# 一 概述

## 1.1 Springcloud

Springcloud是一个含概多个子项目的的开发工具集，集合了众多开源框架，利用了springboot的便利性实现了很多功能。

## 1.2 常用组件

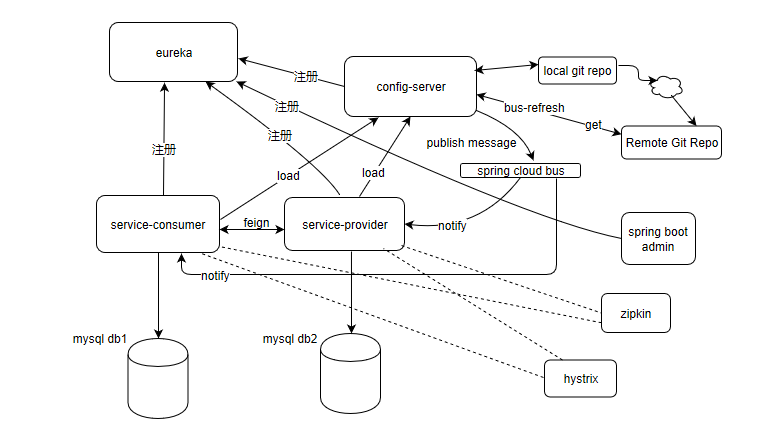
服务注册发现中心：eureka

分布式配置中心：config

Sleuth:链路追踪

Feign:声明式、模板化的http客户端

## 1.3 系统架构



## 1.4 开发环境

IDE: idea2019.2

jdk: 1.8.0\_121

maven: 3.6.1

Springcloud版本：Greenwich.SR2

Springboot版本：2.1.7.RELEASE

Rabbitmq: 3.8.0

Mysql:5.5.19

Mybatis:3.5.1

# 二 Eureka

## 2.1 Eureka作为服务注册中心

### 2.1.1 引入依赖

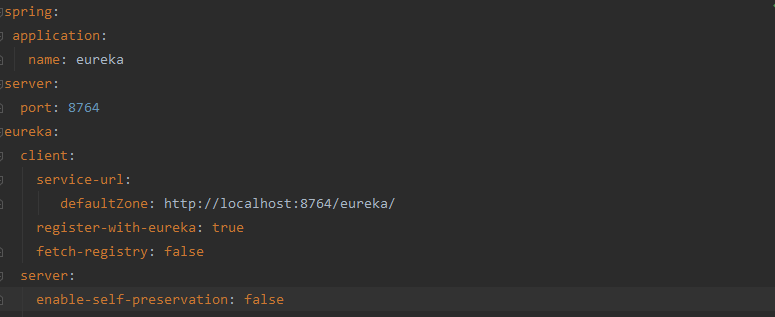
<dependency>

<groupId>org.springframework.cloud</groupId>

<artifactId>spring-cloud-starter-netflix-eureka-server</artifactId>

</dependency>

### 2.1.2 添加配置



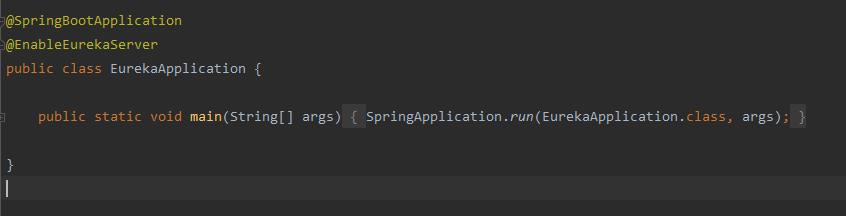
register-with-eureka: 是否向eureka注册

fetch-registry: 表示客户端是否从 Eureka Server 获取实例注册信息。

enable-self-preservation: 服务端自我保护机制。

### 2.1.3 @EnableEurekaServer注解

启动类上添加注解@EnableEurekaServer，标识其是一个eureka的Server。



## 2.2 微服务接入Eureka

### 2.2.1 引入依赖

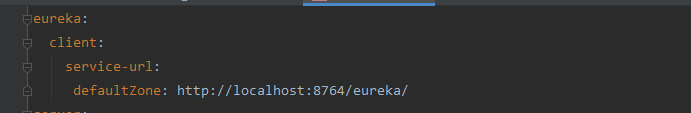
<dependency>

<groupId>org.springframework.cloud</groupId>

<artifactId>spring-cloud-starter-netflix-eureka-client </artifactId>

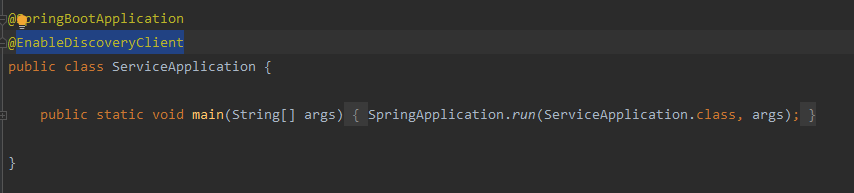
</dependency>

### 2.2.2 添加配置



### 2.2.3 @EnableDiscoveryClient注解

启动类上添加注解@EnableDiscoveryClient，标识其是一个eureka的client。



# 三 使用feign进行服务调用

Springcloud feign主要对netfilx feign进行增强和包装，是声明式的、模板化的的HTTP的客户端。Feign支持Spring MVC注解，整合了ribbon和eureka，使用简单方便。使用feign只需要创建一个接口，并在接口上添加一些注解，代码就完成了。

