**什么是Spring Cloud**

Spring Cloud是一个微服务框架，集成了各种功能组件，包括服务注册发现、统一配置管理、服务远程调用、统一网关路由等等

**优缺点**

优点：组件丰富，功能齐全。缺点：微服务过多，治理成本高，不利于维护系统

**Spring Cloud Config**

**微服务配置管理**——**统一管理服务的配置信息**

**Spring Cloud Consul**

微服务注册发现——注册服务，集中管理服务

**Spring Cloud Netflix**

Netflix OSS 开源组件集成，包括Eureka、Hystrix、Ribbon、Feign、Zuul等核心组件。

Eureka： 微服务注册发现——注册服务，集中管理服务

Ribbon：微服务负载均衡

Hystrix：微服务熔断——能够快速失败并返回一个备选的响应

Feign：微服务接口调用——多个服务之间的通讯

Zuul：API网关——有路由转发的作用

**Spring Cloud Gateway**

API网关——有路由转发的作用

**Spring Cloud OpenFeign**

微服务接口调用——多个服务之间的通讯

**Spring Cloud和SpringBoot版本对应关系**

|  |  |
| --- | --- |
| **Spring Cloud**  **Version** | **SpringBoot**  **Version** |
| Hoxton | 2.2.x |
| Greenwich | 2.1.x |
| Finchley | 2.0.x |
| Edgware | 1.5.x |
| Dalston | 1.5.x |

**SpringBoot和SpringCloud的区别？**

SpringBoot可以离开SpringCloud独立使用，SpringCloud离不开SpringBoot

SpringBoot专注于快速、方便的开发单个微服务个体，SpringCloud关注全局的服务治理框架。

微服务

微服务是一种分布式架构方案，微服务按业务边界划分，高内聚低耦合**。**单一职责，每一个服务都对应唯一的业务能力，面向服务，微服务对外暴露业务接口。各个服务独立自治，隔离性强。

Spring Cloud是一个微服务框架，集成了各种功能组件，包括服务注册发现、统一配置管理、服务远程调用、统一网关路由、服务熔断降级等等

**Nacos自动刷新配置**

@RefreshScope注解

微服务架构原理

更细小粒度服务的拆分，将功能分解到各个服务当中，从而降低系统的耦合性，提供更加灵活的服务支持

各个组件的作用

Eureka微服务的注册与发现，用于定位服务

Nacos，配置中心，统一管理配置、动态更新不重启

Ribbon：负载均衡。从多个服务实例中挑一个访问，负载均衡算法，例如轮询、随机等

Feign，微服务之间进行远程调用

Gateway，微服务路由，用于身份认证、权限校验、请求限流

Hystrix，微服务熔断，当依赖的服务出现故障时，能够快速失败并返回一个备选的响应

nacos配置文件加载顺序

bootstrap.yml > application.yml > application-dev.yml > nacos-service.yaml > nacos-service-dev.yaml

本地bootstrap.yml配置环境dev

spring:

profiles:

active: dev

本地配置文件

bootstrap.yml

application.yml

application-dev.yml

nacos配置文件

nacos-service.yaml

nacos-service-dev.yaml

sentinel熔断配置

1. 修改yml文件，开启feign的sentinel功能
2. 给FeignClient编写失败后的降级逻辑，定义类并实现FallbackFactory接口，实现方法并编写降级逻辑，注册成为一个bean对象，
3. 在feign接口的FeignClient注解上指定fallbackFactory





分布式事务@GlobalTransactional原理

全局事务由多个本地事务和一张表实现，其中表记录的是已提交的本地事务的XID。若所有的本地事务都成功提交，那么全局事务也成功提交了；若存在一个本地事务失败，那么全局事务就会根据表来查询出已提交的本地事务，将这些事务回滚，从而实现全局事务的回滚。

请求从外部到微服务的数据流，以及微服务调用微服务的数据流，都经过哪些组件

外部客户端 → API 网关 → 负载均衡 → 服务发现 → 目标微服务实例

微服务A → 服务发现 → 负载均衡 → 微服务B实例