的使用，其实就是各类组件的使用。
5. 各种组建的使用方法大相径庭：1.导jar包2.加配置



# 1.核心组件说明

- eurekaserver、consul、nacos 服务注册中心组件

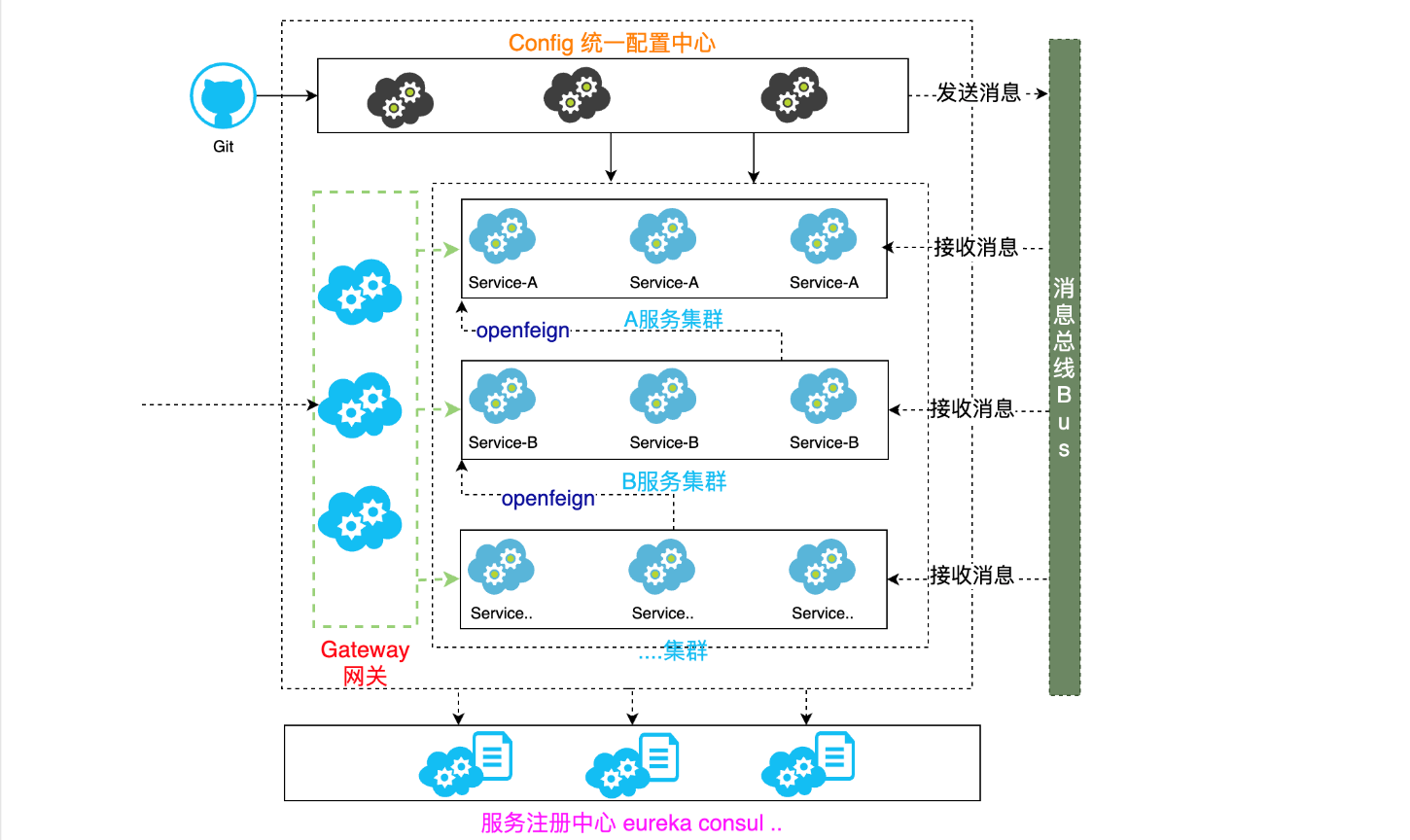
- rabbion & openfeign 服务负载均衡 和 服务调用组件

- hystrix & hystrix dashboard 服务断路器 和 服务监控组件

- zuul、gateway 服务网关组件

- config 统一配置中心组件

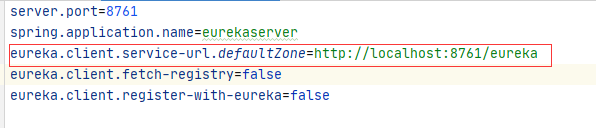
- bus 消息总线组件

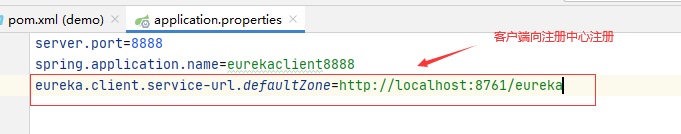


1. 面试答：我们springcloud用的是H SR5版本，springBoot用的是2..2.5版本（详情见课程笔记）
2. 搭建教程见笔记。

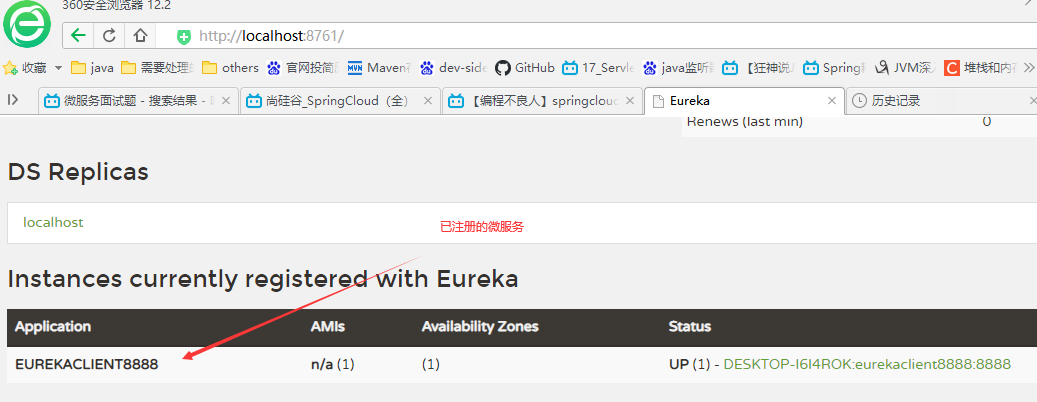
## 组件一：服务注册中心

1. 注册中心负责各个微服务的端口、IP等信息的记录，新的为服务的发现，以及通过心跳来监测微服务是否down机，如发现某实例长时间无法访问，就会从注册中心移除该实例，但是不影响客户端访问。
