# SpringCloud概述

## 为什么使用SpringCloud

Spring Cloud提供了一套完整的微服务整体解决方案。提供了分布式配置中心、分布式锁、分布式跟踪、分布式服务治理、分布式任务调度平台等模块解决对应的问题。

## 什么是SpringCloud

Spring Cloud是一个分布式的整体解决方案。Spring Cloud 为开发者提供了在分布式系统（配置管理，服务发现，熔断，路由，微代理，控制总线，一次性token，全局琐，leader选举，布式session，集群状态） 中快速构建的工具，使用Spring Cloud的开发者可以快速的启动服务或构建应用、同时能够快速和云平台资源进行对接。

**SpringCloud分布式常用组件**

服务发现一 Netflix Eureka

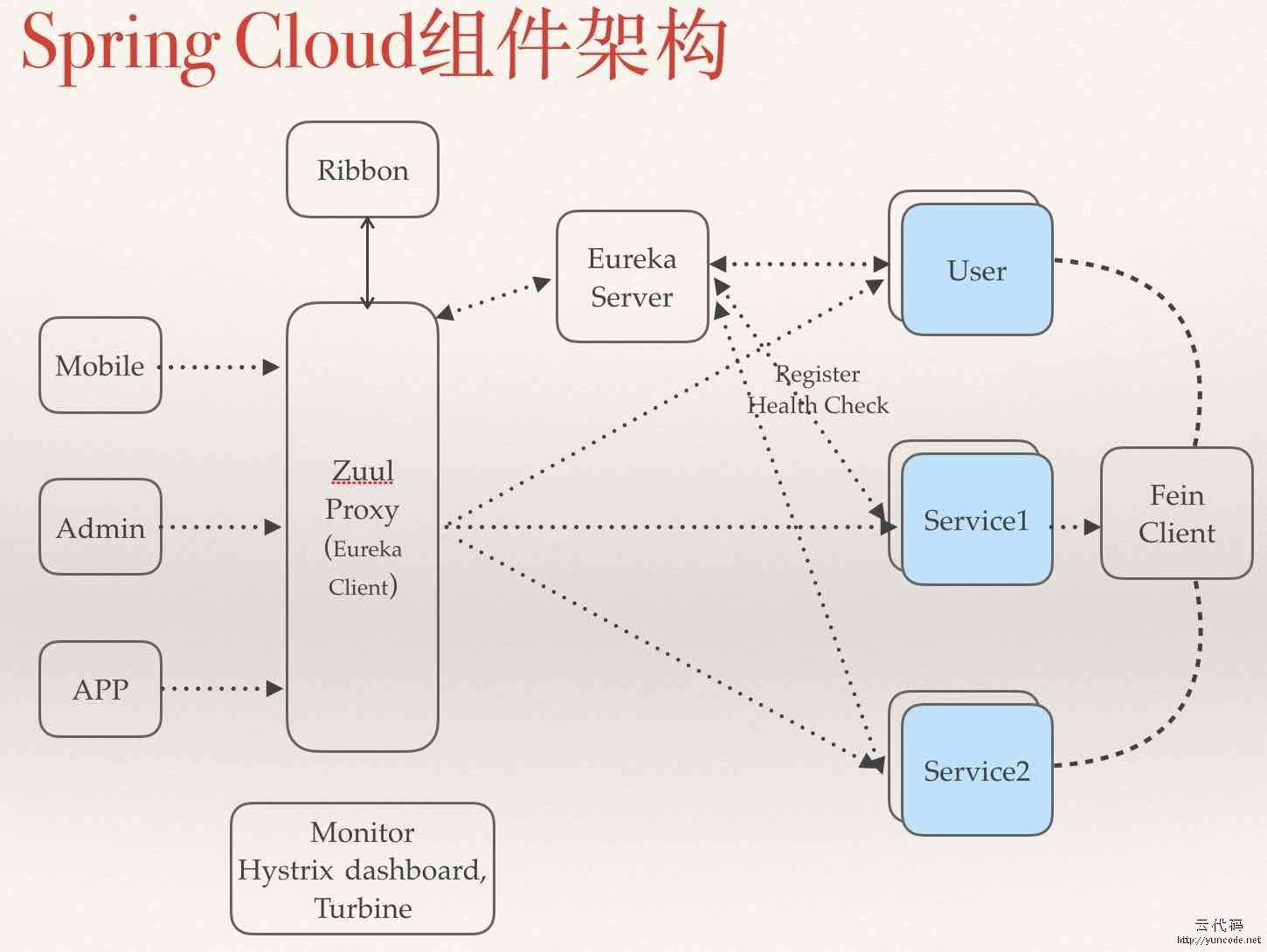
客服端负载均衡一Netflix Ribbon

断路器-Netflix Hystrix

服务网关-Netflix Zuul

分布式配置中心一Spring Cloud Config

服务调用组件—Feign



# 服务发现与注册

一个应用拆分为多个独立微服务。多个微服务可以使用Docker容器化进行多实例部署。

架构演化到这里遇到了一个难题，如果要查询用户所有的订单，用户服务可能会依赖订单服务，用户服务如何与订单服务交互呢？订单服务有多个实例该访问哪一个？服务注册和发现成为了一种很好的解决方案。

