# Spring Cloud

### 一、Spring Cloud简介

Spring Cloud 为开发人员提供了快速构建分布式系统的一些工具，包括配置管理、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。

它运行环境简单，可以在开发人员的电脑上跑。

另外说明spring cloud是基于springboot的，所以需要开发中对springboot有一定的了解。

另外对于“微服务架构” 不了解的话，可以通过搜索引擎搜索“微服务架构”了解下。

### 二、Ribbon

在上一篇文章，讲了服务的注册和发现。

在微服务架构中，业务都会被拆分成一个独立的服务，服务与服务的通讯是基于http restful的。

Spring cloud有两种服务调用方式，一种是ribbon+restTemplate，另一种是feign。在这一篇文章首先讲解下基于ribbon+rest。

### ribbon简介

Ribbon is a client side load balancer which gives you a lot of control over the behaviour of HTTP and TCP clients. Feign already uses Ribbon, so if you are using @FeignClient then this section also applies.

—–摘自官网

ribbon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。Feign默认集成了ribbon。

ribbon 已经默认实现了这些配置bean：

IClientConfig ribbonClientConfig: DefaultClientConfigImpl

IRule ribbonRule: ZoneAvoidanceRule

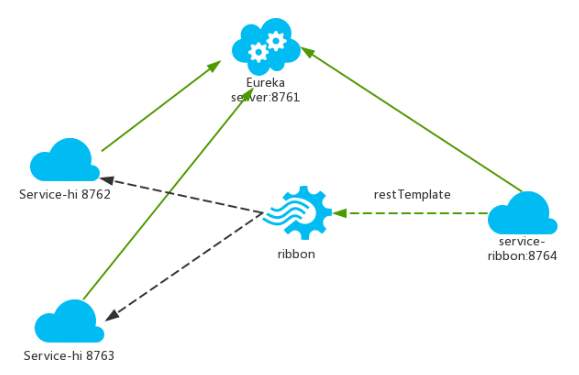
IPing ribbonPing: NoOpPing

ServerList ribbonServerList: ConfigurationBasedServerList

ServerListFilter ribbonServerListFilter: ZonePreferenceServerListFilter

ILoadBalancer ribbonLoadBalancer: ZoneAwareLoadBalancer

### 此时的架构



* 一个服务注册中心，eureka server,端口为8761
* service-hi工程跑了两个实例，端口分别为8762,8763，分别向服务注册中心注册
* sercvice-ribbon端口为8764,向服务注册中心注册
* 当sercvice-ribbon通过restTemplate调用service-hi的hi接口时，因为用ribbon进行了负载均衡，会轮流的调用service-hi：8762和8763 两个端口的hi接口；

### 三、Feign

Feign简介

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。

简而言之：

Feign 采用的是基于接口的注解

Feign 整合了ribbon

### 四、Hystrix

在微服务架构中，根据业务来拆分成一个个的服务，服务与服务之间可以相互调用（RPC），在Spring Cloud可以用RestTemplate+Ribbon和Feign来调用。

为了保证其高可用，单个服务通常会集群部署。

由于网络原因或者自身的原因，服务并不能保证100%可用，如果单个服务出现问题，调用这个服务就会出现线程阻塞，此时若有大量的请求涌入，Servlet容器的线程资源会被消耗完毕，导致服务瘫痪。