2. 常见的注册中心：Eureka(netflix)、Consul、Zookeeper、以及阿里巴巴推出Nacos组件
3. **Eureka的使用（具体见笔记）（停更了，了解下就行）**
4. Eureka通过配置文件中暴露的注册地址，微服务配置文件中绑定该地址来完成注册。

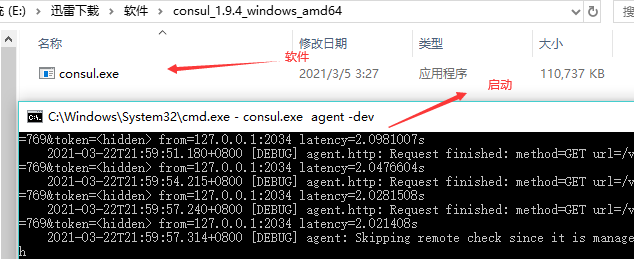


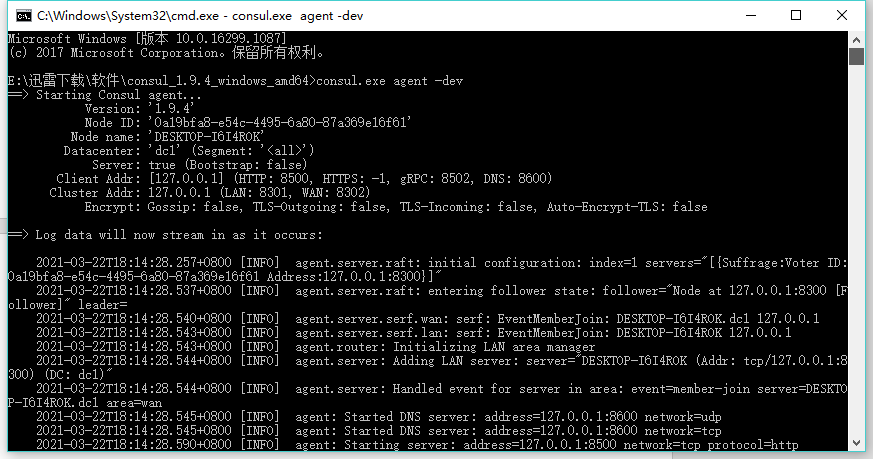


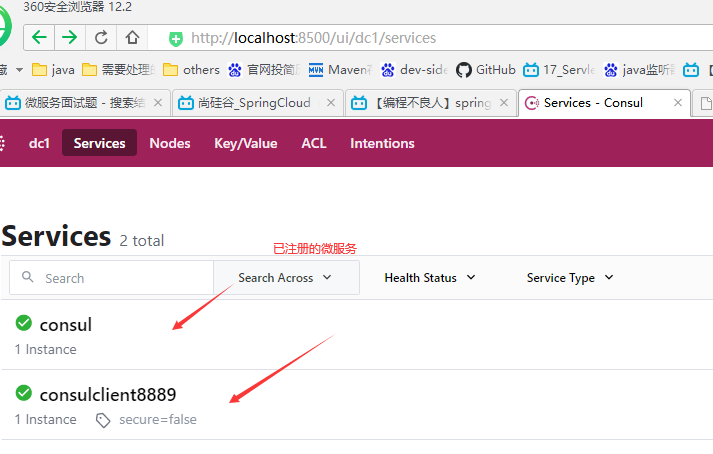
1. 可以通过访问端口的方式查看注册详情



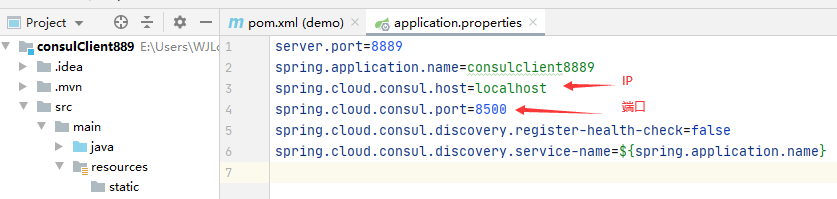
1. 当网络出出现波动时，可能心跳检测不会得到回应，如果没有自我保护机制的情况下，服务会从注册中心移除，自我保护机制在回应不低于85%情况下，不会移除服务，不建议关闭自我保护机制。
2. **Consul（具体见笔记**）
3. Consul是用go语言写的注册中心软件，所以只需要从官网下载下来，启动即可，不需要手写注册中心。而且也可以通过端口访问。







