目录

[一、什么是Spring Cloud 1](#_Toc528244426)

[SpringCloud特点 1](#_Toc528244427)

[Spring cloud子项目（这里用到的）： 2](#_Toc528244428)

[二、spring cloud demo 2](#_Toc528244429)

[Eureka：服务注册 3](#_Toc528244430)

[Server端 3](#_Toc528244431)

[Client端 4](#_Toc528244432)

[Feign：服务调用 4](#_Toc528244433)

[Zuul：API网关 5](#_Toc528244434)

[Hystrix：服务熔断 6](#_Toc528244435)

[一. 第一种方式 6](#_Toc528244436)

[二．第二种方式 6](#_Toc528244437)

[Config：统一配置 7](#_Toc528244438)

[Server端 7](#_Toc528244439)

[Client端 7](#_Toc528244440)

# 一、什么是Spring Cloud

Spring Cloud是基于Spring Boot的一整套实现微服务的框架。提供了微服务开发所需的配置管理、服务发现、断路器、智能路由、微代理、控制总线、全局锁、决赛竞选、分布式会话和集群状态管理等组件。

## SpringCloud特点

1：约定优于配置

2：开箱即用、快速启动

3：适用于各种环境

4：轻量级的组件

5：组件支持丰富，功能齐全

## Spring cloud子项目（这里用到的）：

Spring cloud netflix：针对多种netflix组件提供的开发工具包，其中包括eureka、hystrix、zuul等。

Netflix eureka：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。

Netflix feign：Spring Cloud提供的一种声明式REST客户端。可以通过Feign访问调用远端微服务提供的REST接口。

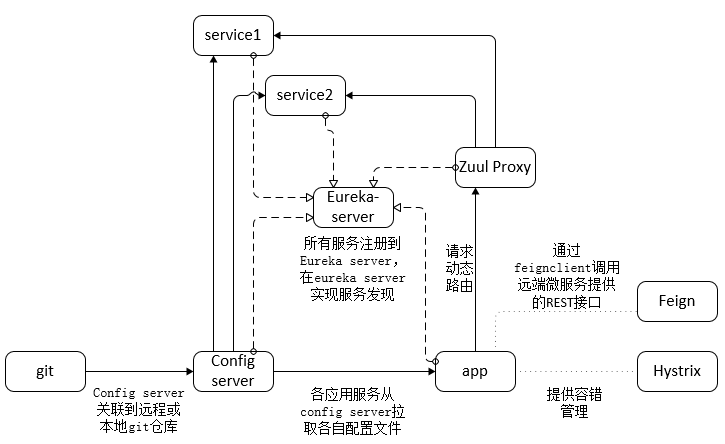
Netflix Hystrix：容错管理工具，旨在通过控制服务和第三方库的节点，从而对延迟和故障提供更强大的容错能力。

Netflix Zuul：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：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。

# 二、spring cloud demo

本demo的简要架构图



1.所有服务注册到eureka server，实现服务的注册、发现。

2.需要配置文件的应用从config server拉取配置，配置文件放在git上统一管理

3.app发送出请求时，通过feign调用zuul中根据请求关联到的service，实现请求路由

4.在连接超时等特殊情况时，通过Hystrix实现容错处理。

构建一个spring boot项目：

https://start.spring.io/

搭建spring cloud时要注意spring cloud版本与spring boot对应：[https://projects.spring.io/spring-cloud/#quick-start](https://projects.spring.io/spring-cloud/" \l "quick-start)

## Eureka：服务注册

### Server端

1.添加依赖：

<**dependency**>  
 <**groupId**>org.springframework.cloud</**groupId**>  
 <**artifactId**>spring-cloud-starter-netflix-eureka-server</**artifactId**>  
</**dependency**>

2.更改Application.java

添加注解@EnableEurekaServer

3.更改application.properties

**server.port**=**8761  
  
eureka.instance.hostname**=**localhost**

**#**是否注册自身到eureka服务器 **eureka.client.registerWithEureka**=**false  
eureka.client.fetch-registry**=**false  
eureka.client.serviceUrl.defaultZone**= **http://${eureka.instance.hostname}:${server.port}/eureka/  
#**服务关闭后是否保存一段时间 **eureka.server.enable-self-preservation**=**false**

