Spring Cloud Eureka：服务治理

- 主要负责完成微服务架构中的服务治理功能，其实现围绕着服务注册与服务发现机制来完成对微服务应用实例的自动化管理
- 随着业务的发展，系统功能越来越复杂，相应的微服务应用也不断增加，我们的静态配置就会变得越来越难以维护。
- 并且面对不断发展的业务， 我们的集群规模、 服务的位置 、 服务的命名等都有可能发生变化，如果还是通过手工维护的方式， 那么极易发生错误或是命名冲突等问题。
- 对于这类静态内容的维护也必将消耗大量的人力

服务注册

- 每个服务单元向注册中心登记自己提供的服务， 将主机与端口号、 版本号、 通信协议等一些附加信息告知注册中心， 注册中心按服务名分类组织服务清单
- 另外， 服务注册中心还需要以心跳的方式去监测清单中的服务是否可用， 若不可用需要从服务清单中剔除， 达到排除故障服务的效果

服务发现

- 由于在服务治理框架下运作， 服务间的调用不再通过指定具体的实例地址来实现， 而是通过向服务名发起请求调用实现
- 所以， 服务调用方在调用服务提供方接口的时候， 并不知道具体的服务实例位置
- 调用方需要向服务注册中心咨询服务， 并获取所有服务的实例清单， 以实现对具体服务实例的访问
- 比如， 现有服务C希望调用服务A, 服务C就需要向注册中心发起咨询服务请求， 服务注 册中心就会将服务A的位置清单返回给服务C, 如按上例服务A的情况，C便获得了服务A的两个可用位置192.168.0.100:8000和192.168.0.101:8000。 当服务C要发起调用的时候， 便从该清单中以某种轮询策略取出 一 个位置来进行服 务调用

Eureka

- 服务端与客户端均采用Java编写
- 由于Eureka服务端的服务治理机制提供了完备的RESTful APL所以它也支持将非Java语言构建的微服务应用纳入Eureka的服务治理体系中来
- Eureka服务端，也称为服务注册中心
    - 支持高可用配置。它依托于强一致性提供良好的服务实例可用性，可以应对多种不同的故障场景
        - 如果Eureka以集群模式部署， 当集群中有分片出现故障时， 那么Eureka就转入自我保护模 式。 它允许在分片故障期间继续提供服务的发现和注册， 当故障分片恢复运行时， 集群中 的其他分片会把它们的状态再次同步回来
- Eureka客户端，主要处理服务的注册与发现
    - 客户端服务通过注解和参数配置的方式，嵌入在客户端应用程序的代码中，

    - 在应用程序运行时，Euerka客户端向注册中心注册自身提供的服务并周期性地发送心跳来更新它的服务租约
    - 同时， 它也能从服务端查询当前注册的服务信息并把它们缓存到本地并周期性地刷新服务状态

服务注册中心

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

- 通过`@EnableEurekaServer`注解主类启动一个服务注册中心提供给其他应用进行对话
- 在默认设置下， 该服务注册中心也会将自己作为客户端来尝试注册它自己

```
# application.properties
server.port=1110
eureka.instance.hostname=localhost
#关闭自我保护
eureka.server.enable-self-preservation=false
# 自己不需要注册自己
eureka.client.register-with-eureka=false
# 注册中心不需要去检索服务
eureka.client.fetch-registry=false
# 访问 http://${eureka.instance.hostname}:${server.port}
eureka.client.serviceUrl.defaultZone=http://${eureka.instance.hostname}:${
server.port}/eureka/
```

- 高可用注册中心
    - 实际上就是将自己作为服务向其他服务注册中心注册自己， 这样就可以形成一组互相注册的服务注册中心， 以实现服务清单的互相同步， 达到高可用的效果
        - 创建application-peer1.properties 作为peer1服务中心的配置， 并将serviceUrl指向peer2

          ```
          spring.application.name=eureka-server
          server.port=1111

          eureka.instance.hostname=peer1
          eureka.client.serviceUrl.defaultZone=http://peer2:1112/eureka/
          ```

        - 创建application-peer2.properties 作为peer1服务中心的配置， 并将serviceUrl指向peer1

```
spring.application.name=eureka-server
server.port=1112

eureka.instance.hostname=peer2
eureka.client.serviceUrl.defaultZone=http://peer1:1111/eureka/
```

- 在/etc/hosts文件中添加对peer1和 peer2的转换， 让上面配置的host形式的 serviceUrl能在本地正确访问到

```
127.0.0.1 peerl
127.0.0.1 peer2
```

- 通过spring.profiles.active属性来分别启动peerl和peer

```
 java -jar eureka-server-1.0.0.jar --spring.profiles.active=peer1
 java -jar eureka-server-1.0.0.jar --spring.profiles.active=peer2
```