1. 微服务只需要在配置文件中加入注册中心地址即可注册



1. Consul使用健康监控，对心跳检测异常的微服务给以提示。
2. 面试题：不同注册中心区别
3. CAP原理：

**一致性（C）**：注册集群中的各个节点，同一数据必须保持一致才可用，这叫保持数据一致性

**可用性（A）：**注册集群中，即使剩余一个节点，该节点仍然可用，这叫可用性

**分区容忍性（P）：**少部分节点崩了，并不影响其他节点的使用

1. 各个注册中心区别

* 一致性：

**Eureka**并没有强制一性算法，而是通过复制尽量保持各节点一致

**Consul** Raft算法强一致性算法，选出一个leader节点承担所有工作，leader节点上所有改动过的数据必须同步到子节点，经子节点反馈，数据才可用

**Zookeeper** zab协议，所有的节点都是可以注册的，任意一个节点修改后，数据会通知给master节点，然后通知给所有节点同步更新，所有节点确认后，数据才可用。

* **可用性：**

**Consul** 和**Zookeeper** 为了保证各节点数据的统一，当绝大部分节点崩掉后，将不提供服务

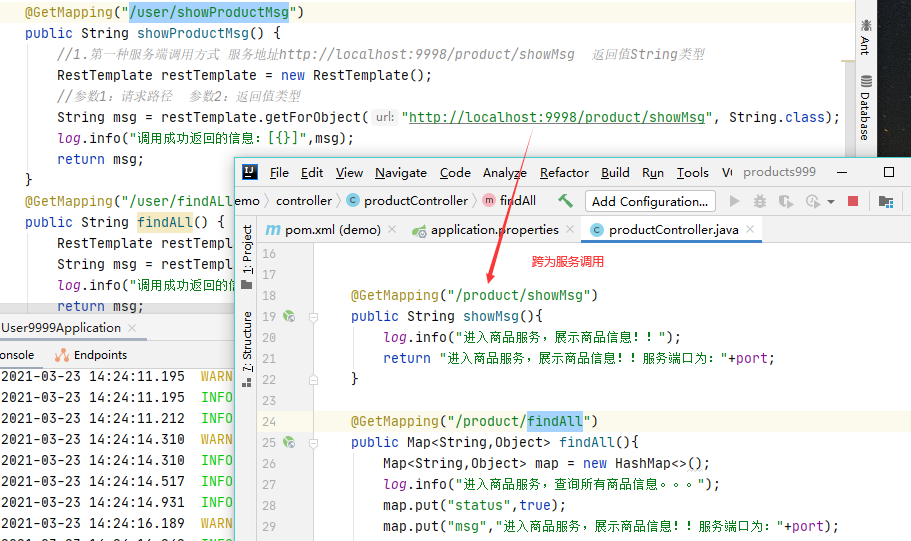
* **分区容忍性**

都具有一定的抗干扰能力。

1. 所以：**Eureka是AP，Consul** 和**Zookeeper是CP**

**组件二：服务之间的调用**

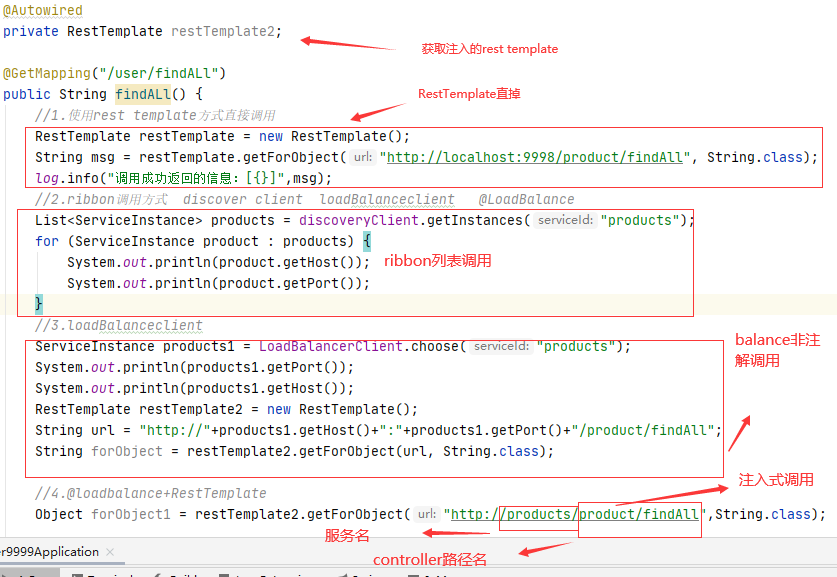
1. Restemplate类，微服务之间不经过注册中心，采取点对点的直接调取方式，当某业务类型成微服务集群时，无法完成负载均衡，不建议使用
2. 使用方法：Restemplate.getForObject(“IP+端口+Controller路径”,返回值类型)，还可以是postForObject、putForObject,来获取其他微服务控制层返回值。



1. Ribbon是根据服务名，在注册中心中获取服务列表，并在客户端形成缓存，默认提供轮询算法的负载均衡组件，与Restemplate结合使用，获取其他微服务的值。
2. 最常见的使用方式：配置文件中@loadBalanced修饰并注入Restemplate，这样Restemplate根据服务名+controller路径，就可以完成负载均衡式访问







1. OpenFeign组件（又叫Feign）,是对ribbon+RestTemplate组合的封装，以简化微服务之间的调用，所以Feign依旧沿用着ribbon的负载均衡策略。
2. Feign使用：以@FeignClient+服务名声明并注入接口，表示对接某类集群的微服务，以@requestMapping+路径的形式表示对接微服务中哪个路径，其他类用@autowired调用，即可完成对其他微服务集群访问，默认轮询。

