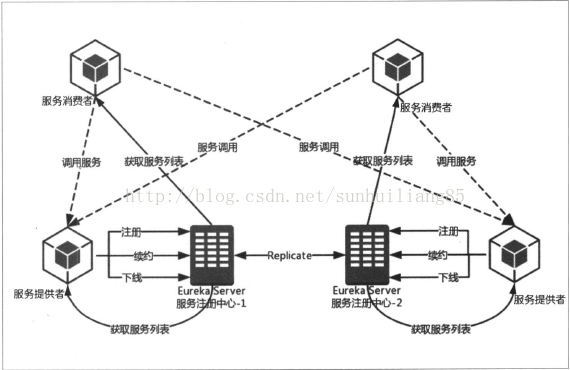
# Spring Cloud Eureka

## 1.1 服务治理

服务治理是微服务架构中最为核心和基础的模块，它主要用来实现各个微服务实例的自动化注册和发现。

Spring Cloud Eureka是Spring Cloud Netflix微服务套件中的一部分，它基于Netflix Eureka做了二次封装。主要负责完成微服务架构中的服务治理功能。

Eureka服务治理体系如下：



## 1.2 服务注册

在服务治理框架中，通常都会构建一个注册中心，每个服务单元向注册中心登记自己提供的服务，包括服务的主机与端口号、服务版本号、通讯协议等一些附加信息。注册中心按照服务名分类组织服务清单，同时还需要以心跳检测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，以达到排除故障服务的效果。

## 1.3 服务发现

在服务治理框架下，服务间的调用不再通过指定具体的实例地址来实现，而是通过服务名发起请求调用实现。服务调用方通过服务名从服务注册中心的服务清单中获取服务实例的列表清单，通过指定的负载均衡策略取出一个服务实例位置来进行服务调用。

## 1.4 Netflix Eureka介绍

Spirng Cloud Eureka使用Netflix Eureka来实现服务注册与发现。它既包含了服务端组件，也包含了客户端组件，并且服务端与客户端均采用java编写，所以Eureka主要适用于通过java实现的分布式系统，或是JVM兼容语言构建的系统。Eureka的服务端提供了较为完善的REST API，所以Eureka也支持将非java语言实现的服务纳入到Eureka服务治理体系中来，只需要其他语言平台自己实现Eureka的客户端程序。目前.Net平台的Steeltoe、Node.js的eureka-js-client等都已经实现了各自平台的Ereka客户端组件。

## 1.5 Eureka服务端

Eureka服务端，即服务注册中心。它同其他服务注册中心一样，支持高可用配置。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。

Eureka服务端支持集群模式部署，当集群中有分片发生故障的时候，Eureka会自动转入自我保护模式。它允许在分片发生故障的时候继续提供服务的发现和注册，当故障分配恢复时，集群中的其他分片会把他们的状态再次同步回来。集群中的的不同服务注册中心通过异步模式互相复制各自的状态，这也意味着在给定的时间点每个实例关于所有服务的状态可能存在不一致的现象。

## 1.6 Eureka客户端

Eureka客户端，主要处理服务的注册和发现。客户端服务通过注册和参数配置的方式，嵌入在客户端应用程序的代码中。在应用程序启动时，Eureka客户端向服务注册中心注册自身提供的服务，并周期性的发送心跳来更新它的服务租约。同时，他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行的刷新服务状态。

## 1.7服务注册中心功能概述

在服务治理框架中，通常都会构建一个注册中心，每个服务单元向注册中心登记自己提供的服务，包括服务的主机与端口号、服务版本号、通讯协议等一些附加信息。注册中心按照服务名分类组织服务清单，同时还需要以心跳检测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，以达到排除故障服务的效果。

## 1.8创建Eureka Server服务

创建一个Spring Boot工程，新建项目eureka-server-100，并在pom文件中引入依赖：

<parent>  
 <groupId>org.springframework.boot</groupId>  
 <artifactId>spring-boot-starter-parent</artifactId>  
 <version>2.1.6.RELEASE</version>  
 <relativePath /> <!-- lookup parent from repository -->  
</parent>

<dependencies>  
 <dependency>  
 <groupId>org.springframework.cloud</groupId>  
 <artifactId>spring-cloud-starter-netflix-eureka-server</artifactId>  
 </dependency>  
  
 <dependency>  
 <groupId>org.springframework.boot</groupId>  
 <artifactId>spring-boot-starter-test</artifactId>  
 <scope>test</scope>  
 </dependency>  
</dependencies>