### 3.1 服务提供方提供feign接口

#### 3.1.1 引入依赖

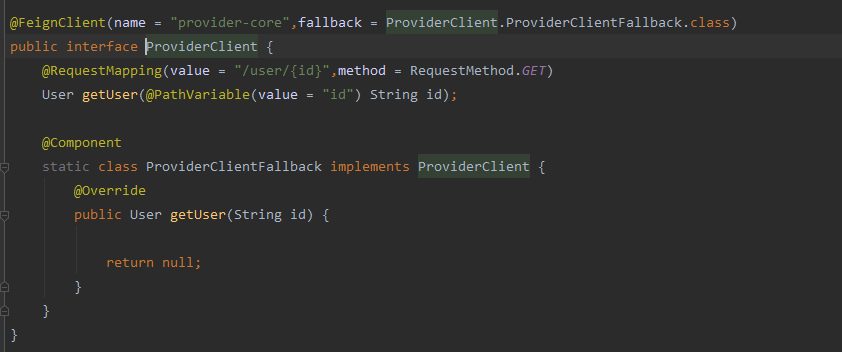
<dependency>

<groupId>org.springframework.cloud</groupId>

<artifactId>spring-cloud-starter-feign</artifactId>

</dependency>

#### 3.1.2 feign接口声明



使用@feignClient注解，表示该接口是一个feign的客户端。注解的name属性为暴露接口的服务名。将该接口打成一个jar包，供其他服务调用。

### 3.2 服务调用方使用feign接口

#### 3.2.1引入依赖

<dependency>

<groupId>org.springframework.cloud</groupId>

<artifactId>spring-cloud-starter-feign</artifactId>

</dependency>

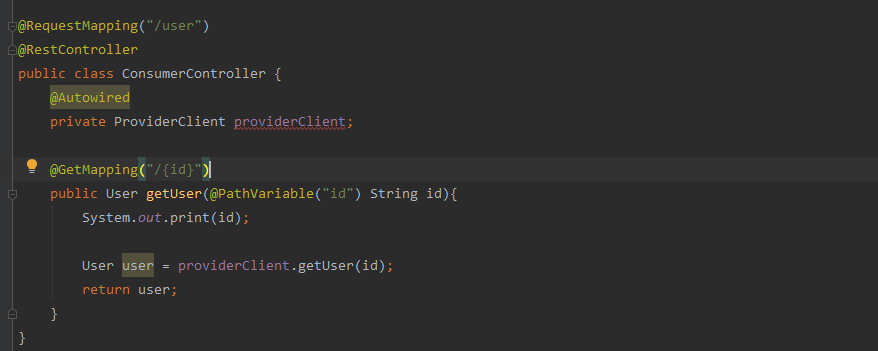
#### 3.2.2 通过注解声明feign接口实例

在启动类上添加@EnableFeignClients注解。



需要注意的是，引入的feign的接口后，其还未被实例化，需要通过@ComponentScan扫描。

#### 3.2.4 feign接口调用示例



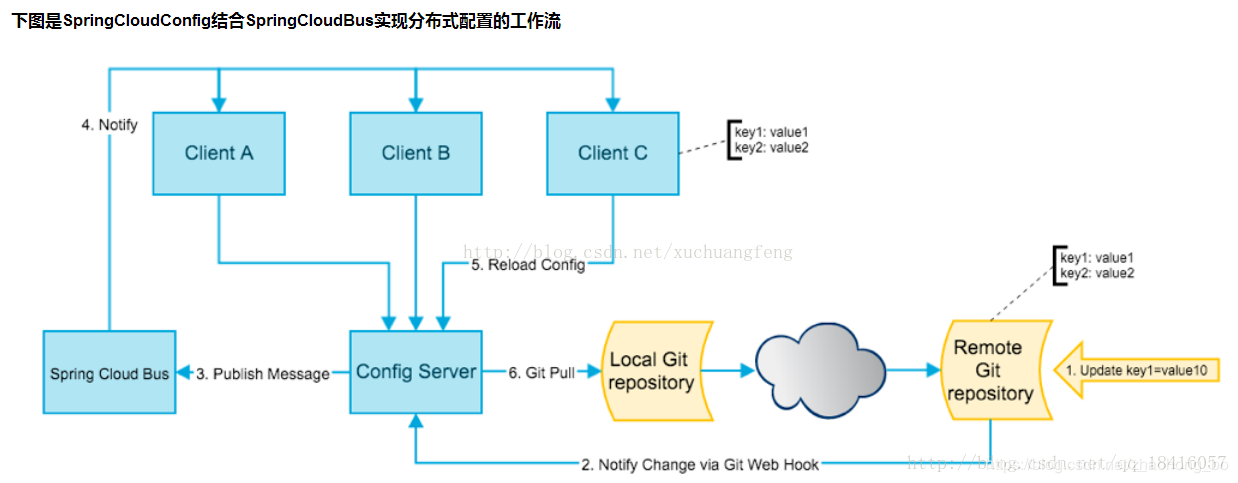
# 四 分布式配置中心Config

## 4.1配置中心作用

基于微服务架构风格的微服务，就意味着要将单体应用中的业务拆分成最小逻辑单元的子服务，每个服务的粒度相对较小，因此系统中会出现大量的服务。由于每个服务都需要必要的配置信息才能运行，所以一套集中式的、动态的配置管理体系设施是必不可少的。SpringCloud 提供了 Config Server 来解决这个问题，如果每一个微服务自己带着一个application.yml，而 Config Server 可以轻松的实现上百个配置文件的管理工作。

作用：

* 运行期间动态调整配置，不再需要在每个服务部署的机器上编写配置文件，服务会向配置中心统一拉取配置自己的信息
* 不同环境不同配置，动态化的配置更新，分环境部署比如：dev/test/prod/beta/release
* 当配置发生变动时，服务不需要重启即可感知到配置的变化并应用新的配置
* 将配置信息以 REST 接口的形式暴露
* 集中管理配置文件