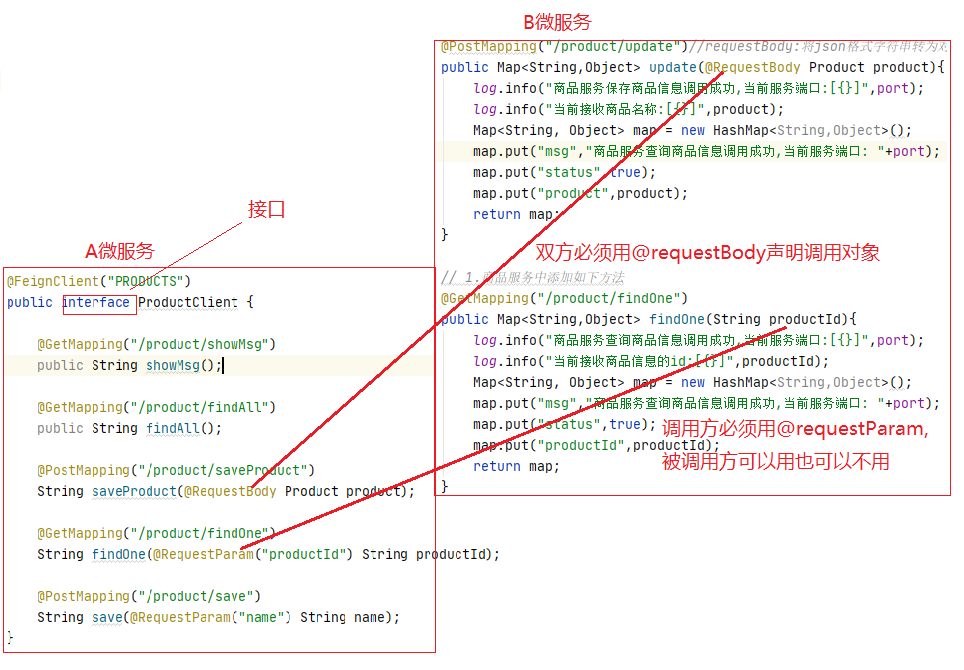




1. Feign传值：

**对于单一属性**：连接服务的接口必须用**@RequestParam**声明对应微服务方法中的形参，被调用方最好也用**@RequestParam**来修饰形参

**对于实体类型：**连接服务的接口和被调用方都必须用**@RequestBody**来修饰形参，**@RequestBody**可以将传来的json转换为对象类型





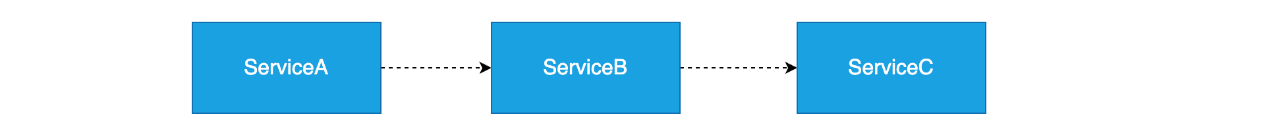
1. Feign默认一秒收不到微服务回复就会报错，所以配置文件中需要修改相应时间

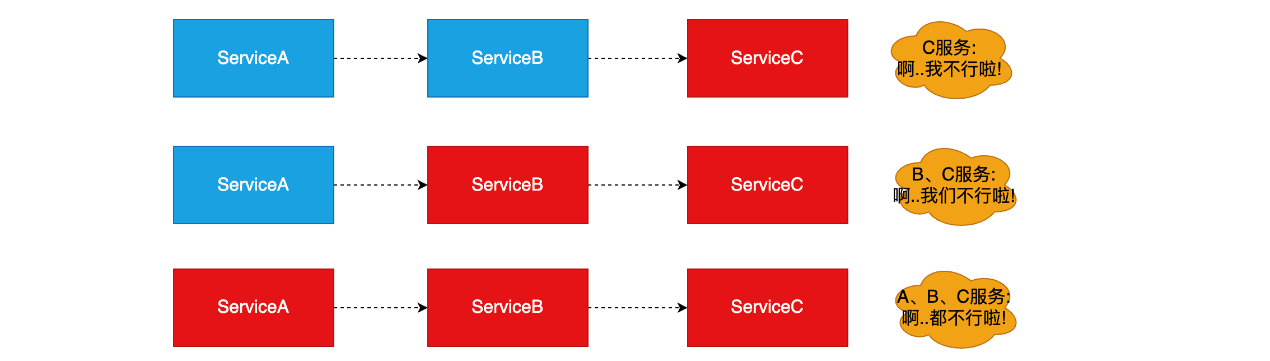


**组件三：****Hystrix熔断器（面试重点）**

1. **服务雪崩**：
2. 什么是服务雪崩？

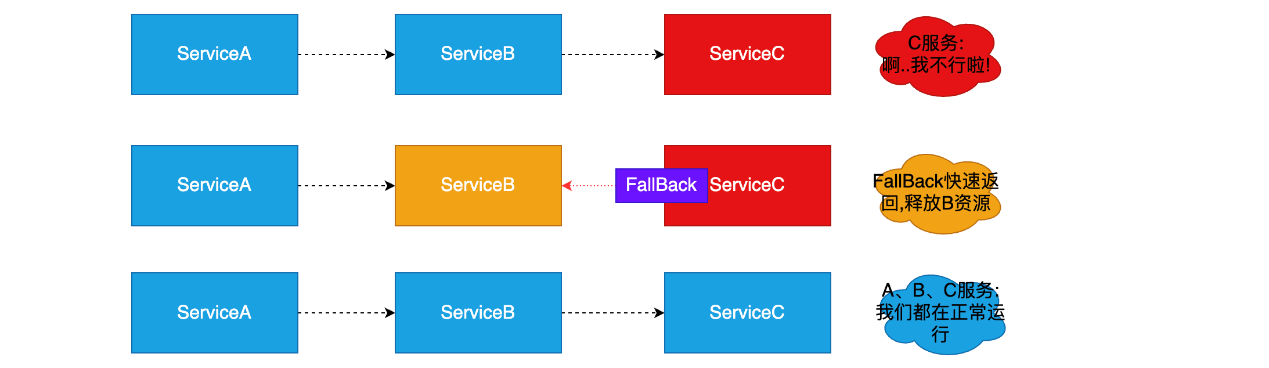
微服务日常调用中，通常会有调用链路，例如A调用B，B调用C，当有大量请求发来时，如果C处理业务较为复杂，反应时间很慢，就会有大量的线程堆在C的线程池中，由于springboot内嵌的tomcat，中线程池是有上限的，当超过上限就会发生故障，进而导致B中大量线程未得到响应堆在线程池中，进而导致B的故障，又导致A的故障，这种现象叫做服务雪崩。



