spring cloud 周阳

spring cloud ， 基于spring boot提供了一套微服务解决方案， 包括5大神兽。 spring cloud 利用spring boot 的开发便利性巧妙的简化了分布式系统基础设施的开发。。。

springcloud = 分布式微服务架构下的一站式解决方案，是各个微服务架构落地技术的集合体，俗称微服务全家桶

spring cloud 和 dubbo 的区别？

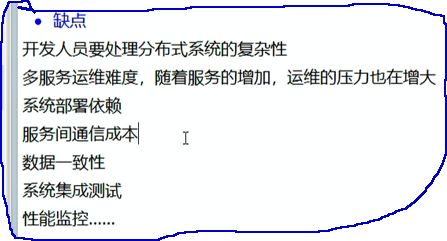
dubbo服务之间采用RPC调用， spring cloud 采用基于HTTP的RESTful API 调用

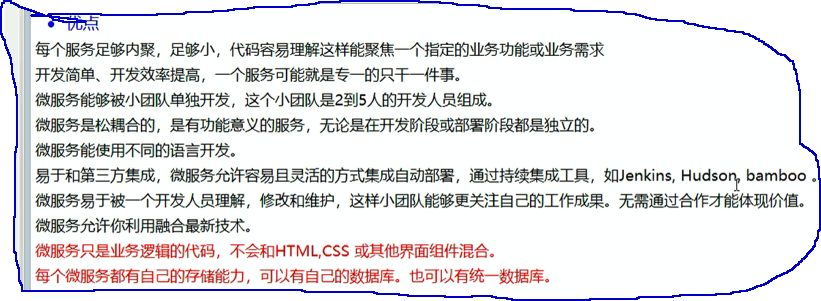
微服务和微服务架构的区别？

微服务强调的是服务的大小，它关注的是某一个点，

微服务架构是一种架构模式， 它提倡将单一的应用程序划分为一组小的服务，服务之间互相协调，互相配合，为用户提供最终价值。每个服务运行在其独立的进程中，服务与服务间基于HTTP协议的RESTful API的协议进行通信

微服务的优缺点？





SpringCloud和SpringBoot是什么关系？

SpringBoot专注于快速方便的开发单个个体微服务。

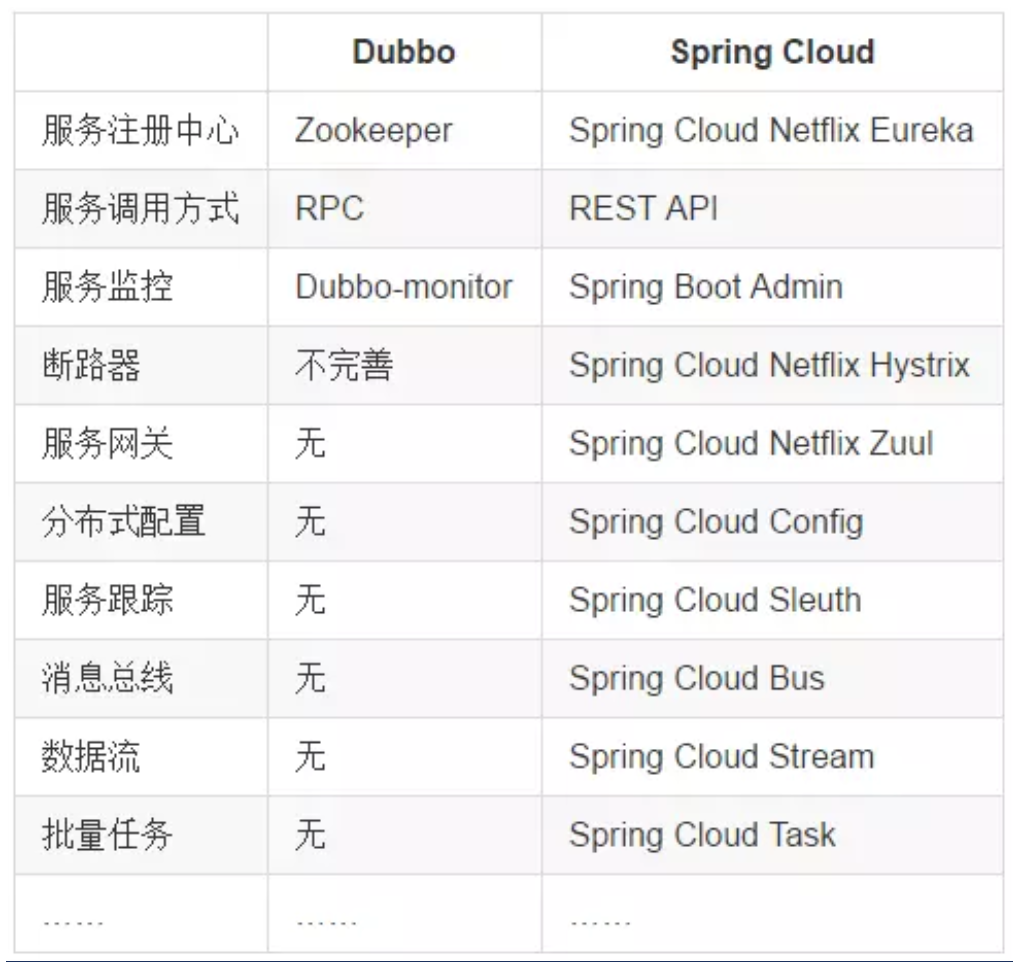
SpringCloud是关注全局的微服务协调整理治理框架，它将SpringBoot开发的一个个单体微服务整合并管理起来，为各个微服务之间提供，配置管理、服务发现、断路器、路由、微代理、事件总线、全局锁、决策竞选、分布式会话等等集成服务