[**微服务**](https://so.csdn.net/so/search?q=%E5%BE%AE%E6%9C%8D%E5%8A%A1&spm=1001.2101.3001.7020)**架构中最核心的部分是服务治理**，服务治理最基础的组件是注册中心。随着微服务架构的发展，出现了很多微服务架构的解决方案，其中包括我们熟知的 Dubbo 和 Spring Cloud。

在SpringCloud 中提供了多种服务注册与发现组件：Eureka,Consul,Zookeeper。官方推荐使用Eureka。

在dubbo 支持了 Zookeeper、Redis、Multicast 和 Simple，官方推荐 Zookeeper。

## 相关概念

注册中心概念：注册中心是存放服务地址相关信息。

SpringCloud中支持Eureka、Consul、Zookeeper注册中心。

Dubbo支持常用的Redis和Zookeeper。

在传统架构中，服务与服务之间调用直接使用地址调用。

服务提供者：提供服务接口；

服务消费者：调用接口进行使用；

一个服务既可以做服务者，也可以作为消费者。

服务注册：将服务信息注册到注册中心上。

服务发现：从注册中心上获取服务信息。

## 基本步骤：

1. 首先启动注册中心
2. 启动服务提供者
3. 服务提供者启动的时候，会把当前服务的基本信息如服务地址和端口以别名方式注册到注册中心去。
4. 消费者在调用接口的时候使用别名去注册中心获取实际的RPC远程调用地址。
5. 如果消费者获取到了实际的RPC远程调用地址后，使用本地的HttpClient技术实现调用。默认情况下，eureka每隔30秒更新一次服务调用地址。

## 服务注册和发现搭建

## 搭建注册中心

**1.创建Springboot项目工程，并在父项目中添加springCloud依赖**

<dependencyManagement>

<dependencies>

<!-- springCloud核心包 -->

<dependency>

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

<artifactId>spring-cloud-dependencies</artifactId>

<version>2021.0.4</version>

<type>pom</type>

<scope>import</scope>

</dependency>

</dependencies>

</dependencyManagement>

**2.创建explore-eureka-server模块项目并添加对应依赖**

<!-- eureka server-->

<dependency>

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

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

</dependency>

**3.配置eureka注册中心**

server:

port: 9000 #服务注册中心端口号

spring:

application:

name: explore-eureka-server

eureka:

instance:

hostname: 127.0.0.1 #服务注册中心IP

client:

service-url:

defaultZone: http://${eureka.instance.hostname}:${server.port}/eureka #注册地址

register-with-eureka: false #是否往服务注册中心注册自己，默认为true。单机版的可设置成false。集群搭建中，设置true。

fetch-registry: false #单机版的可设置成false，集群版的由于需要同步其他节点的服务注册数据，故设成true。

server:

enable-self-preservation: false #是否开启自我保护模式，默认值true

eviction-interval-timer-in-ms: 60000 #扫描失效服务的间隔时间，单位毫秒，默认值 60 \* 1000

**4.配置启动类**

@SpringBootApplication

@EnableEurekaServer

**public** **class** EurekaApp {

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

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

}

}

## 搭建服务提供者

**1.创建服务模块(explore-member-service)并父项目中添加依赖**

<dependency>

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

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

</dependency>

**2.配置eurek将服务注册到eureka注册中心**

server:

port: 9010

spring:

application:

name: explore-member-service

profiles:

active: dev

eureka:

client:

service-url:

defaultZone: http://@eureka.instance.hostname@:@server.eureka.port@/eureka

instance:

prefer-ip-address: true

**3.配置启动类**

@EnableEurekaClient

@SpringBootApplication

**public** **class** MemberApp {

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

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

}

}

## 搭建Eureka集群环境

如果注册中心应为某一个原因出现故障了会导致整个微服务环境不可用，应该如何解决？解决办法：搭建注册中心集群

Eureka搭建集群原理使用相互注册原理，形成一组相互注册的注册中心，从而实现数据相互同步，达到高可用效果。

## Eureka自动保护机制

1. 服务提供者启动时，会把当前服务信息注册到eureka中；
2. 服务消费者调用服务提供者接口时，会去注册中心获取服务提供者实际RPC远程调用地址，获取地址后，在本地使用httpClient术进行远程调用。
3. Eureka自我保护机制：

默认情况下,EurekaClient定时向EurekServer端定时发送心跳包。

如果EurekaServer端在一定的时间内没有收到EurekaClient发送心跳包，便会直接从服务注册列表中剔除该服务。但是在短时间内丢失大量的服务实例心跳。这时候EurekaServer会开启自我保护机制，不会剔除该服务。