## 4.2.Spring Cloud config作为配置中心

### 4.2.1 引入依赖

<dependency>

<groupId>org.springframework.cloud</groupId>

<artifactId>spring-cloud-config-server</artifactId>

</dependency>

<dependency>

<groupId>org.springframework.cloud</groupId>

<artifactId>spring-cloud-starter-bus-amqp</artifactId>

</dependency>

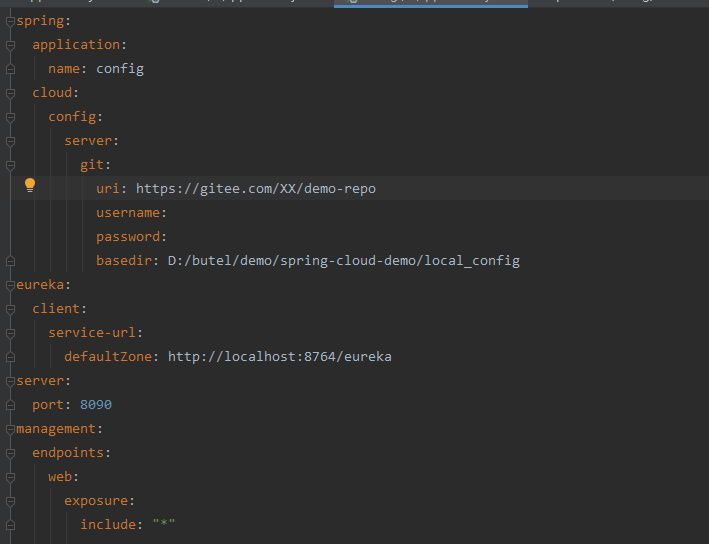
<dependency>

<groupId>org.springframework.cloud</groupId>

<artifactId>spring-cloud-config-monitor</artifactId>

</dependency>

### 4.2.2 添加配置

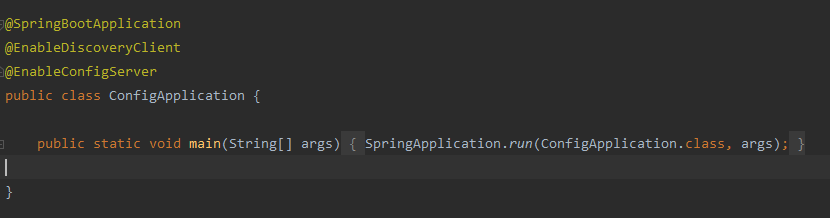


git：需在码云或者github配置存放配置文件的仓库

management.endpoints.web:exposure.inclde: 暴露所有的web断点，此时暴露的是/bus-refresh接口，供远端git修改之后，推送修改后的配置信息到服务。

### 4.2.3 @EnableConfigServer注解

启动类上添加@EnableConfigServer注解，标识其是一个配置中心。



## 4.3 微服务接入配置中心

### 4.3.1 引入依赖

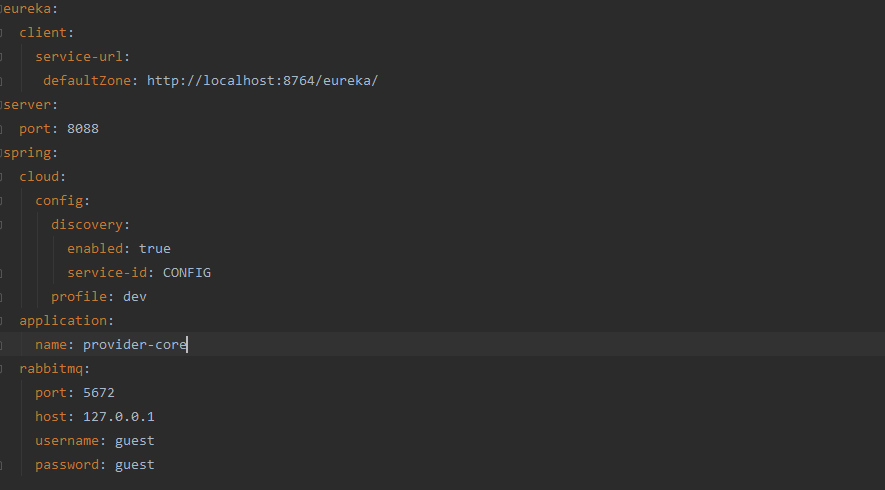
<dependency>

<groupId>org.springframework.cloud</groupId>