服务与服务之间的依赖性，故障会传播，会对整个微服务系统造成灾难性的严重后果，这就是服务故障的“雪崩”效应。

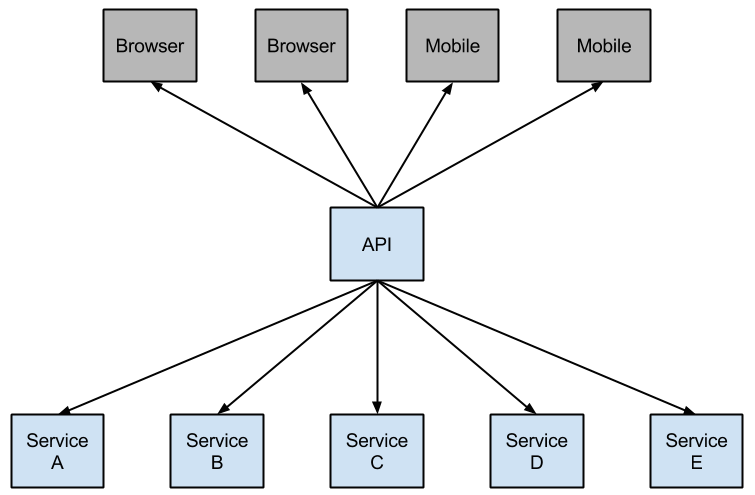
为了解决这个问题，业界提出了断路器模型。

一、断路器简介

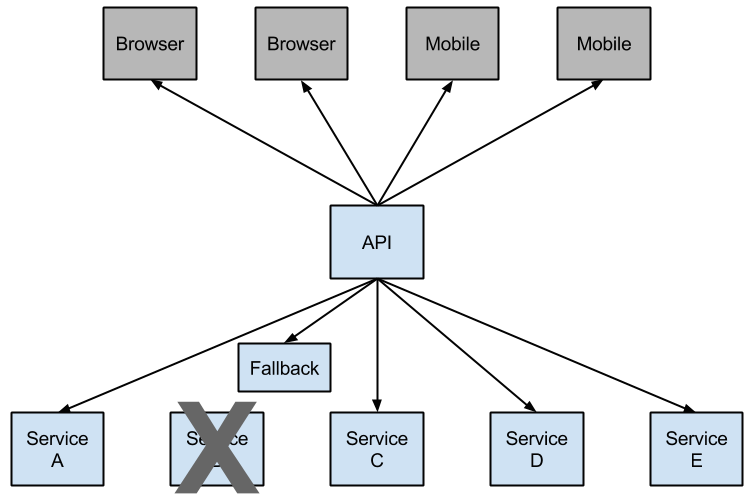
Netflix has created a library called Hystrix that implements the circuit breaker pattern. In a microservice architecture it is common to have multiple layers of service calls.

—-摘自官网

Netflix开源了Hystrix组件，实现了断路器模式，SpringCloud对这一组件进行了整合。 在微服务架构中，一个请求需要调用多个服务是非常常见的，如下图：



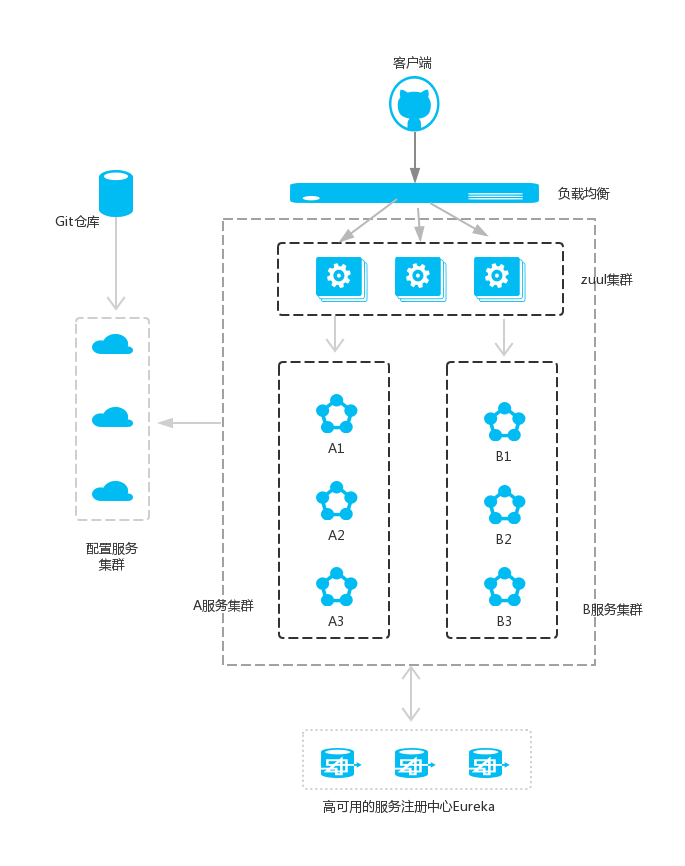
较底层的服务如果出现故障，会导致连锁故障。当对特定的服务的调用的不可用达到一个阀值（Hystric 是5秒20次） 断路器将会被打开。



