SpringCloud 相关概念解析

作者：ChenforCode
原文链接：https://ld246.com/article/1593767338438
来源网站：链滴
许可协议：署名-相同方式共享 4.0 国际 (CC BY-SA 4.0)

● Eureka相关：首先Eureka分为Server和Client，其中Server可以为单节点，也可以为多节点，其中节点主要是为了防止该节点坏掉之后所有节点都不能使用。

● Eureka Server：首先，单节点Server的话，主要有以下配置：

```
# eureka.client.fetch-registry: 表示是否从 Eureka Server 获取注册信息，默认为true。如果这是一单点的 Eureka Server，不需要同步其他节点的数据，设为false
    fetch-registry: false
    # eureka.client.register-with-eureka: 表示是否将自己注册到 Eureka Server, 默认为true。由于前应用就是 Eureka Server, 因此设为 false
    register-with-eureka: false
    # 设置 Eureka Server 所在的地址，查询服务和注册服务都需要依赖这个地址
    service-url:
      defaultZone: http://${eureka.instance.hostname}:${server.port}/eureka/
```

其中fetch-registry代表是否需要从Eureka Server中获取所有服务的注册信息，由于是单点的服务注中心，他自己本身就有了所有的注册信息，不需要从其他注册中心节点中获取。register-with-eurek代表是否将该服务注册到服务中心，由于自己就是服务中心，因此不注册。service-url代表服务注册心的地址，也就是将本服务注册到哪个位置，不是自己向外提供服务的地址。

● 多节点的Eureka Server：多节点的配置如下：

```
---
spring:
  application:
    name: ad-eureka
  profiles: server1
server:
  port: 8000
eureka:
  instance:
    hostname: server1
    prefer-ip-address: false
```

```
  client:
    service-url:
      defaultZone: http://server2:8001/eureka/,http://server3:8002/eureka/

---
spring:
  application:
    name: ad-eureka
  profiles: server2
server:
  port: 8001
eureka:
  instance:
    hostname: server2
    prefer-ip-address: false
  client:
    service-url:
      defaultZone: http://server1:8000/eureka/,http://server3:8002/eureka/

---
spring:
  application:
    name: ad-eureka
  profiles: server3
server:
  port: 8002
eureka:
  instance:
    hostname: server3
    prefer-ip-address: false
  client:
    service-url:
      defaultZone: http://server1:8000/eureka/,http://server2:8001/eureka/
```

由于三个节点都是注册中心，但是需要不断的保持三个注册中心的注册信息一直，因此某个节点需要直向另外两个节点发送请求进行实时更新。因此以server1为例，他首先需要将自己注册为server2和sever3的服务，并且不断将server2和server3中的信息更新到自己中来。那么fetch-registry和register-ith-eureka都默认为true。server1的service-url写的是server2和server3的地址。

● Eureka Client：我认为我们可以把所有的除了注册中心以外的所有服务都看作为Client。但是clien本身又可以分为纯粹的服务消费者，既有消费又有服务。对于单纯的服务消费者，可以不将自己注册Eureka Server中，因为不会有其他服务调用该服务。

● Ribbon：客户端的负载均衡，只要是如果相同的服务存在于不同的服务器中，客户端访问的时候通过负载均衡算法选择一个服务进行调用。

● Hystrix：服务熔断。设想以下过程：如果服务A调用B，B调用服务C，C调用服务D。如果此时服务出现错误，那么服务ABC的请求均处在一个延迟状态，这样由于一个单个服务导致所有的请求处于延状态最终导致资源耗尽，最终导致服务不可用与系统不可用。这种现象称为“雪崩”。服务熔断就是了解决该问题，Hystrix提供的保护机制主要分为断路器和线程隔离等机制。

● 断路器机制：当某个服务单元发生故障（类似用电器发生短路）之后，通过断路器的故障监控（类熔断保险丝），向调用方返回一个错误响应，而不是长时间的等待。这样就不会使得线程因调用故服务被长时间占用不释放，避免了故障在分布式系统中的蔓延。