启动后，访问<http://localhost:8761/>，可以看到注册的服务信息。

### Client端

1.添加依赖

<**dependency**>  
 <**groupId**>org.springframework.cloud</**groupId**>  
 <**artifactId**>spring-cloud-starter-netflix-eureka-client</**artifactId**>  
</**dependency**>

不要添加多余的依赖，因为可能会报错。比如我添加mybat/jpa/postgresql的依赖却没使用，在启动时报错检测不到spring.database.source之类的

2.更改Application.java

添加注解@EnableEurekaClient或者@EnableDiscoveryClient

3.更改application.properties

**spring.application.name**=**court-eureka  
eureka.client.service-url.defaultZone**=**http://localhost:8761/eureka/**

注：1.在使用druid作为数据源配置的时候报错：java.lang.ClassNotFoundException: org.springframework.boot.bind.RelaxedDataBinder。 是因为一开始druid使用的1.1.0版本，其中依赖的org.springframework.boot.bind在spring boot2.0.x版本中已经删除了。将druid升级到1.1.1版本，问题解决。

2.在编写dao层和其对应的mapper文件后，启动项目报No qualifying bean of type 'com.chaishuai.spring.demo.springcloud.dao.CourtDao' available: 找不到相应的Dao，最后在dao类上加注解@Mapper，解决问题

## Feign：服务调用

1.添加依赖

<**dependency>  
 <groupId>org.springframework.cloud</groupId>  
 <artifactId>spring-cloud-starter-openfeign</artifactId>  
</dependency>**

2.更改Application.java 添加注解@EnableFeignClients

3.在application.yml 添加配置 feign.client.config.feignName.connectTimeout: 5000

feign.client.config.feignName.readTimeout: 5000

4.编写client接口，调用其他在eureka注册的服务，示例如下

@Component  
@FeignClient(**"court-eureka"**) //要调用的已经注册的服务名  
**public interface** CourtClient {  
 @GetMapping(**"/court/listprovince"**)  
 Object getProvinceList();  
 @GetMapping(**"/court/listcourtid/{province}"**)  
 Object getCourtIds(@PathVariable(**"province"**) String provice);  
 @GetMapping(**"/court/getcourtname/{courtid}"**)  
 String getCourtName(@PathVariable(**"courtid"**) String courtid);  
}

？？在调用“court-eureka”服务时，以Object接收传来的“实际为String”的Object对象时报错：org.springframework.web.client.RestClientException: Could not extract response: no suitable HttpMessageConverter found for response type [class java.lang.Object] and content type [text/plain;charset=UTF-8]，模仿getCourtIds方法那样接收也不可以，最后改为String才行。

Feign本身有负载均衡算法，启动两个相同服务名的服务注册到同一个注册中心，feign会自动选择调用哪个服务。

## Zuul：API网关

1.添加依赖 ：spring-cloud-starter-netflix-zuul

2.在application.java 中添加注解@EnableZuulProxy

3.在application.yml中添加属性配置

**zuul:  
 routes:  
 court:  
 path:** /court/\*\*  
 **serviceId:** court-eureka

4.使用

@Component  
@FeignClient(**"eureka-client-zuul"**)  
**public interface** DataClient {  
 @GetMapping(**"/court/court/listcourtid/{province}"**)  
 Object getCourtIds(@PathVariable(**"province"**) String provice);  
}

注：配置的path：/court/\*\*，只是标识到了@GetMapping上，其value值要在这基础上拼接对应的服务请求方法。

## Hystrix：服务熔断

### 一. 第一种方式

1.添加依赖 spring-cloud-starter-netflix-hystrix

2.在Application.java添加注解 @EnableCircuitBreaker

3.在controller类中要熔断的方法之上加注解 @HystrixCommand(fallbackMethod=”方法名”)，示例：

@GetMapping(**"/trialinfo/{province}"**)  
@HystrixCommand(fallbackMethod = **"defaultTrialInfo"**)  
**public** Object getTrialInfo(@PathVariable(**"province"**) String province, Model model) {**return trialService**.getTrialInfo(province);  
}  
  