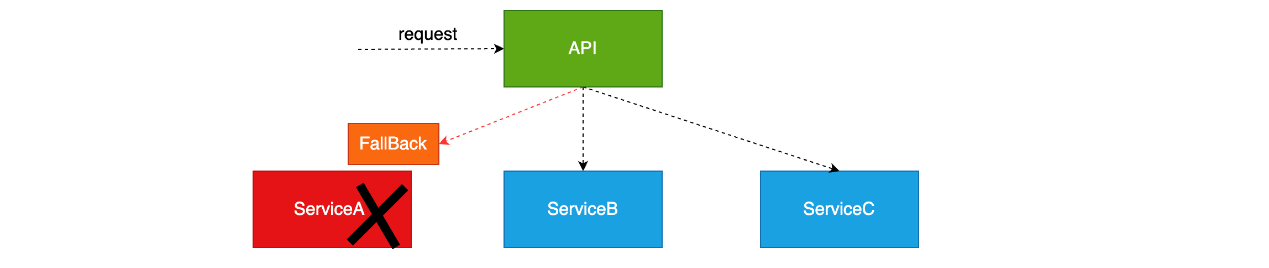


1. 服务熔断
2. 什么是服务熔断

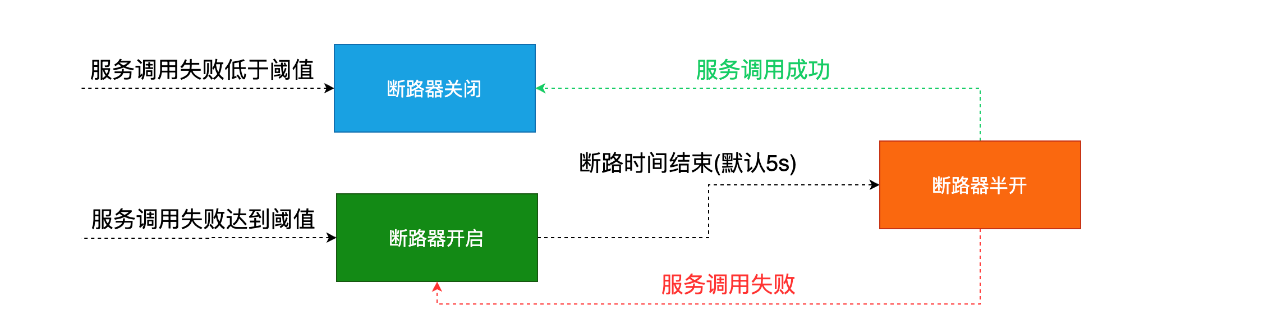
熔断器会对微服务中进行线程监测，某个异常条件被触发，直接熔断整个服务。向调用方法返回一个符合预期的、可处理的备选响应(FallBack),而不是长时间的等待或者抛出调用方法无法处理的异常，就保证了服务调用方的线程不会被长时间占用，避免故障在分布式系统中蔓延，乃至雪崩。如果目标服务情况好转则恢复调用。服务熔断是解决服务雪崩的重要手段。



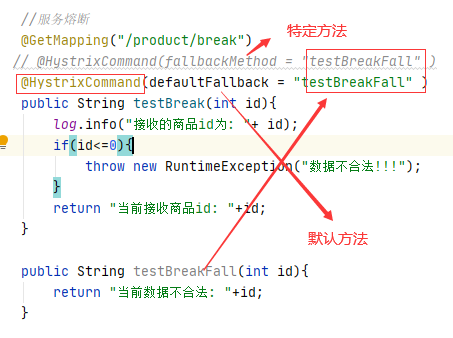
1. 服务降级
2. 服务器压力剧增的时候，手动关掉不重要的服务，来缓解服务器压力，当用户访问这些停掉的服务时，会有默认的返回，例如双十一，淘宝关闭了修改地址等服务，如果有人访问，将提示系统正忙。



1. 总结：熔断必会触发降级,所以熔断也是降级一种,区别在于熔断是对调用链路的保护,而降级是对系统过载的一种保护处理
2. **Hystrix熔断器原理**：当单位时间内，触发异常的请求（达到某个条件，一般通过throw new exception 方式抛出）达到某个值，或者 错误率达到某个值，熔断器就会被打开，打开之后，无论什么请求都不会进入该方法，而是返回预先设定好的对象，然后过一段时间（默认5秒）会放行一个请求，如果再次触发异常，熔断器继续保持打开状态，再过相同时间，再放行请求，直到不触发异常，熔断器关闭为止。



1. **Hystrix熔断器 熔断实现，**在要熔断的方法前加@HystrixCommand可以指定默认的返回方法defaultFallback，也可以指定特定的返回方法fallbackMethod。



1. **Hystrix熔断器 降级实现**

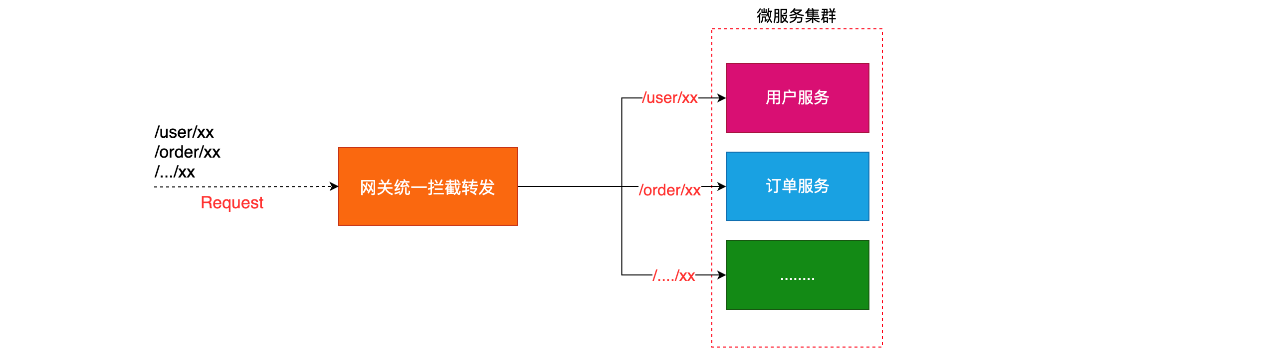
降级一般实现在请求方，结合feign来实现，我们在写微服务之间通信时会写FeignClient接口来完成通信，降级的实现就是对该接口实现，并注入到spring中，然后用fallback参数来指定该实现类。当被请求的服务关闭来节约电脑资源时，请求放就会返回实现类中的值来完成降级。