- 访问peer1的注册中心http://localhost:1111/和peer2的注册中心http://localhost:1112/
- 服务提供方

```
eureka.client.serviceUrl.defaultZone=http://peerl:llll/eureka/,htt
p://peer2:lll
2/eureka/
```

- 不想使用主机名来定义注册中心的 地址， 也可以使用IP地址的形式， 但是需要在配置文件中增加配置参数eureka.instance.prefer江p-address= true, 该值默认为false

服务提供者

```
<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-web</artifactId>
</dependency>
<dependency>
  <groupId>org.springframework.cloud</groupId>
  <artifactId>spring-cloud-starter-eureka</artifactId>
</dependency>
```

- 在主类中通过加上`@EnableDiscoveryClient`注解， 激活 Eureka 中的DiscoveryClient 实现(自动化配置， 创建 DiscoveryClient 接口针对 Eureka 客户端的 EurekaDiscoveryClient 实例)
- 通过注入 DiscoveryClient对象， 在日志中打印出服务（RequsetMapping方法中）的相关内容 @Autowired private DiscoveryClient client; ServiceInstance instance = client.getLocalServiceinstance();

```
# application.properties
# 服务命名
spring.application.name=hello-service
# 指定注册中心地址
eureka.client.serviceUrl.defaultZone=http://localhost:1110/eureka

# 访问 http://localhost:8080/sayhello
```

服务消费者

- 发现服务以及消费服务
- 服务发现的任务由E田eka的客户端完成，而服务消费的任务由ribbon完成
  - Ribbon是一 个基于HTTP和TCP的客户端负载均衡器，它可以在通过客户端中配置的ribbonServerList 服务端列表去轮询访问以达到均衡负载的作用
  - 当Ribbon与Eureka联合使用时，Ribbon的服务实例清单RibbonServerList会被DiscoveryEnabledNIWSServerList重写， 扩展成从Eureka注册中心中获取服务端列表。
  - 同时它也会用 NIWSDiscoveryPing来取代IPing, 它将职责委托给Eureka 来确定服务端是否已经启动
- 先启动注册中心，然后两个服务提供者

```
java -jar hello-service-0.0.1-SNAPSHOT.jar --server.port=8081
java -jar hello-service-0.0.1-SNAPSHOT.jar --server.port=8082
```

- 创建消费者

```
<dependency>
  <groupId>org.springframework.cloud</groupId>
  <artifactId>spring-cloud-starter-ribbon</artifactId>
</dependency>
<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-web</artifactId>
</dependency>
<dependency>
  <groupId>org.springframework.cloud</groupId>
  <artifactId>spring-cloud-starter-eureka</artifactId>
</dependency>
```

- 通过`@EnableDiscoveryClient`注解主类，让该应用注册为 Eureka 客户端应用， 以获得服务发现的能力。 同时， 在该主类中创建 `RestTemplate` 的 Spring Bean 实例，并通过 `@LoadBalanced` 注解开启客户端 负载均衡

```
@Bean
@LoadBalanced
RestTemplate restTemplate() {return new RestTemplate();}
```

- 创建ConsumerController类并实现/ribbon-consumer接口。 在该接口中，通过在上面创建的RestTemplate来实现对HELL-SERVICE服务提供的/hello接口进行调用

```
@RestController
public class ConsumerController {
  @Autowired
  RestTemplate restTemplate;
  @RequestMapping(value = "/ribbon-consumer", method = RequestMethod.GET)
  public String helloConsumer() {
  //访问地址是服务名，而不是具体的地址
  return restTemplate.getForEntity("http://HELLO-SERVICE/hello",
String.class).getBody();
}
```

```
# application.properties
spring.application.name=ribbon-consumer
server.port=9000
# 配置服务注册中心
eureka.client.serviceUrl.defaultZone=http://localhost:1110/eureka/
```

- 访问`http://localhost:9000/ribbon-consumer` 发起 GET 请求

基础架构

- 服务注册中心： Eureka提供的服务端， 提供服务注册与发现的功能 eureka-server
- 服务提供者：提供服务的应用， 可以是 Spring Boot 应用， 也可以是其他技术平台且遵循 Eureka 通信机制的应用。它将自己提供的服务注册到 Eureka, 以供其他应用发现HELLO-SERVICE
- 服务消费者：消费者应用从服务注册中心获取服务列表， 从而使消费者可以知道去何处调用其所需要的服务