<artifactId>spring-cloud-config-client</artifactId>

</dependency>

### 4.3.2 添加yml配置



server-id:配置中心注册在eureka上的服务名

profile:当前服务使用什么环境。

### 4.3.3 git远端配置文件命名规则

/{application}-{profile}.yml ：

application代表应用名，profile代表环境，按照规则，如服务名是provider-core,使用dev环境，微服务启动时将通过配置中心去加载远端git的provider-core-dev.yml文件。

/{label}/{application}-{profile}.yml：

其中label代表分支名，application代表应用名，profile代表环境。

### 4.2.3 postman post方法调用/bus-refresh接口，动态更新配置

地址：<http://ip:port/actuator/bus-refresh>

<http://localhost:8090/actuator/bus-refresh>

### 4.2.4 码云或git的WebHooks配置回调地址

http://xxxxx.com/monitor

# 五 服务熔断降级hystrix

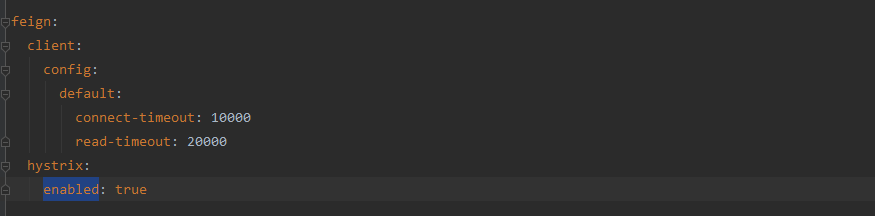
## 5.1 概述

Spring cloud Hystrix是Spring Cloud提供的一种带有熔断机制的框架，是基于netflix hystrix。由于在微服务系统中同一个操作会由多个不同的微服务来共同完成，所以微服务与微服务之间会由很多相互的调用，由于在分布式环境中有时会出现某个微服务节点故障的情况，所以会有调用失败发生，而熔断器的作用就是当出现远程调用失败的时候提供一种机制来保证程序的正常运行而不会卡死在某一次调用，类似Java程序中的try-catch结构，而只有当异常发生的时候才会进入catch的代码块。