被请求方的降级一般表现为熔断



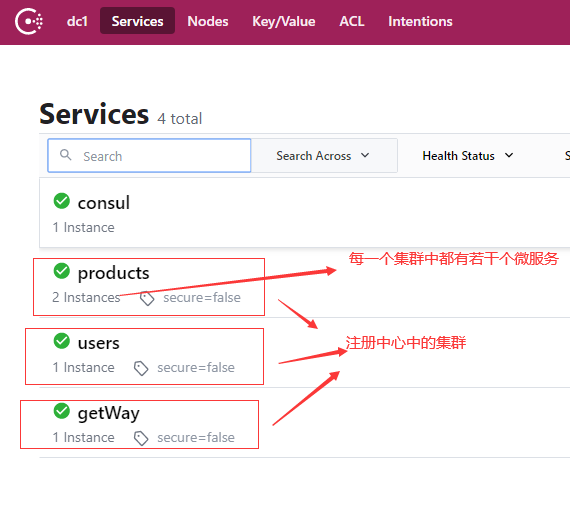
**组件五：服务网关****gateWay**

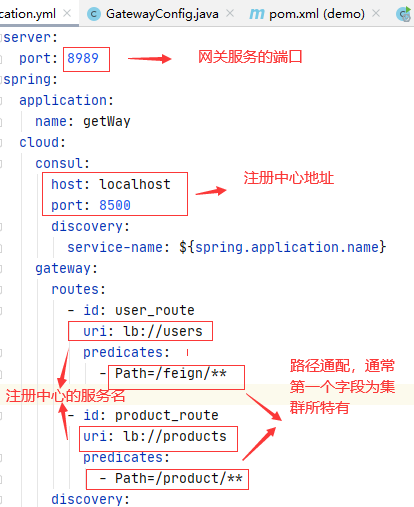
1. Zuul已经逐渐被gateWay替代。
2. 注册中心一般是用来支持微服务间的通信的组件，但是我们访问某个服务集群时，不可能记住每一个集群每个节点的IP和端口，这就造成了很大的不便。服务网关可以对集群节点进行统一管理，进行路由转发的组件，同时还兼备用户身份、权限验证等功能



1. **服务网关的实现：**做一个portal项目，然后在YML文件中进行路由规则的设置，通常每个微服务路径的第一字段为和服务相关的字段，所以路由规则一般为，当访问路径为某字段开头时，就去注册中心访问对应服务集群，所用的负载均衡也是注册中心提供的，然后访问到对应集群的某个节点上。我们只需要访问服务网关的IP+端口和对应微服务的路径即可。

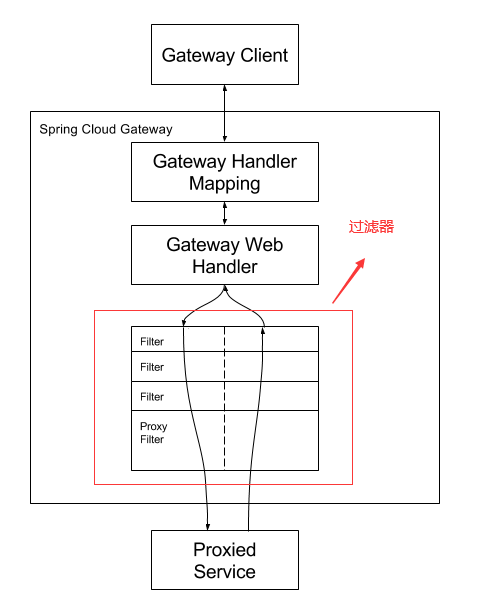
**如**用户集群我们一般是：/user/save、/user/findOne商品集群：/products/save、/products/list





如上图，我们在访问<http://localhost:8989/feign/showProductMsg>时，就会自动去注册中心中根据负载均衡算法，访问users集群中的某个微服务。

1. 当然路由规则也可以用代码的形式配置，不建议使用
2. **Filter**和springBoot中过滤器基本一致，是对请求中参数的处理，一般包括参数的增减修改，用户身份权限的认证等



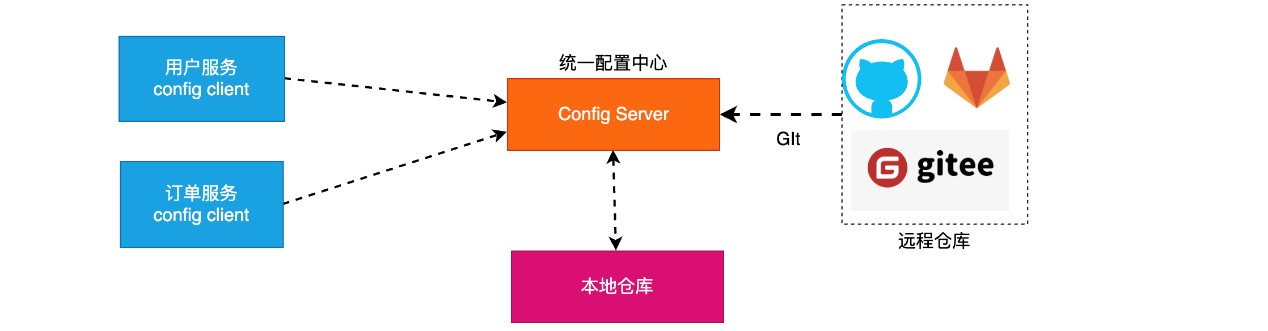
**gateWay**一般不使用默认的，使用自定义的，，在网关服务中进行配置，具体发生在注册中心和被调微服务之间，到达微服务前和请求传回之后都可以进行操作。