<dependencyManagement>  
 <dependencies>  
 <dependency>  
 <groupId>org.springframework.cloud</groupId>  
 <artifactId>spring-cloud-dependencies</artifactId>  
 <version>${spring-cloud.version}</version>  
 <type>pom</type>  
 <scope>import</scope>  
 </dependency>  
 </dependencies>  
</dependencyManagement>

### 1.9 Eureka Server相关配置

在默认配置下，Eureka Server会将自己也作为客户端来尝试注册自己，我们需要禁用它的客户端禁用行为。

**server**:  
 **port**: 8761  
  
**eureka**:  
 **instance**:  
 **hostname**: localhost  
 **client**:  
 **serviceUrl**:  
 **defaultZone**: http://${**eureka.instance.hostname**}:${**server.port**}/eureka/  
 **register-with-eureka**: **false  
 fetch-registry**: **false**

### 2.0 启动服务注册中心

通过@EnableEurekaServer注解启动服务注册中心，如下：

package org.learning.eureka;  
  
import org.springframework.boot.SpringApplication;  
import org.springframework.boot.autoconfigure.EnableAutoConfiguration;  
import org.springframework.cloud.netflix.eureka.server.EnableEurekaServer;  
  
@EnableAutoConfiguration  
@EnableEurekaServer  
public class EurekaApplication {  
  
 public static void main(String[] args) {  
 SpringApplication.*run*(EurekaApplication.class, args);  
 }  
  
}

## 2.1 高可用服务注册中心

## 2.2 高可用服务注册中心的概念

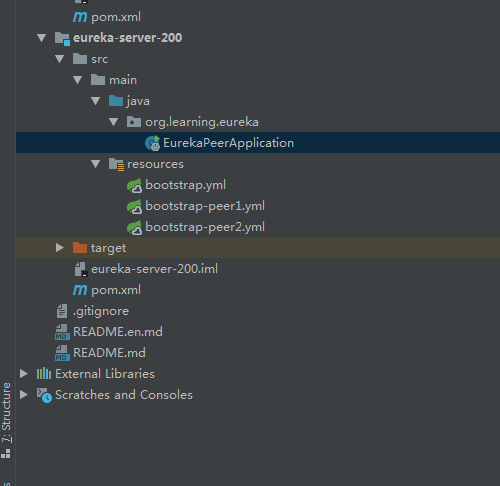
考虑到发生故障的情况，服务注册中心发生故障必将会造成整个系统的瘫痪，因此需要保证服务注册中心的高可用。