为了防止EurekaClient可正常运行，但是与EurekaSever网络不通的情况下，EurekaServer不会将EurekaClient服务进行剔除。

4.在什么环境下开启自我保护机制

建议在本地环境中禁用自我保护机制,在开发环境中开启自我保护机制。

如果服务真的宕机了，本地调用应该使用重试机制、保证接口网络延迟幂等性、服务降级功能。

## SpringCloud整合Zookeeper作为注册中心

Zookeeper:分布式调用工具，可以实现注册中心；采用零时节点方式注册服务，零时节点和生命周期进行关联，如果服务断开连接后，给节点会自动被删除。

安装zookeeper工具

创建服务消费者

## Zookeerp整合Ribbon负载均衡

使用RestTemplate做远程调用；

SpringCloud整合Consul注册中心

安装consul注册中心

创建服务消费者

# 客户端本地负载均衡

本地负载均衡本地从Eureka上获取对应的注册信息列表，获取到本地列表之后，让本地实现负载均衡，是在客户端实现的。

# Ribbon和Nginx实现负载均衡的区别

1. ribbon是本地负载均衡，在调用接口的时候，会在Eureka注册中心上获取注册信息列表，获取成功后，缓存在JVM本地。在本地使用rpc远程调用技术进行调用。即是在客户端实现负载均衡的。
2. Nginx是服务器端负载均衡，客户端发送的所有请求都会发个Nginx，让后让Ngnix实现转发请求。即是在服务端实行负载均衡的。
3. 本地负载均衡器适合在微服务rpc远程调用，如dubbo、springCloud。
4. Nginx服务器负载均衡适合于针对服务器端比如tomacat。

# 服务网关

为什么使用网关服务

<https://blog.51cto.com/u_15486212/5286296>

# SpringCloud Feign声明式客户端调用工具

SpringCloud中支持两种客户端调用工具：feign和rest；

Feign是一个声明式的Http客户端调用工具，才有接口+注解的方式实现，可读性强。

Feign客户端超时配置

Fegin客户对Ribbon的支持

1. 添加依赖

<dependency>

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

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

</dependency>

2.在服务消费者启动类中添加@EnableFeignClients注解

@SpringBootApplication

@EnableEurekaClient

@EnableFeignClients

**public** **class** ShiroApp {

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

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

}

}

1. 在接口上添加对应注解@FeignClient

@FeignClient(name="explore-member-service")

**public** **interface** MemberTestApi {

@RequestMapping("/hello")

String hello();

}

# RestTemplate调用服务接口

在SpringCloud中有两种调用方式：rest和fegin

RestTemplate是springboot web组件提供的，底层使用了httpClinent技术。默认整合了ribbon负载均衡器。

@SpringBootApplication

@EnableEurekaClient

@EnableFeignClients

**public** **class** ShiroApp {

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

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

}

//1.将RestTemplate注册到Spring容器中

//2.使用rest方式别名方式调用依赖负载均衡器,@LoadBalanced能让RestTemplate在请求时拥有客户端负载均衡的作用

@Bean

@LoadBalanced

RestTemplate restTemplate() {

**return** **new** RestTemplate();

}

}

使用

**public** String testRest() {

//使用别名方式,使用别名方式取从注册中心上获取对应的服务调用地址。使用rest方式别名方式调用依赖ribbon负载均衡器

String forObject = restTemplate.getForObject("http://explore-member-service/member/hello", String.**class**);

**return** forObject;

}

# SpringCloud服务保护框架Hystrix

服务雪崩产生原因

Tomcat有个线程池，每个线程去处理客户端发送的请求。默认情况下tomacat只有一个线程池处理客户端发送的所有服务请求，在高并发的情况下,如果客户端的所有请求堆积到同一个服务接口上，就会产生tomcat的所有线程去处该理服务接口，可能导致服务接口无法访问。就会导致其他服务接口访问的时候，产生延迟和等待。可能导致整个微服务接口不能访问，所有的服务都会瘫痪。

如果tomcat最大请求数20，客户端同一时间发送了100个请求，会有80个请求产生延迟等待。如何证明tomacat默认情况下只有一个线程去处理所有客户端的请求，看线程的名称。Thread.currentTheread.getName()

Hystrix是服务保护框架，在微服务中Hystrix能帮我们解决断路器、服务器降级、服务器雪崩效应、服务熔断、服务隔离机制。

基于Hysrix解决服务器雪崩效应原理：服务降级、服务隔离、服务熔断。

服务降级：在高并发情况下（在tomcat中没有线程进行处理客户端请求的时候，不应该让用户一直等待。），防止用户一直等待，使用服务降级方式，返回一个友好的提示给客户端，不会去处理请求，调用fallback方法。目的是为了用户体验。

服务熔断：服务熔断是为了保护服务，在高并发的情况下，如果请求达到极限（阈值），如果流量超出阈值，自动开启服务保护功能，使用服务降级方式返回一个友好提示。服务熔断机制和服务降级一起使用。

服务隔离：

线程池隔离:每个服务都有自己独立的线程池，每个线程池互不影响。缺点是cpu占用率非常高。一般用于核心的接口。

信号量隔离：