服务治理机制

- 服务提供者
  - 服务注册
    - “服务提供者” 在启动的时候会通过发送REST请求的方式将自己注册到EurekaServer上，同时带上了自身服务的一些元数据信息
    - Eureka Server接收到这个REST请求之后，将元数据信息存储在一个双层结构Map中，其中第一层的key是服务名， 第二层的key是 具体服务的实例名
    - 在服务注册时， 需要确认一 下 eureka.cli ent.register-with-eureka=true参数是否正确， 该值默认为true。 若设置为false将不会 启动注册操作
  - 服务同步
    - 如果两个服务提供者分别注册到了两个不同的服务注册中心上， 由于服务注册中心之间因互相注册为服务， 当服务提供者发送注册请求到一个服务注册中心时， 它会将该请求转发给集群中相连的其他注册中心， 从而实现注册中心之间的服务同步
    - 通过服务同步，两个服务提供者的服务信息就可以通过这两台服务注册中心中的任意一

台获取到

- 服务续约
  - 在注册完服务之后，服务提供者会维护一个心跳用来持续告诉EurekaServer: "我还活 着”， 以防止Eureka Server 的 “ 剔除任务 ” 将该服务实例从服务列表中排除出去

```
# 服务续约任务的调用间隔时间，默认为30秒
eureka.instance.lease-renewal-interval-in-seconds=30
# 服务失效的时间， 默认为90秒
eureka.instance.lease-expiration-duration-in-seconds=90
```

- 服务消费者
  - 获取服务
    - 启动服务消费者的时候， 它会发送一 个REST请求给服务注册中心，来获取上面注册的服务清单 。 为了性能考虑， Eureka Server会维护一份只读的服务清单来返回给客户端，同时该缓存清单会每隔30秒更新一次
    - 确保`eureka.client.fetch-registry=true`参数没有被修改成false, 该值默认为true
    - 缓存清单的 更新时间`eureka.client.registry-fetch-interval-seconds=30`
  - 服务调用
    - 服务消费者在获取服务清单后，通过服务名可以获得具体提供服务的实例名和该实例的元数据信息。
    - 因为有这些服务实例的详细信息， 所以客户端可以根据自己的需要决定具体调用哪个实例，在Ribbon中会默认采用轮询的方式进行调用，从而实现客户端的负载均衡
    - 对于访问实例的选择，Eureka中有Region和Zone的概念
      - 一个Region中可以包含多个Zone, 每个服务客户端需要被注册到一个Zone中， 所以每个客户端对应一个Region和一个Zone
      - 在进行服务调用的时候，优先访问同处一个Zone中的服务提供方， 若访问不到，就访问其他的Zone
  - 服务下线
    - 在客户端程序中， 当服务实例进行正常的关闭操作时， 它会触发一个服务下线的REST请求给Eureka Server, 告诉服务注册中心:“我要下线了”。 服务端在接收到请求之后， 将该服务状态置为下线(DOWN), 并把该下线事件传播出去
- 服务注册中心
  - 失效剔除
    - 为了从服务列表中将这些无法提供服务的实例剔除， Eureka Sre vre 在启动的时候会创建一个定时任务，默认每隔一段时间(默认为60秒) 将当前清单中超时(默认为90秒)没有续约的服务剔除出去
  - 自我保护
    - 服务注册到EurekaSrevre 之后，会维护一个心跳连接，告诉EurekaServer自己还活着
    - Eurke aServer在运行期间，会统计心跳失败的比例在15分钟之内是否低于85%, 如果出现低于的情况(在单机调试的时候很容易满足， 实际在生产环境上通常是由于网络不稳定导致)， EurekaServer会将当前的实例注册信息保护起来， 让这些实例不会过期， 尽可能保护这些注册信息
    - 但是， 在这段保护期间内实例若出现问题， 那么客户端很容易拿到实际已经不存在的服

务实例， 会出现调用失败的清况， 所以客户端必须要有容错机制， 比如可以使用请求重试、 断路器等机制
    - 本地进行开发的时候， 可以使用`eureka.server.enable-self-preservation=false`参数来关闭保护机制， 以确保注册中心可以将不可用的实例正确剔除

源码分析

- DiscoveryClient 是Spring Cloud的接口，它定义了用来发现服务的常用抽象方法， 通过该接口可以有效地屏蔽服务治理的实现细节， 所以使用 Spring Cloud 构建的微服务应用可以方便地切换不同服务治理框架， 而不改动程序代码， 只需要另外添加一些针对服务治理框架的配置即可
- EurekaDiscoveryClient是对该接口的实现，从命名来判断，它实现的是对 Eureka 发现服务的封装

```
//DiscoveryClient类，用于帮助与Eureka Server互相协作
//Eureka Client负责下面的任务：
        //-向Eureka Server注册服务实例
  //-向Eureka Server服务租约
  //-当服务关闭期间， 向Eureka Server取消租约
  //-查询Eureka Server中的服务实例列表
  //Eureka Client还需要配置一个Eureka Server的 URL列表。
```