**public** Object defaultTrialInfo(String province, Model model) {  
 TrialInfoBO trialInfoBO = **new** TrialInfoBO();  
 trialInfoBO.setCourtName(**"hik"**);  
 trialInfoBO.setCaseBrief(**"test"**);  
 trialInfoBO.setCaseType(**"001"**);  
 trialInfoBO.setCaseNo(**"001"**);  
 trialInfoBO.setChiefjudgerName(**"cs"**);  
 **return** trialInfoBO;  
}

设置超时时间，默认为1000ms

@HystrixCommand(fallbackMethod = **"getDefaultMsg"**, commandProperties = {@HystrixProperty(name = **"execution.isolation.thread.timeoutInMilliseconds"**, value = **"10"**)})

注：fallbackMethod中指定的方法，要与原方法的返回值、参数相同。

### 二．第二种方式

1.添加依赖 spring-cloud-starter-netflix-hystrix

2. 在Application.java添加注解 @EnableCircuitBreaker

3.在application.yml中添加属性配置 feign.hystrix.enabled:ture

4.使用：

@Component

@FeignClient(value = **"eureka-client-zuul"**, fallback = DataClientFallback.**class**)  
**public interface** DataClient { …… }

然后编写DataClientFallback类，实现DataClient接口

@Component  
**public class** DataClientFallback **implements** DataClient { …… }

这样在DataClient要连接的服务不可用时（比如停掉其要调用的服务），就会去执行其本地方法（DataClientFallback中的）

## Config：统一配置

### Server端

1．添加依赖 spring-cloud-config-server

2. 更改Application.java。 添加注解@EnableConfigServer

3.更改application.yml 添加

spring.cloud.config.server.git.uri= https://github.com/Xin2XiangRong/springcloud

spring.cloud.config.server.git.searchPaths=config

配置文件由git统一管理

### Client端

1.添加依赖spring-cloud-config-client

2.添加属性配置

spring.cloud.config.name: trial-eureka-feign-gateway-hystrix-config2

spring.cloud.config.profile: dev

spring.cloud.config.uri: http://10.192.36.23:8088/

这样即可调用config server上的trial-eureka-feign-gateway-hystrix-config2-dev.properties文件

。

注：与spring-cloud相关的属性必须配置在bootstrap.yml中，config部分内容才能被正确加载。因为config的相关配置会先于application.properties，而bootstrap.yml的加载也是先于application.yml

yml格式的属性文件，要注意间隔符，即使复制过去也不一定正确

在client端不能正确读到属性文件时，可以先连接一下config-server端看看有没有内容显示。http://localhost:8088/trial-eureka-feign-gateway-hystrix-config2-dev.properties

在测试client端能不能连到server端时，可以test测试。

@Value(**"${author}"**)  
**private** String **author**;  
@Test  
**public void** contextLoads() {  
 System.***out***.println(**"\*\*\*\*\*\*\*\*\*2"**+**author**);  
}

“微服务架构”描述了一种将软件应用程序设计为可独立部署的服务套件的特定方式。

简而言之，微服务架构是一种将单应用程序作为一套小型服务开发的方法，每种应用程序都在其自己的进程中运行，并与轻量级机制（通常是HTTP资源的API）进行通信。

这些服务是围绕业务功能构建的，可以通过全自动部署机制进行独立部署。这些服务的集中化管理已经是最少的，它们可以用不同的编程语言编写，并使用不同的数据存储技术

通过服务（Sevice）实现组件化

只要我们参与过软件行业，这就存在一种期盼：通过将组件整合在一起来构建系统，这与我们在现实世界中看待事物的方式非常相似.

我们的定义是，组件是可独立更换和升级的软件单元。

微服务架构一样会用到各种库，但这种架构会把软件给拆分成各种不同的服务来实现组件化。这里我们定义两个重要的概念：库(library) 指的是链接到程序的组件，通过本地函数调用来

使用库提供的功能；而服务 (service) 是进程外的组件，通过网络服务请求 (web service request) 或者远程函数调用之类的机制来使用里面的功能。

之所以在组件化的软件里用服务，而不是库，一个主要原因就是各个服务是可以独立部署的。比如说，如果在同一个软件 [4] 里用了多个库，那么就算只是修改了其中一个，都会导致整个

软件要被重新部署；相反，如果用的是服务，那只需要重新部署修改过的就可以。