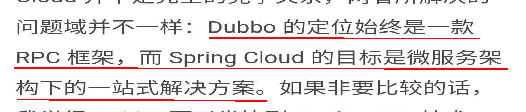
SpringBoot可以离开SpringCloud独立使用开发项目，但是SpringCloud离不开SpringBoot，属于依赖的关系.

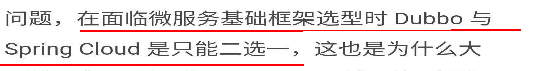
SpringBoot专注于快速、方便的开发单个微服务个体，SpringCloud关注全局的服务治理框架。

FASTDFS, 分布式文件系统， 我们明白， 小图片是不允许放在数据库中的， 必须放到分布式文件系统中。。。。。 do you understand?

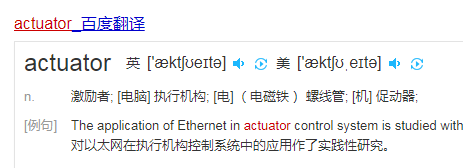
SpringCloud 和 DUBBO的对比

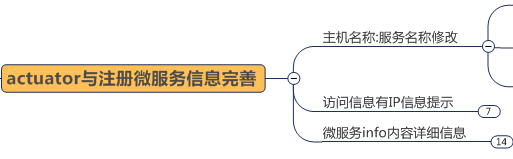






**Actuator**

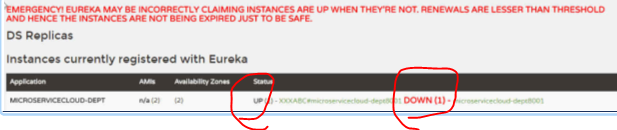




**Eureka的自我保护机制**

设计哲学就是宁可保留错误的服务注册信息，也不盲目注销任何可能健康的服务实例。一句话讲解：好死不如赖活着

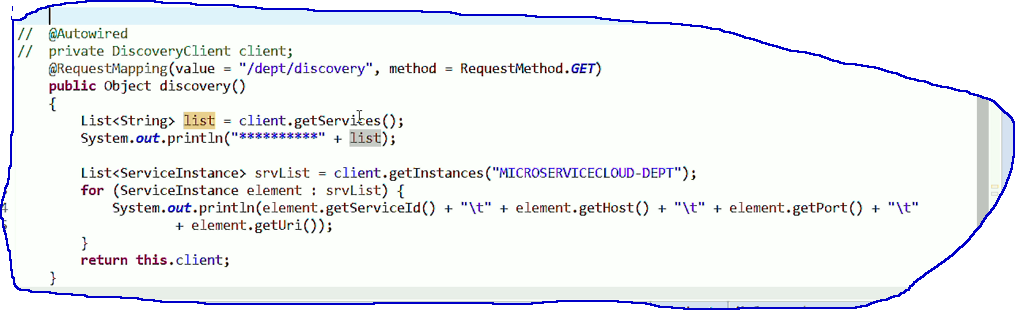
在Spring Cloud中，可以使用eureka.server.enable-self-preservation = false 禁用自我保护模式



