Spring Cloud

项目与Spring Cloud的结合方案

tjlcast - 2017年12月8日

Spring Cloud介绍

Spring Cloud是一系列框架的有序集合。它利用Spring Boot的开发便利性巧妙地简化了分布式系统基础设施的开发，如服务发现注册、配置中心、消息总线、负载均衡、断路器、数据监控等，都可以用Spring Boot的开发风格做到一键启动和部署。Spring并没有重复制造轮子，它只是将目前各家公司开发的比较成熟、经得起实际考验的服务框架组合起来，通过Spring Boot风格进行再封装屏蔽掉了复杂的配置和实现原理，最终给开发者留出了一套简单易懂、易部署和易维护的分布式系统开发工具包。

微服务是可以独立部署、水平扩展、独立访问（或者有独立的数据库）的服务单元，springcloud就是这些微服务的大管家，采用了微服务这种架构之后，项目的数量会非常多，springcloud做为大管家需要管理好这些微服务，自然需要很多模块来帮忙。

核心模块

* **Spring Cloud Netflix**

各项服务依赖与它，与各种Netflix OSS组件集成，组成微服务的核心，相关依赖的模块主要有Eureka, Hystrix, Zuul, Archaius…

* **Netflix Eureka**

服务中心，云端服务发现，一个基于 REST 的服务，用于定位服务，以实现云端中间层服务发现和故障转移。所有的微服务都会在这个模块中进行注册并发现其他微服务。

* **Netflix Hystrix**

熔断器，容错管理工具，旨在通过熔断机制控制服务和第三方库的节点,从而对延迟和故障提供更强大的容错能力。

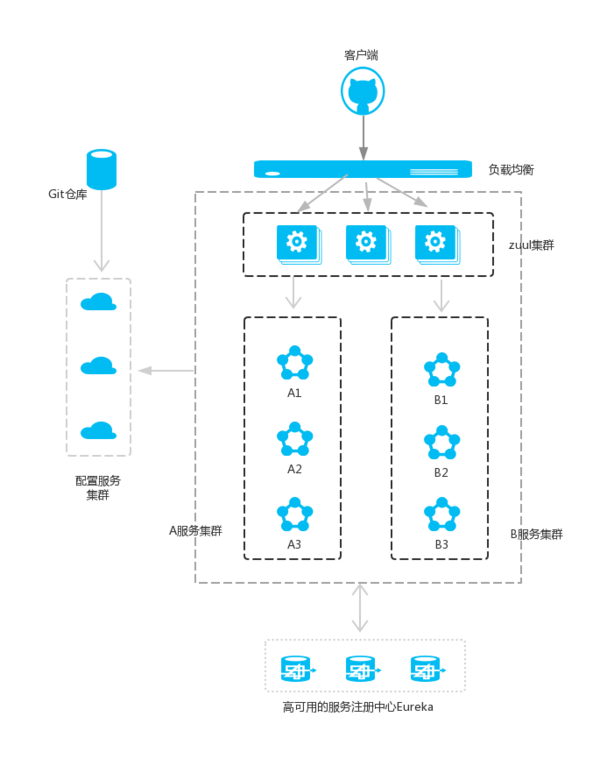
* **Netflix Zuul**

Zuul 是在云平台上提供动态路由,监控,弹性,安全等边缘服务的框架。Zuul 相当于是设备和 Netflix 流应用的 Web 网站后端所有请求的前门。

* **Spring Cloud Config**

俗称的配置中心，配置管理工具包，让你可以把配置放到远程服务器，集中化管理集群配置，目前支持本地存储、Git以及Subversion。

（这里的所有服务仅指以Application提供的服务，不包含设备服务）



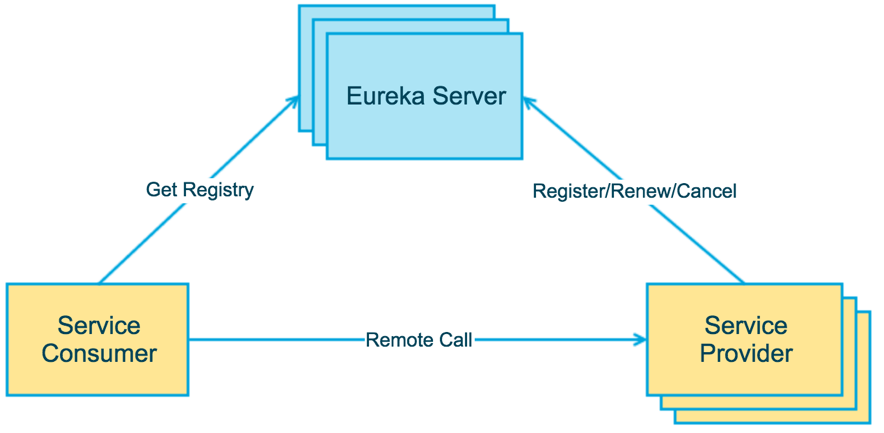
Spring Cloud的项目平台示意图

1. 服务注册中心
2. 介绍

在Spring Cloud中Eureka用于实现注册中心。

Eureka 采用了 C-S 的设计架构。Eureka Server 作为服务注册功能的服务器，它是服务注册中心。而系统中的其他微服务，使用 Eureka 的客户端连接到 Eureka Server，并维持心跳连接。这样系统的维护人员就可以通过 Eureka Server 来监控系统中各个微服务是否正常运行。Spring Cloud 的一些其他模块（比如Zuul）就可以通过 Eureka Server 来发现系统中的其他微服务，并执行相关的逻辑。