**组件六配置中心config\_server**

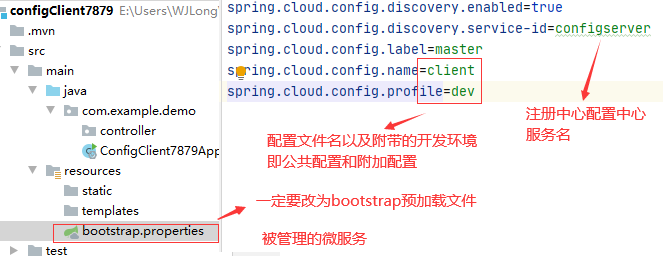
1. 当微服务集群越来越大，节点越来越多，配置文件也越来越多，很有可能我们修改其中的一个配置文件，其他微服务也收到影响，维护起来特别麻烦。

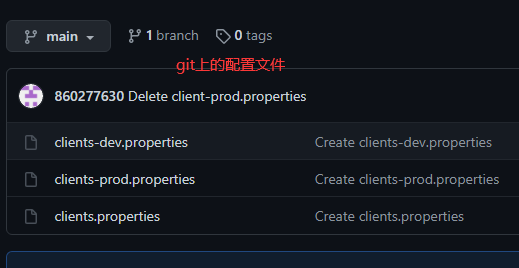
配置中心可以统一管理配置文件，方便检测维护，具体原理：将配置文件上传至远程仓库（git），本地仓库会有策略的有计划的更新来自远程仓库的文件，也可以手动更新，微服务启动时将会向配置中心索要配置文件，如果远程加载失败则从本地加载，每次修改只需要更新远程仓库，配置文件会自动更新至每个微服务中。



1. 具体的实现方式是：创建配置中心的微服务，然后配置文件中指定git的地址及身份信息，并配置本地目录，被管理的微服务配置文件都会更新保存在该目录中，被管理的微服务原先的配置文件上传至git仓库，配置文件以bootstrap命名作为预加载文件，内部指定注册中心中配置中心服务名和加载的配置文件即可，微服务启动时会自动的从注册中心根据服务名找到配置中心指定的目录加载文件。







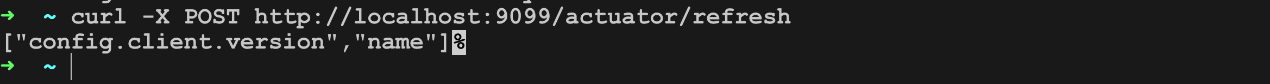
1. 当然我们也可以以配置中心服务IP+端口+公共配置文件-开发环境 的形式进行预览，可以在json、yml、properties各种格式转换，并且可以获得到公共和私有所有的配置。

例如：localhost:7878/clients-dev.yml 将获得开发环境的所有配置



1. 手动刷新（不常用了解一下，被下面的bus总线替换）

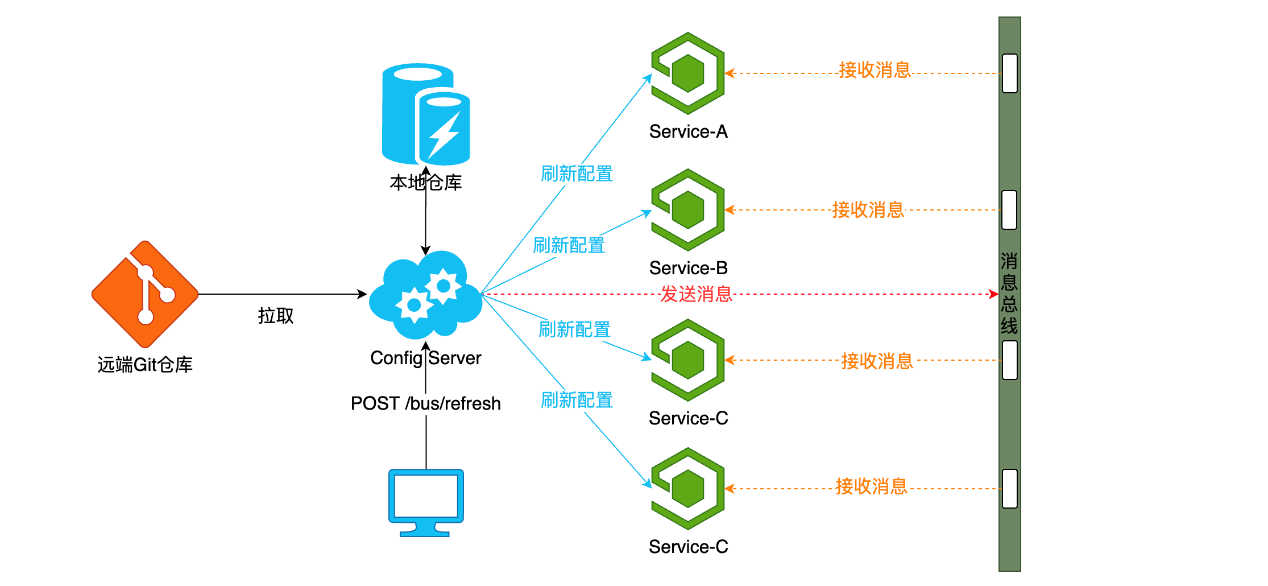
我们在更新git上配置后，不可能每更新一次就重启一下配置中心，官方为我们提供了一个激活接口，在linux下，向要更新配置的微服务发送特定的post请求（curl -X POST <http://localhost>:端口/actuator/refresh），就会通过注册中心，向配置中心发送信号，去更新git上的配置，该微服务节点自动加载更新后的配置文件。