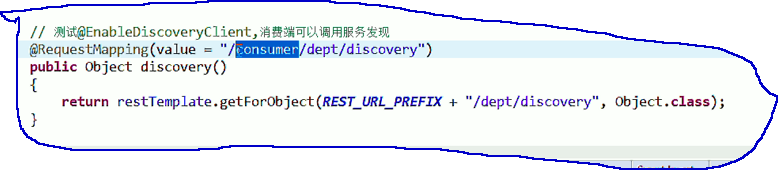
**Eureka 服务发现**

@EnableDiscoveryClient 服务发现，写在provider的主启动类上

Provider类上写下面的

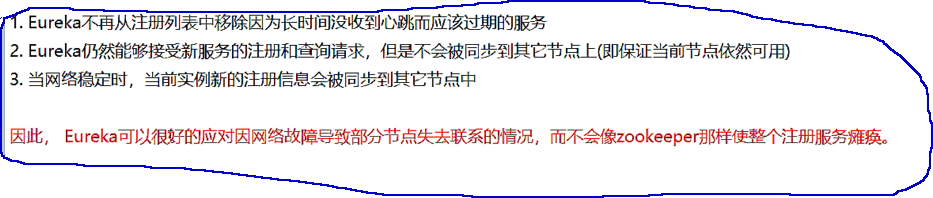


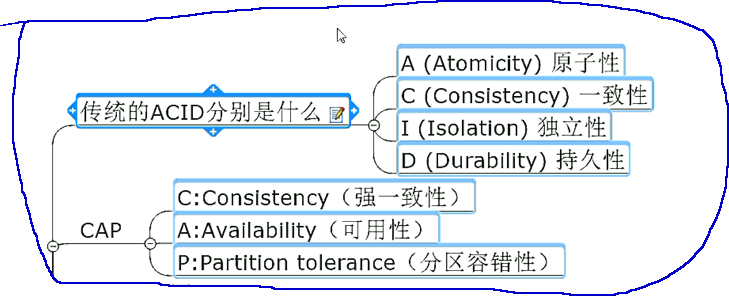
consumer类上写下面的

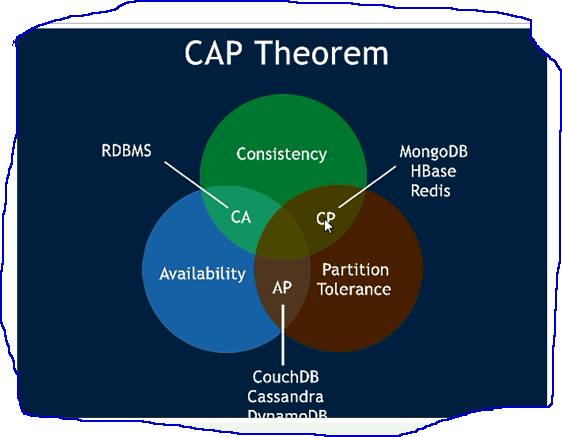


**Eureka比zookeeper好在哪里？**

1， Eureka 保证AP（心跳） ， zookeeper 保证CP(选举)