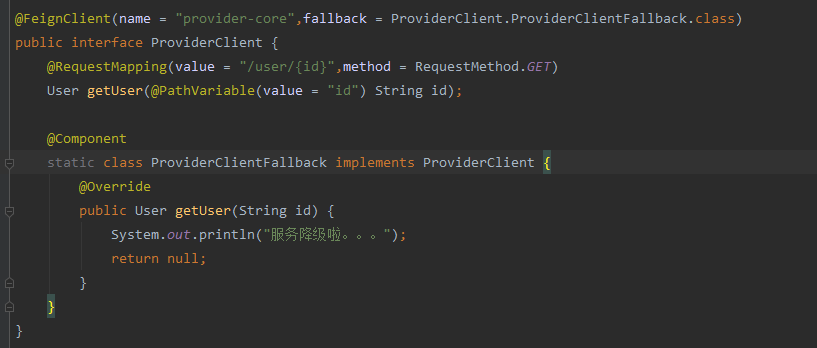
## 5.2 feign接口整合hystrix

### 5.2.1 添加配置

在yml配置中，打开配置：



### 5.2.3使用@FeigClient注解的fallBack属性配置降级后的业务处理类



Fallback配置处理服务降级/熔断时业务的类。

ProviderClientFallback类中的getUser方法处理/user/{id}降级时的逻辑。

## 5.3 hystrix-dashboard使用

spring cloud提供了hystrix dashbord用来监控hystrix。

### 5.3.1引入依赖

<dependency>

<groupId>org.springframework.cloud</groupId>

<artifactId>spring-cloud-starter-hystrix-dashboard</artifactId>

<version>1.4.3.RELEASE</version>