断路打开后，可用避免连锁故障，fallback方法可以直接返回一个固定值。

### Zuul

在微服务架构中，需要几个基础的服务治理组件，包括服务注册与发现、服务消费、负载均衡、断路器、智能路由、配置管理等，由这几个基础组件相互协作，共同组建了一个简单的微服务系统。一个简答的微服务系统如下图：



**注意**：A服务和B服务是可以相互调用的，作图的时候忘记了。并且配置服务也是注册到服务注册中心的。

在Spring Cloud微服务系统中，一种常见的负载均衡方式是，客户端的请求首先经过负载均衡（zuul、Ngnix），再到达服务网关（zuul集群），然后再到具体的服。  
服务统一注册到高可用的服务注册中心集群，服务的所有的配置文件由配置服务管理（下一篇文章讲述），配置服务的配置文件放在git仓库，方便开发人员随时改配置。

**Zuul简介**

Zuul的主要功能是路由转发和过滤器。路由功能是微服务的一部分，比如／api/user转发到到user服务，/api/shop转发到到shop服务。zuul默认和Ribbon结合实现了负载均衡的功能。

zuul有以下功能：

Authentication

Insights

Stress Testing

Canary Testing

Dynamic Routing

Service Migration

Load Shedding

Security

Static Response handling

Active/Active traffic management

### config

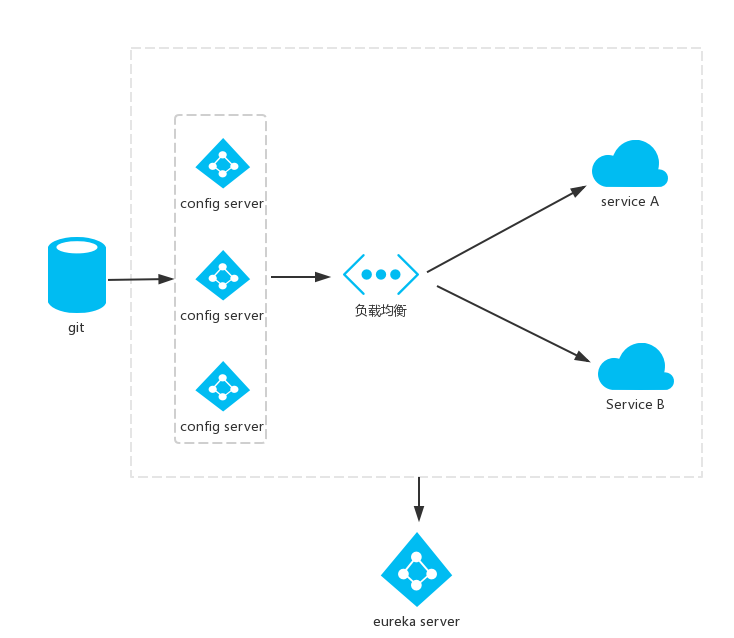
在分布式系统中，由于服务数量巨多，为了方便服务配置文件统一管理，实时更新，所以需要分布式配置中心组件。

在Spring Cloud中，有分布式配置中心组件spring cloud config ，它支持配置服务放在配置服务的内存中（即本地），也支持放在远程Git仓库中。

在spring cloud config 组件中，分两个角色，一是config server，二是config client。

### 高可用的分布式配置中心

上一篇文章讲述了一个服务如何从配置中心读取文件，配置中心如何从远程git读取配置文件，当服务实例很多时，都从配置中心读取文件，这时可以考虑将配置中心做成一个微服务，将其集群化，从而达到高可用，架构图如下：



### Spring Cloud Bus

Spring Cloud Bus 将分布式的节点用轻量的消息代理连接起来。

它可以用于广播配置文件的更改或者服务之间的通讯，也可以用于监控。

本文要讲述的是用Spring Cloud Bus实现通知微服务架构的配置文件的更改。