CAP的3进2 理论

**Ribbon**

Spring Cloud Ribbon是基于Netflix Ribbon实现的一套客户端 负载均衡的工具。

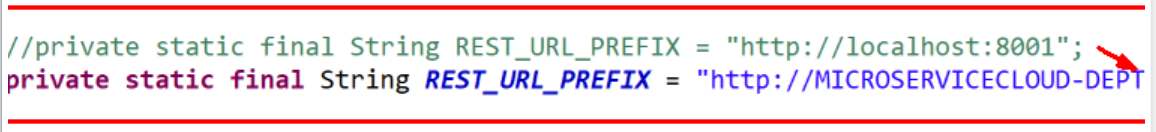
提供客户端的软件负载均衡算法， 轮询，权重， 取模，等算法。

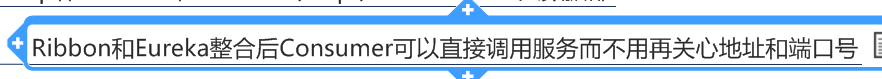
LB, 即负载均衡(Load Balance)， 分为2种

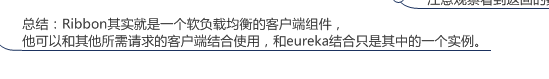
集中式(硬件): 硬件F5

进程内(软件)：Nginx, LVS, Ribbon

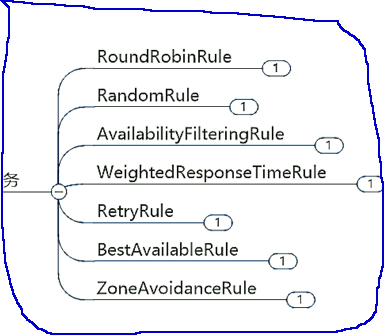
spring cloud的负载均衡的算法可以自定义

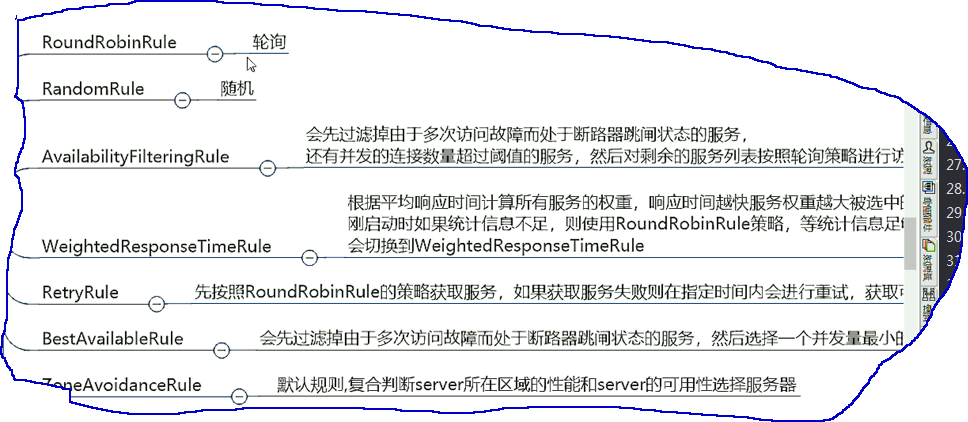






**Ribbon 的7中系统自带负载均衡算法**





**1.Ribbon 默认使用轮询的算法做负载均衡**

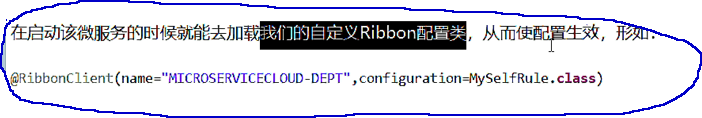
**2.Ribbon 可以更换自带的其他的算法做负载均衡，如下图：**



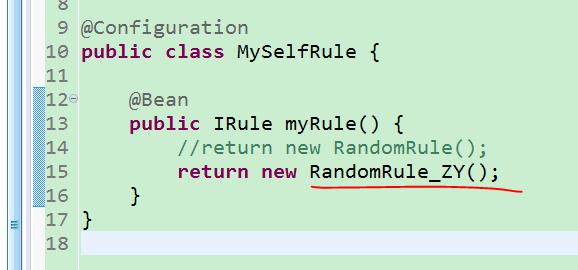
**3. Ribbon 也可以使用 我自己定义规则的算法来做负载均衡。 ①②③**