</dependency>

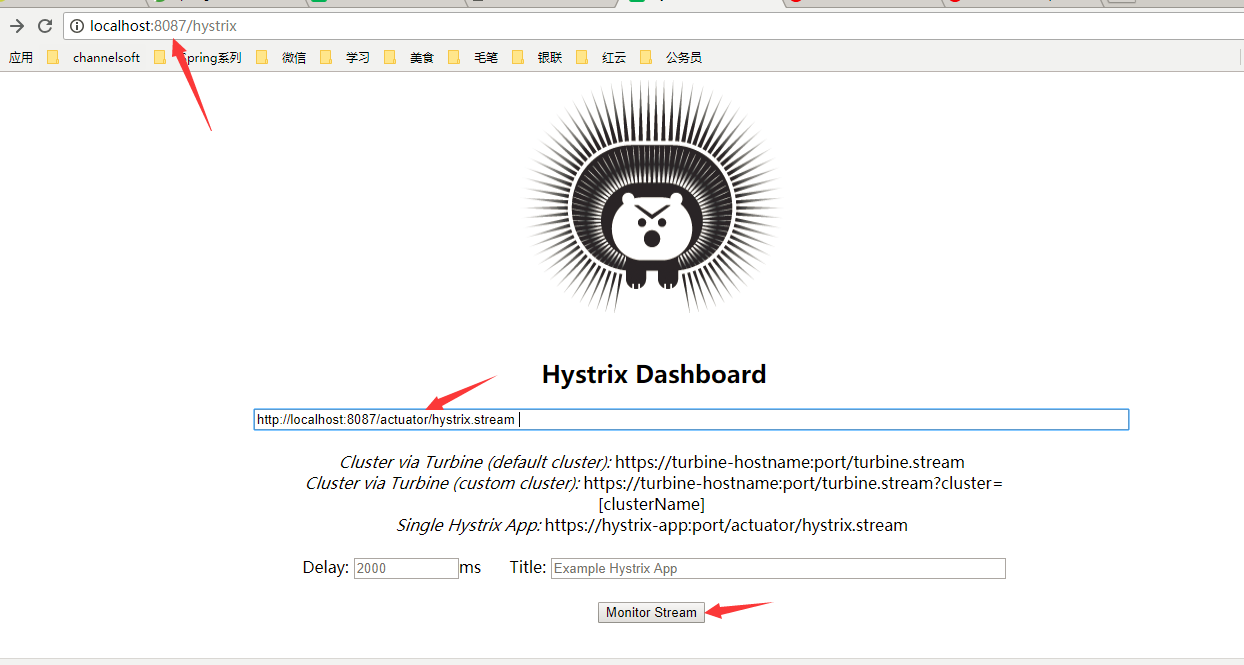
### 5.3.2 @EnableHystrixDashboard注解

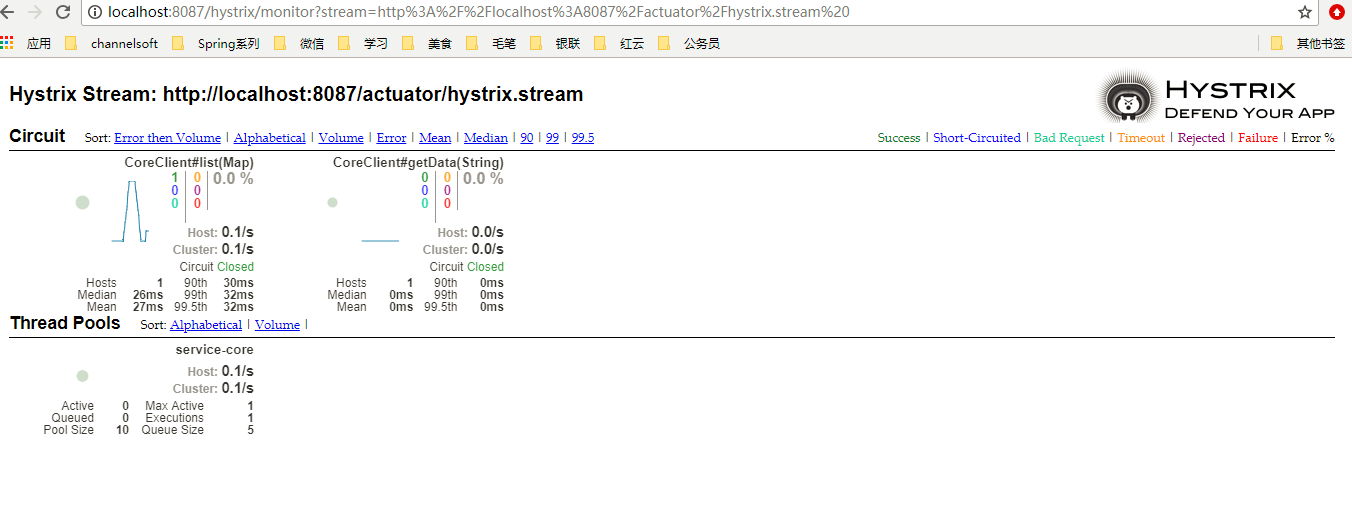
启动类添加@EnableHystrixDashboard注解，标识其是hystrix-dashbord监控端。



### 5.3.3界面展示

Consumer消费方的端口为8087,此时hystrix-dashbord监控的地址为localhost:8087/hystrix。





# 六 服务追踪链路监控sleuth+zipkin

## 6.1 Sleuth简述

在微服务架构中，项目中前端发起一个请求，后端可能跨几个服务调用才能完成这个请求。如果系统越来越庞大，服务之间的调用与被调用关系就会变得很复杂，假如一个请求中需要跨几个服务调用，其中一个服务由于网络延迟或异常等原因挂掉了，这时候我们需要分析具体哪一个服务出问题了就会显得很困难。