Eureka由两个组件组成：Eureka服务器和Eureka客户端。Eureka服务器用作服务注册服务器。Eureka客户端是一个java客户端，用来简化与服务器的交互、作为轮询负载均衡器，并提供服务的故障切换支持。



上图简要描述了Eureka的基本架构，由3个角色组成：

1. Eureka Server  
   • 提供服务注册和发现
2. Service Provider  
   • 服务提供方  
   • 将自身服务注册到Eureka，从而使服务消费方能够找到
3. Service Consumer  
   • 服务消费方  
   • 从Eureka获取注册服务列表，从而能够消费服务
4. 方案

本项目平台中以restful形式提供的服务都应该在一个注册中心进行注册以便于可以被其他服务发现并使用。

在项目的当前，Eureka可以覆盖当前服务注册中的功能。这里可以直接使用Eureka，通过Spring boot在集群中启动一个Eureka服务Server用于支持服务注册功能，其他的服务提供者通过Eureka客户端向Eureka的Server进行服务注册。

Spring Cloud是在Spring的基础之上发展起来的，相关的实现也是基于注解的形式进行的。

**Eureka Server**

单独启动一个模块作为Eureka Server，专门用于提供服务注册中心的功能，并配置相关地址。

在模块中引入相应Spring Cloud的起步依赖类，并进行一定的文件配置。

以下是Spring Cloud基于Spring boot的具体实现：

**@SpringBootApplication**

**@EnableEurekaServer**

**public** **class** **SpringCloudEurekaApplication** **{**

**public** **static** **void** **main(**String**[]** args**)** **{**

SpringApplication**.**run**(**SpringCloudEurekaApplication**.**class**,** args**);**

**}**

**}**

**Eureka Client**

Eureka Client也是基于Spring Boot的进行工作的，Eureka Client会把服务相关RestFul接口注册到Eureka Server中。代码实现上也是在Spring Boot的启动类上添加@SpringBootApplication、@EnableEurekaClient、@RestController这三个注解，同时在配置文件中写入Eureka Server的地址，在服务启动的时候就会伴随启动Eureka的客户端，在服务注册之后Eureka Client于Eureka Server之间通过心跳机制进行保持连接。