**4. 如果2，3 同时出现， 则实际使用的是3**

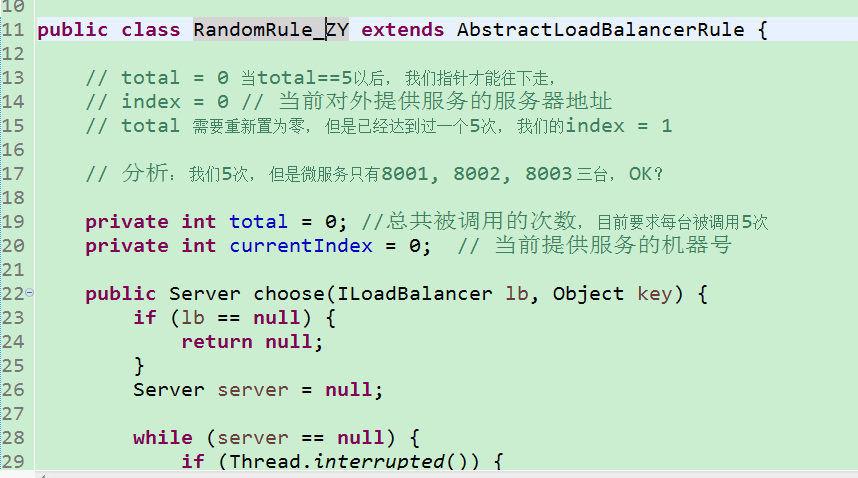
**①**



**②**



**③**



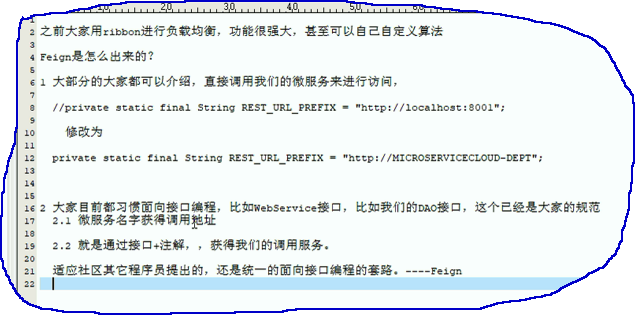
**！！！！！！！！！！！！！！！！！！**

**MySelfRule 这个类， 只能写在主启动类的上级目录里面。。。**

**Feign**

Feign， 是一个申明式WebService客户端， Feign可以与Eureka和Ribbon组合使用以支持负载均衡。

Feign， 旨在使编写Java Http 客户端变得更加容易， 只需要创建一个接口，然后在上面添加注解即可。



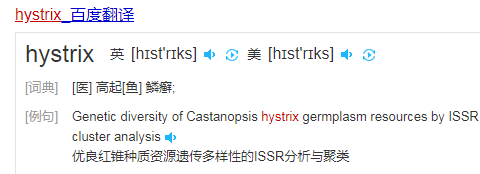
Feign通过接口的方法调用Rest服务（之前是Ribbon+RestTemplate），

该请求发送给Eureka服务器（http://MICROSERVICECLOUD-DEPT/dept/list）,

通过Feign直接找到服务接口，由于在进行服务调用的时候融合了Ribbon技术，所以也支持负载均衡作用。

**Hystrix断路器**

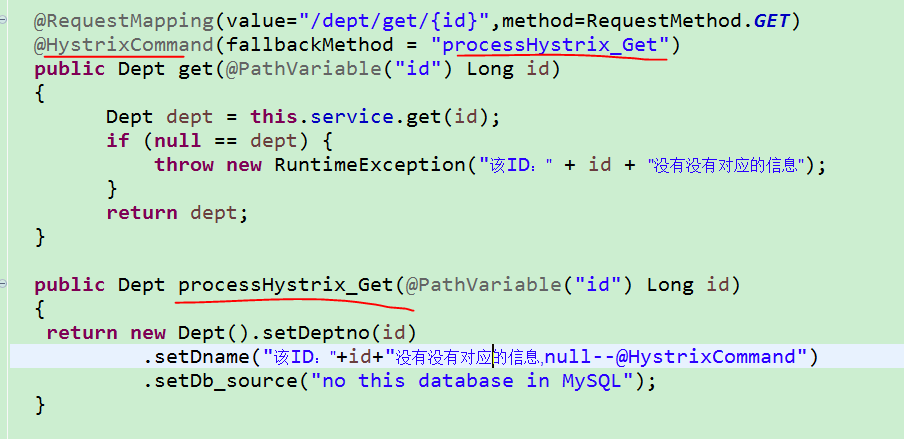
类似熔断保险丝），向调用方返回一个符合预期的、可处理的备选响应（FallBack），而不是长时间的等待或者抛出调用方无法处理的异常，这样就保证了服务调用方的线程不会被长时间、不必要地占用，从而避免了故障在分布式系统中的蔓延，乃至雪崩。