配置详解

- 在Eureka的服务治理体系中， 主要分为服务端与客户端两个不同的角色， 服务端为服务注册中心， 而客户端为各个提供接口的微服务应用
- Eureka客户端的配置对象存在于所有Eureka服务治理体系下的应用实例中
    - 服务注册相关的配置信息， 包括服务注册中心的地址、 服务获取的间隔时间、 可用区域等
    - 服务实例相关的配置信息， 包括服务实例的名称、IP地址、 端口号、 健康检查路径等
    - `org.springframework.cloud.netflix.eureka.server.EurekaServerConfigBean`
- 指定注册中心
    - 配置文件中指定注册中心， 主要通过eureka.client.serviceUrl 参数实现
    - 它的配置值存储在HashMap类型中，并且设置有一组默认值，默认值的key为defaultZone、value 为http://localhost:8761/eureka/
- 其他配置

```
# eureka.client 为前缀
enabled 启用Eureka客户端 true
registryFetchIntervalSeconds  从Eureka服务端获取注册信息的间隔时间(秒)， 30
instancelnfoReplicationlntervalSeconds  更新实例信息的变化到Eureka服务端的间隔时间，30
...
```

- 元数据

- 它是Eureka 客户端在向服务注册中心发送注册请求时，用来描述自身服务信息的对象， 其中包含了一些标准化的元数据， 比如服务名称、实例名称、 实例IP、实例端口等用于服务治理的重要信息;以及一些用 千负载均衡策略或是其他特殊用途的自定义 元数据信息
- 可以通过eureka.instance.=的格式对标准化 元数 据直接进行配置， 其中就是EurekainstanceConfigBean对象中的成员变量名
- 对于自定义 元数据， 可以通过 eureka.instance.metadataMap.=的格式来进行配置

  `eureka.instance.metadataMap.zone=shanghai`
- 实例名
  - 即Instanceinfo 中的instanceId参数， 它是区分同一服务中不同实例的唯一 标识
  - 实例名的默认命名规则 `${spring.cloud.client.hostname}:${spring.application.name}:${spring.application .instance—id:${server.port}}`
  - 实例名的命名 规则，可以通过eureka.instance.instanceId参数来进行配置
  - 启动同一服务的多个实例
    - 可以直接通过设置server.port=0 或者使用随机数 server.port=${randorn.int[10000,19999]} 来 让Tomcat 启动的时候采用随机端口，注册到 Eureka Server 的实例名都是相同的，使得只有一个服务实例能够正常提供服务
    - 通过设置实例名规则来轻松解决：`eureka.instance.instanceid=${spring.application.name}:${random.int}`
    - 利用应用名加随机数的方式来区分不同的实例， 从而实现在同一主机上， 不指定端口就能轻松启动多个实例的效果
- 端点配置
  - 如果为应用设置了 context-path，这时， 所有 spring-boot-actuator 模块的监控端点都会增加一个前缀

```
management.context-path=/hello
eureka.instance.statusPageUrlPath=${management.context-path}/info
eureka.instance.healthCheckUrlPa七h=${management.context-path}/health

# 有时候为了安全考虑， 也有可能会修改/info 和/health 端点的原始路径。
endpoints.info.path=/appinfo endpoints.health.path=/checkHealth
eureka.instance.statusPageOrlPath=/${endpoints.info.path}
eureka.instance.healthCheckOrlPath=/${endpoints.health.path}

# 绝对路径的配置参数
eureka.instance.statusPageUrl=https://${eureka.instance.hostname}/info
eureka.instance.healthCheckUrl=https://${eureka.instance.hos七
name}/health
eureka.instance.homePageUrl=https://${eureka.instance.hostname}/
```

- 健康检测

- 默认情况下，Eureka中各个服务实例的健康检测并不是通过spring-boot-actuator模块的/health 端点来实现的， 而是依靠客户端心跳的方式来保持服务实例的存活
- 默认的心跳实现方式可以有效检查客户端进程是否正常运作， 但却无法保证客户端应用能够正常提供服务
  - 由于大多 数微服务应用都会有一些其他的外部资源依赖， 比如数据库、 缓存、 消息代理等，如果我 们的应用与这些外部资源无法联通的时候， 实际上已经不能提供正常的对外服务了， 但是因为客户端心跳依然在运行，所以它还是会被服务消费者调用， 而这样的调用实际上并不能获得预期的结果
  - 以通过简单的配置， 把Eureka客户端的健康检测交给spring-boot-actuator模块的/health端点， 以实现更加全面的健康状态维护
    - `spring-boot-starter-actuator`
    - 增加参数配置 `eureka.client.healthcheck.enabled=true`
    - 让服务注册中心可以正确访问到健康检测端点