● 线程隔离机制：它会为每一个依赖服务创建一个独立的线程池，这样就算某个依赖服务出现延迟过的情况，也只是对该依赖服务的调用产生影响，而不会拖慢其他的依赖服务。

● Hystrix提供几个熔断关键参数：滑动窗口大小（20）、 熔断器开关间隔（5s）、错误率（50%）每当20个请求中，有50%失败时，熔断器就会打开，此时再调用此服务，将会直接返回失败，不再调程服务。直到5s钟之后，重新检测该触发条件，判断是否把熔断器关闭，或者继续打开。

● Feign：Feign是一种负载均衡的HTTP客户端, 使用Feign调用API就像调用本地方法一样，从避免调用目标微服务时，需要不断的解析/封装json 数据的繁琐。下面给出Feign的用法。

● 首先Feign是一个HTTP的客户端，因此feign一般是写在你要调用某个服务的地方。加入你要在A服中调用B服务，那么feign是写在A服务中的。

● 建立一个FeignClient：其中sponsor就是请求的服务，该服务是注册到Eureka Server中的。

```
@FeignClient(value = "eureka-client-ad-sponsor",
        fallback = SponsorClientHystrix.class)
public interface SponsorClient {

    @RequestMapping(value = "/ad-sponsor/get/adPlan",
            method = RequestMethod.POST)
    CommonResponse<List<AdPlan>> getAdPlans(
            @RequestBody AdPlanGetRequest request);
}
```

这里的Client类可以和Service看做同层，需要注入到Controller中进行调用。

● Controller调用：可以直接将Client注入到Controller中，然后直接调用。

```
@IgnoreResponseAdvice
    @PostMapping("/getAdPlans")
    public CommonResponse<List<AdPlan>> getAdPlans(
            @RequestBody AdPlanGetRequest request
    ) {
        log.info("ad-search: getAdPlans -> {}",
                JSON.toJSONString(request));
        return sponsorClient.getAdPlans(request);
    }
```

注入的过程省略。

● 在该服务的启动类中，需要加入一个@EnableFeignClients注解，代表该类是一个Feign客户端。

● 调用的服务长什么样子：

```
@PostMapping("/get/adPlan")
    public List<AdPlan> getAdPlanByIds(
            @RequestBody AdPlanGetRequest request) throws AdException {
        log.info("ad-sponsor: getAdPlanByIds -> {}",
                JSON.toJSONString(request));
        return adPlanService.getAdPlanByIds(request);
    }
```

调用的服务就是正常的一个向外暴露的接口。但是要注意，feignClient中调用的url和这里的接口url保持一致。这样才能通过feign调用到该服务。

● Zuul：API网关，外层调用的所有调用均需要通过Zuul，也就是整个系统的大门，从外部想要进入须通过该大门，在API网关服务上进行统一调用来对微服务接口做前置过滤，以实现对微服务接口的截和校验。

● Zuul和Feign的关系：Zuul主要是把控外部调用的请求，即从外部进入到系统的请求。Feign主要制的是各个微服务之间的调用请求。例如一个小区，Zuul是小区的大门，控制人从外部进入到小区内Feign类似于各个楼栋的大门，负责小区内部各个楼栋之间的调用关系。Zuul和Feign都支持Ribbon

Hystrix，也即是负载均衡和服务熔断。

● Spring Cloud Config：是一个解决分布式系统的配置管理方案。它包含了Client和Server两个部，server提供配置文件的存储、以接口的形式将配置文件的内容提供出去，client通过接口获取数据并依据此数据初始化自己的应用。简单来说，使用Spring Cloud Config就是将配置文件放到统一的置管理(比如GitHub)，客户端通过接口去获取这些配置文件。在GitHub上修改了某个配置文件，应加载的就是修改后的配置文件。修改了配置文件，希望不用重启来动态刷新配置，配合Spring Cloud us 使用。