**服务熔断**

@HystrixCommand报异常后如何处理

@EnableCircuitBreaker 主启动类DeptProvider8001\_Hystrix\_App并添加新注解

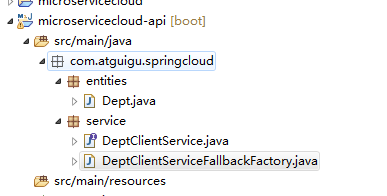


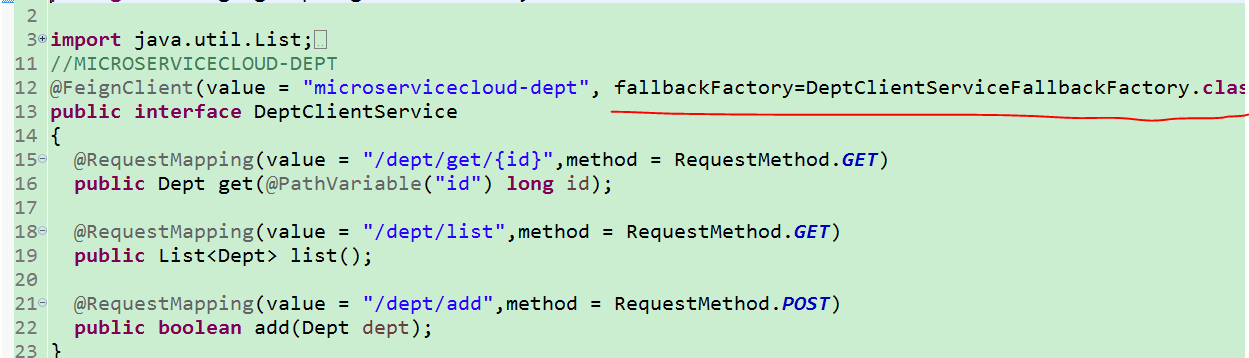
有点像SPRING AOP 的 AfterThrowing....