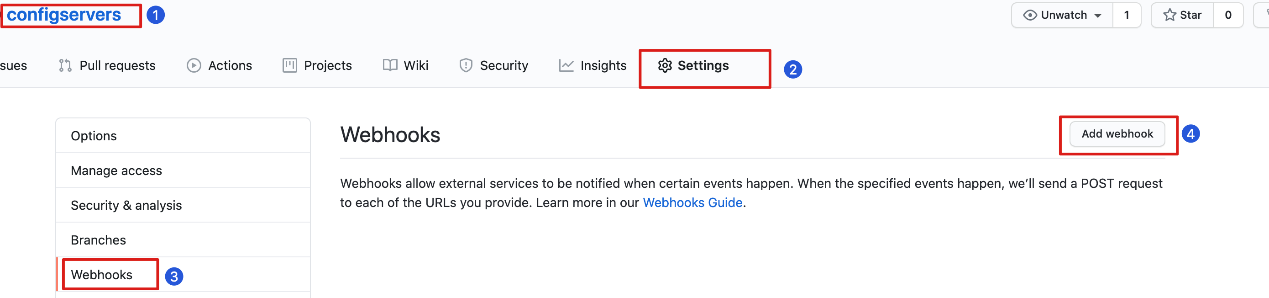
**组件七，BUS消息总线**

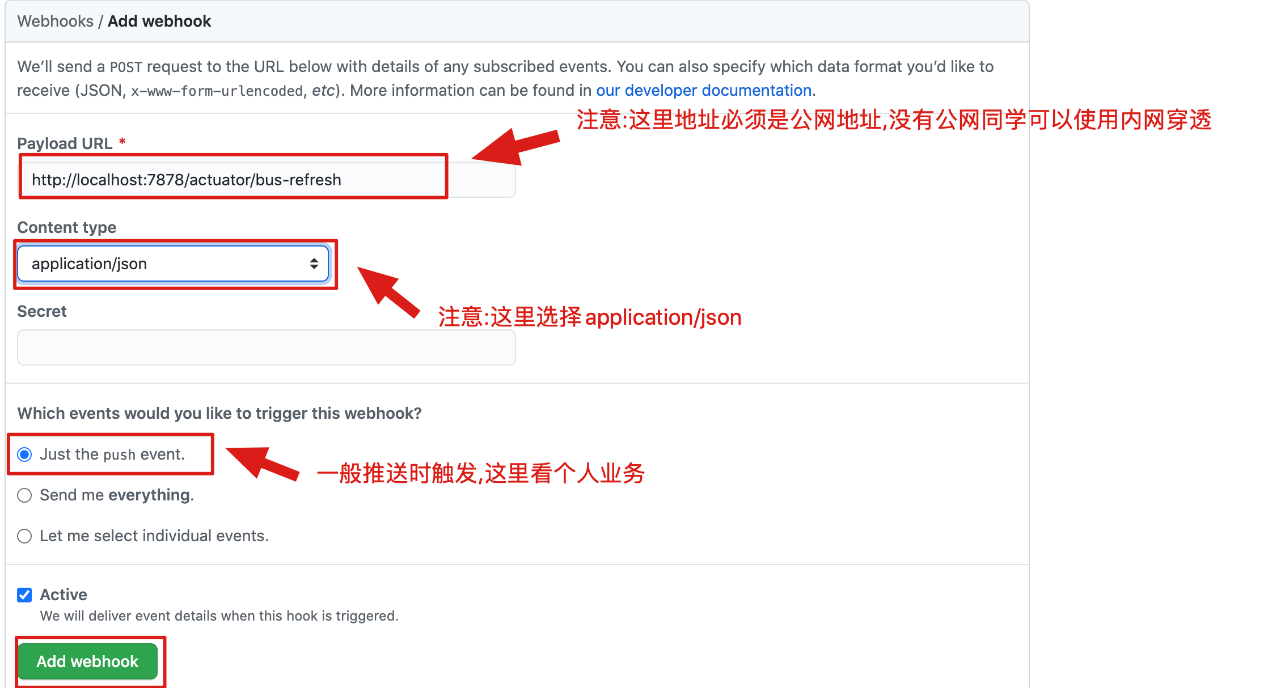
1. Git配置文件更新后，为了及时的更新到集群中的每个节点，一些笨办法重启配置中心微服务或者对每个节点都发送post请求，这样很明显是很繁琐，甚至集群特别大时根本行不通。

Bus组件就是解决这一问题，自动集成MQ和配置中心，配置中心以及微服务都连接MQ消息中心Bus创建的消息队列来供微服务和配置中心的通信，当有特定的post请求直接访问配置中心，配置中心会更新配置文件，然后bus组件发送消息导消息队列，通知各个微服务更新配置文件。



当然也可以在git上集成webhook，当配置文件更新时就会自动发请求配置中心，通知各个节点更新配置文件。

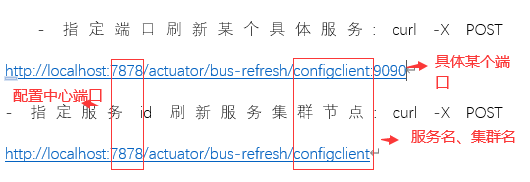




Post请求 linux下

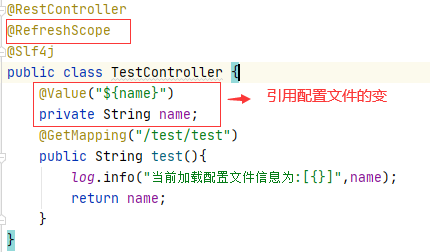
- 指定端口刷新某个具体服务: curl -X POST <http://localhost:7878/actuator/bus-refresh/configclient:9090>

- 指定服务id刷新服务集群节点: curl -X POST <http://localhost:7878/actuator/bus-refresh/configclient>





1. 如果微服务中或配置中心引用了远程配置文件的变量，则需要类前添加@RefreshScope，不然都取最新改变的值。



1. Git的Webhook功能只能针对外网，所以内网开发时是不可能使用的，面试的时候一定要加被指中心的话，可以是内网开发用post请求，上线之后用的是webhooks