在当前项目中，三维展示子系统和数据分析子系统都是基于Spring Boot的。只需要在子系统的启动类上添加相关的注解即可，

但是目前Thingsboard有自己较为独立其完整的一套系统（用户管理、权限管理、接口管理），在对Thingsboard进行服务注册的时候需要对Thingsboard的接口进行一定的研究。

1. 服务网关

在Spring Cloud微服务系统中，一种常见方式是，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。

Zuul在Spring Cloud中扮演着服务网关的角色。通过启动一个Spring Boot模块用于提供服务网关的功能。具体实现同样是在Spring Boot的启动类上添加相关的注解，并进行相关配置。

相关配置如下：

eureka:

client:

serviceUrl:

defaultZone: http://localhost:8761/eureka/

server:

port: 8769

spring:

application:

name: service-zuul

zuul:

routes:

api-a:

path: /api-a/\*\*

serviceId: 3D-Service

api-b:

path: /api-b/\*\*

serviceId: DataAnalysis-Service

在启动该配置的Zuul服务网关后，访问该Zuul服务就会进行对应的服务调用（反向代理）。例如，在本地启动该Zuul服务后，访问地址： http://localhost:8769/api-b/hi?deviceName=wensidu&type=1，Zuul就会把请求转发发给DataAnalysis-Service服务。

同时，zuul不仅只是路由，并且还能过滤，做一些安全验证。通过在Zuul的Spring Boot中添加一个ZuulFilter组件可以进行相关的安全验证，该组件提供了四种不同生命周期的过滤器类型：路由之前、路由之时、路由之后和发送错误调用。

过滤操作可以从以下角度进行：

◦ filterOrder：过滤的顺序

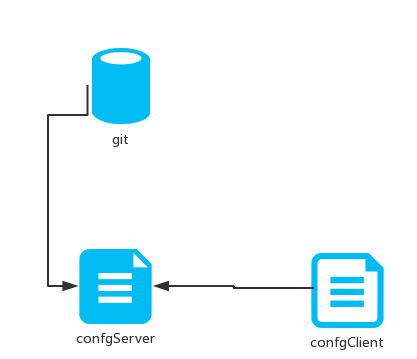
◦ shouldFilter：这里可以写逻辑判断，是否要过滤，本文true,永远过滤。

◦ run：过滤器的具体逻辑。可用很复杂，包括查sql，nosql去判断该请求到底有没有权限访问。

1. 配置中心

随着本项目的推进，系统的扩展，对系统的管理越发困难，其中各个子系统和子模块的配置信息的管理问题尤为突出。而Spring Cloud提供的配置中心组件Spring Cloud Config可以很好的解决该问题。

通过把系统所有配置放置到git上，由一个配置中心去git仓库中拉取所有的配置并保存到配置中心（貌似可以配置信息刷新），之后各个子系统和子模块（均为Spring Boot）通过一个配置Client 向配置中心拉取自己配置信息。



在Spring Cloud Config组件中，分两个角色，一是Config Server，二是Config Client。

通过启动一个Spring Boot项目用于提供Config Server服务。在Spring Boot程序的入口Application类加上@EnableConfigServer注解开启配置服务器的功能，之后在配置文件中添加如下信息：

• spring.cloud.config.server.git.uri：配置git仓库地址

• spring.cloud.config.server.git.searchPaths：配置仓库路径

• spring.cloud.config.label：配置仓库的分支

• spring.cloud.config.server.git.username：访问git仓库的用户名

• spring.cloud.config.server.git.password：访问git仓库的用户密码

Config Client（即需要配置信息的服务提供者）需要在自己配置文件中写入配置中心的地址，在该Client启动后就会自动的去配置中心拉取配置信息。

Client项目配置说明：

• spring.cloud.config.label 指明远程仓库的分支

• spring.cloud.config.profile

◦ dev开发环境配置文件

◦ test测试环境

◦ pro正式环境

• spring.cloud.config.uri= <http://localhost:8888/> 指明配置服务中心的网址。