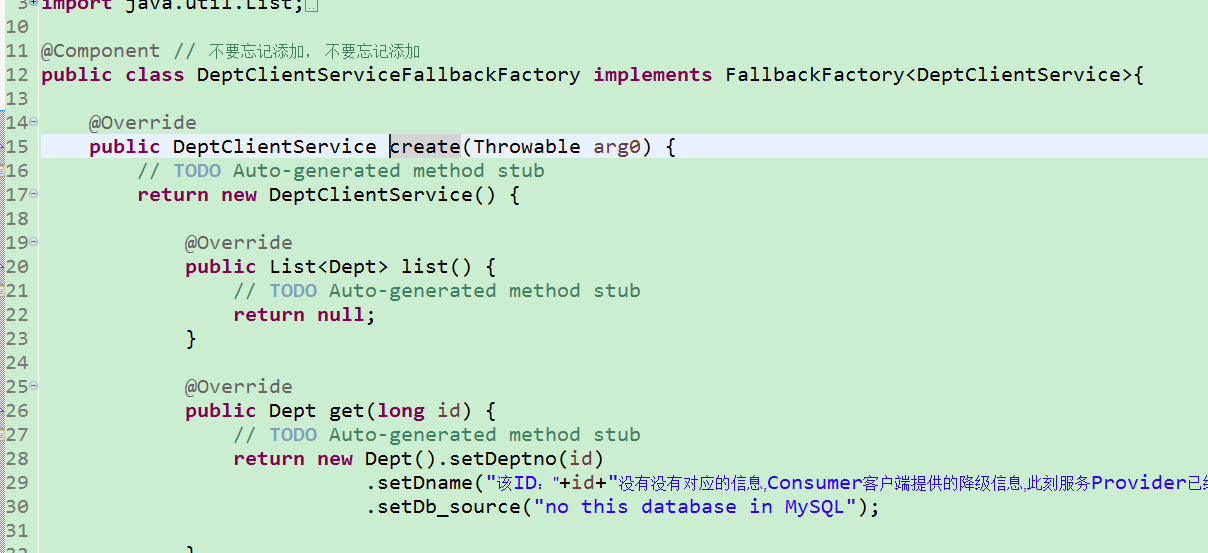
**服务降级**

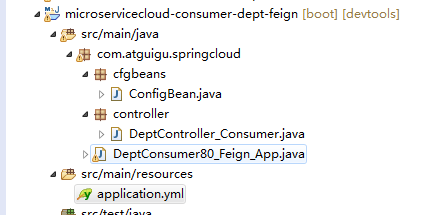
整体资源快不够了，忍痛将某些服务先关掉，待渡过难关，再开启回来。

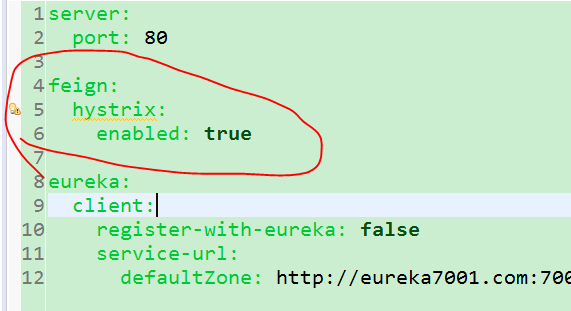
服务降级处理是在客户端处理的， 和服务端没有关系







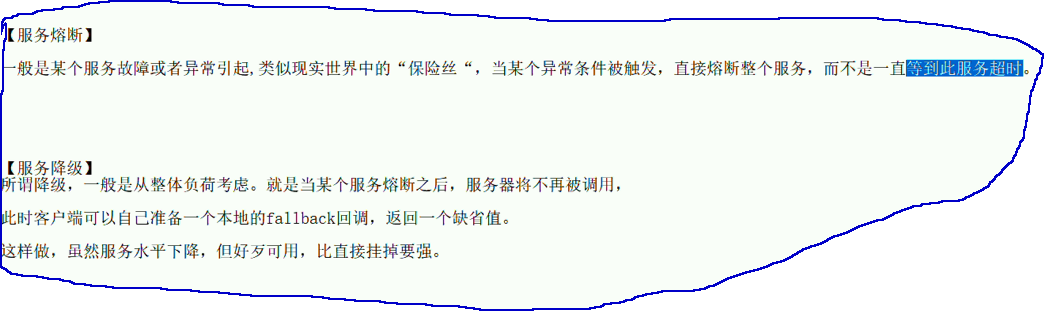




服务降级和服务熔断的区别？

服务降级与业务代码有严重的耦合， 并且随着业务需求的增加， 要增加的代码会不断的增加

服务熔断， 面向接口的， 和业务代码分离， 1处定义多处使用。 就是spring aop的。。

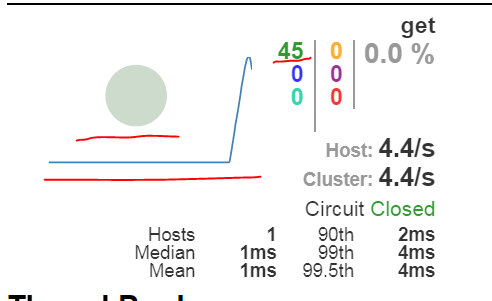


**hystrixDashboard**



实心圆 它的健康度从绿色<黄色<橙色<红色递减。 流量越大该实心圆就越大

曲线：用来记录2分钟内流量的相对变化，可以通过它来观察到流量的上升和下降趋势。



**Zuul**

提供=代理+路由+过滤三大功能

Zuul服务最终还是会注册进Eureka

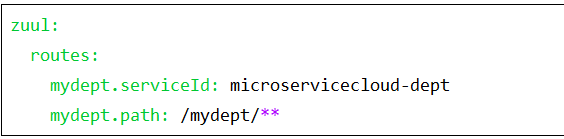
Zuul包含了对请求的路由和过滤两个最主要的功能：

其中路由功能负责将外部请求转发到具体的微服务实例上，是实现外部访问统一入口的基础而过滤器功能则负责对请求的处理过程进行干预，是实现请求校验、服务聚合等功能的基础.

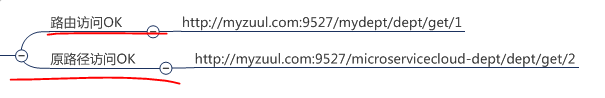
Zuul和Eureka进行整合，将Zuul自身注册为Eureka服务治理下的应用，同时从Eureka中获得其他微服务的消息，也即以后的访问微服务都是通过Zuul跳转后获得。