Spring Cloud Sleuth服务链路跟踪功能就可以帮助我们快速的发现错误根源以及监控分析每条请求链路上的性能等等。

## 6.2 Zipkin简述

zipkin是一个分布式的跟踪系统，由Twitter公司开发的, 其主要功能是聚集来自各个异构系统的实时监控数据，用来追踪微服务架构下的系统延时问题。

## 6.3 Sleuth+ZipKin使用

### 6.3.1 引入依赖

<dependency>

<groupId>org.springframework.cloud</groupId>

<artifactId>spring-cloud-starter-sleuth</artifactId>

</dependency>

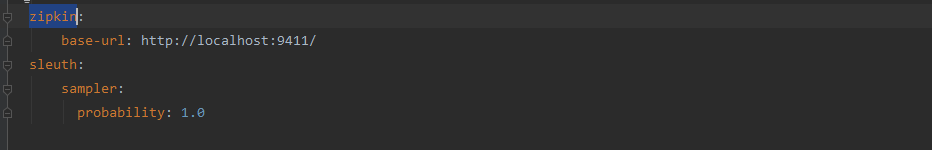
<dependency>

<groupId>org.springframework.cloud</groupId>

<artifactId>spring-cloud-starter-zipkin</artifactId>

</dependency>

### 6.3.2 添加配置



将链路请求抽样发送到第三方进行显示，probability为1表示所有的请求都发送到zipkin。一般将开发环境的probability设置为1，否则会照成服务性能问题。

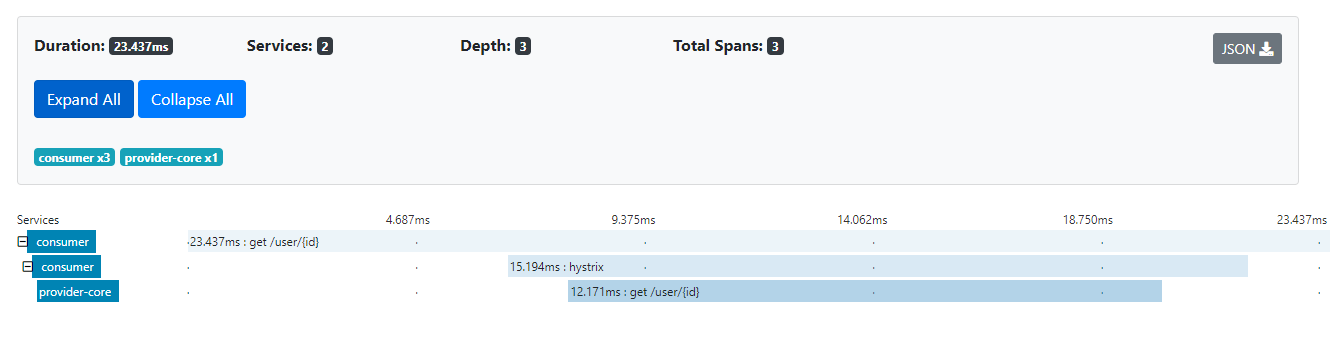
### 6.3.3 调整日志级别，显示链路信息



### 6.3.4.启动ZipKin

java -jar zipkin-server-2.11.9-SNAPSHOT-exec.jar

zipKin默认端口9411:，地址为http://localhost:9411



### 附 ZipKin安装

windows安装zipKin :

1. git克隆源码到本地：git clone <https://github.com/openzipkin/zipkin>
2. cd G:\MyProject\_SpringCloud\zipkin\zipkin
3. maven编译源码：mvnw -DskipTests --also-make -pl zipkin-server clean install
4. 执行zipkin编译出的jar包:

# 七 Spring Boot Admin

## 7.1 Spring Boot Admin简述

Spring Boot Admin 是一个管理和监控Spring Boot 应用程序的开源软件。可以在列表中浏览所有被监控spring-boot项目的基本信息，详细的Health信息、内存信息、JVM信息、垃圾回收信息、各种配置信息（比如数据源、缓存列表和命中率）等，还可以直接修改logger的level。

## 7.2 Spring Boot Admin使用

### 7.2.1引入依赖

<dependency>

<groupId>de.codecentric</groupId>

<artifactId>spring-boot-admin-starter-server</artifactId>

</dependency>

<dependency>

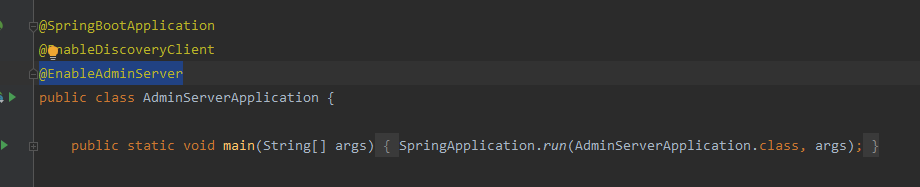
<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-security</artifactId>

</dependency>

### 7.2.2 @EnableAdminServer注解

启动类添加@EnableAdminServer注解标识其是一个adminServer。



### 7.2.3 管理界面展示

启动应用，默认端口为8788。进入地址<http://localhost:8788>监控在线的springboot项目。