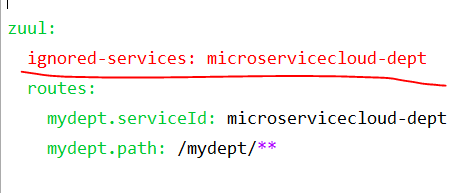
**路由访问映射规则**



但是存在下面的问题，



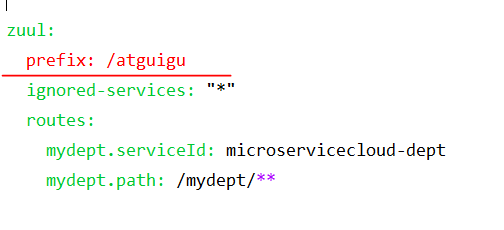
解决如下：



如果要隐藏所有的原路径访问，

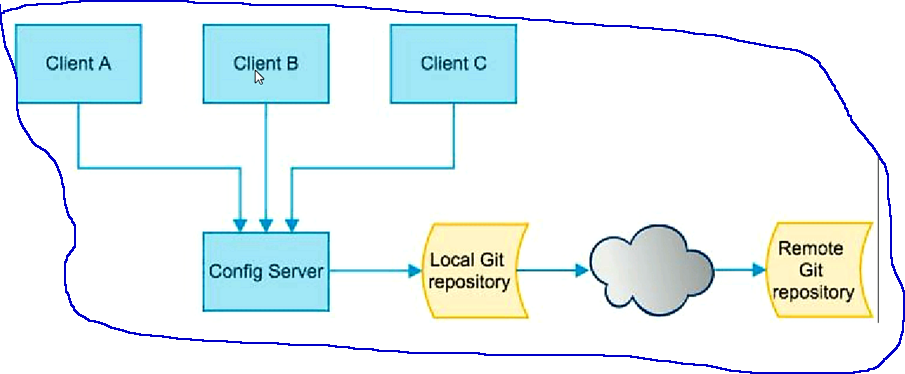
ignored-services: microservicecloud-dept

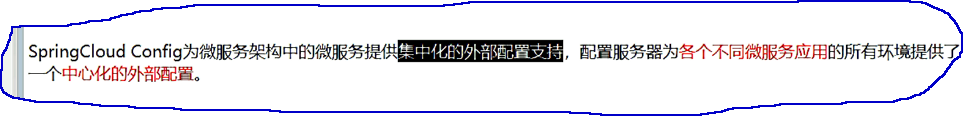
**设置统一公共前缀**

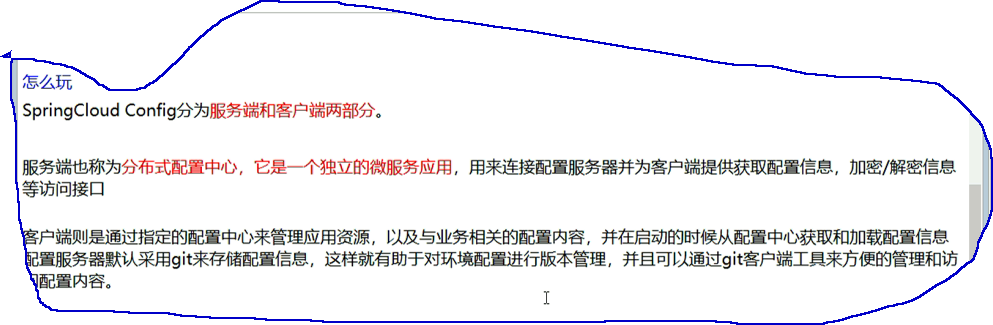


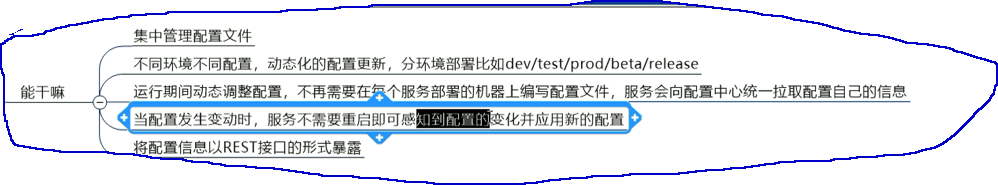
<http://myzuul.com:9527/atguigu/mydept/dept/get/1>

**SpringCloud\_Config**



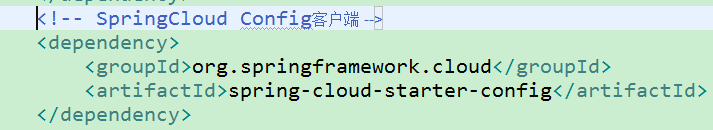








**Springcloud-config Client端**



application.yml 是用户级的资源配置项

bootstrap.yml 是 系统级的，优先级更加高的