Eureka Server在设计的时候就考虑了高可用设计，在Eureka服务治理设计中，所有节点既是服务的提供方，也是服务的消费方，服务注册中心也不例外。

Eureka Server的高可用实际上就是将自己做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。

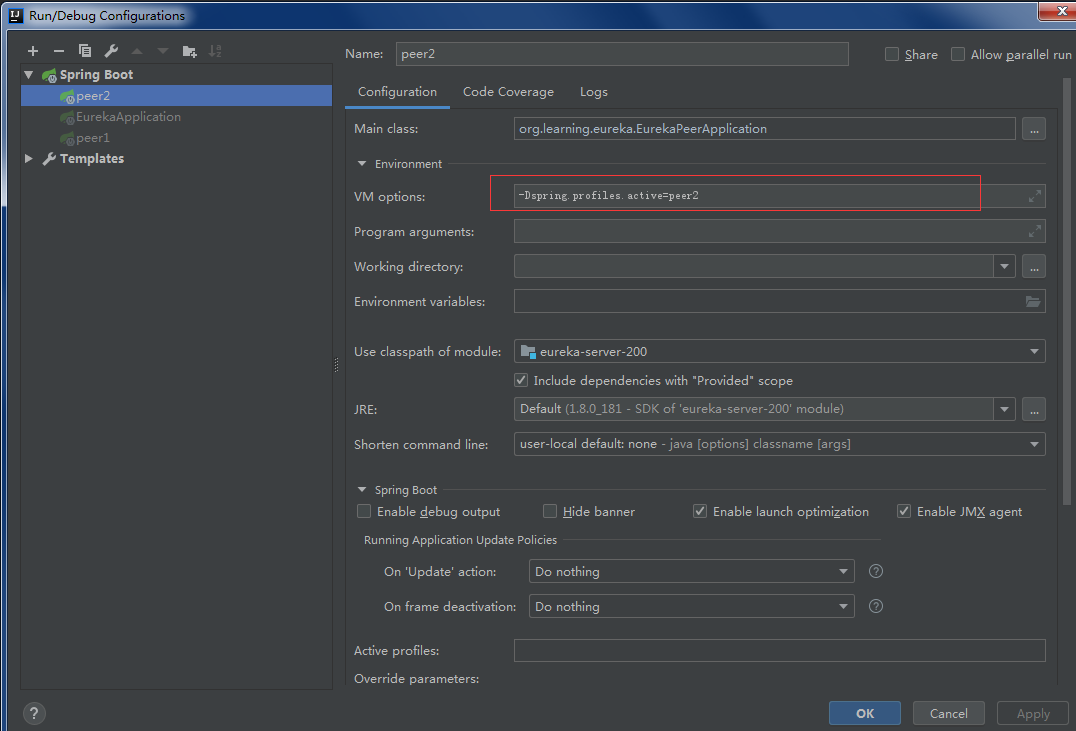
## 2.3 构建服务注册中心集群

Eureka Server的同步遵循着一个非常简单的原则：只要有一条边将节点连接，就可以进行信息传播与同步。可以采用两两注册的方式实现集群中节点完全对等的效果，实现最高可用性集群，任何一台注册中心故障都不会影响服务的注册与发现。



**server**:  
 **port**: 8762  
**eureka**:  
 **instance**:  
 **hostname**: www.peer1.com  
 **client**:  
 **serviceUrl**:  
 **defaultZone**: http://www.peer2.com:8763/eureka/

**server**:  
 **port**: 8763  
**eureka**:  
 **instance**:  
 **hostname**: www.peer2.com  
 **client**:  
 **serviceUrl**:  
 **defaultZone**: http://www.peer1.com:8762/eureka/



记得host文件添加对应域名映射

## 2.4 失效剔除

有些时候，我们的服务实例并不一定会正常下线，可能由于内存溢出、网络故障等原因使服务不能正常运作。而服务注册中心并未收到“服务下线”的请求，为了从服务列表中将这些无法提供服务的实例剔除，Eureka Server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。

## 2.5 自我保护

服务注册到Eureka Server后，会维护一个心跳连接，告诉Eureka Server自己还活着。Eureka Server在运行期间会统计心跳失败的比例在15分钟以之内是否低于85%，如果出现低于的情况，Eureka Server会将当前实例注册信息保护起来，让这些实例不会过期。这样做会使客户端很容易拿到实际已经不存在的服务实例，会出现调用失败的情况。因此客户端要有容错机制，比如请求重试、断路器。

以下是自我保护相关的属性：

eureka.server.enableSelfPreservation=true. 可以设置改参数值为false，以确保注册中心将不可用的实例删除

## 2.6 region与zone

region和zone（或者Availability Zone）均是AWS的概念。在非AWS环境下，我们可以简单地将region理解为地域，zone理解成机房。一个region可以包含多个zone，可以理解为一个地域内的多个不同的机房。不同地域的距离很远，一个地域的不同zone间距离往往较近，也可能在同一个机房内。

region可以通过配置文件进行配置，如果不配置，会默认使用us-east-1。同样Zone也可以配置，如果不配置，会默认使用defaultZone。

Eureka Server通过eureka.client.serviceUrl.defaultZone属性设置Eureka的服务注册中心的位置。

指定region和zone的属性如下：

（1）eureka.client.availabilityZones.myregion=myzone# myregion是region

（2）eureka.client.region=myregion

     Ribbon的默认策略会优先访问通客户端处于同一个region中的服务端实例，只有当同一个zone中没有可用服务端实例的时候才会访问其他zone中的实例。所以通过zone属性的定义，配合实际部署的物理结构，我们就可以设计出应对区域性故障的容错集群。

## 2.7 服务注册

服务提供者在启动的时候会通过REST请求的方式将自己注册到Eureka Server上，同时带上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的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.client.register-with-eureka=true参数是否正确，该值默认为true。若设置为fasle将不会启动注册操作。

## 2.8 服务同步

从eureka服务治理体系架构图中可以看到，不同的服务提供者可以注册在不同的服务注册中心上，它们的信息被不同的服务注册中心维护。

此时，由于多个服务注册中心互相注册为服务，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，从而实现服务注册中心之间的服务同步。通过服务同步，提供者的服务信息就可以通过集群中的任意一个服务注册中心获得。

## 2.9 服务续约

在注册服务之后，服务提供者会维护一个心跳用来持续高速Eureka Server，“我还在持续提供服务”，否则Eureka Server的剔除任务会将该服务实例从服务列表中排除出去。我们称之为服务续约。

下面是服务续约的两个重要属性：

（1）eureka.instance.lease-expiration-duration-in-seconds

leaseExpirationDurationInSeconds，表示eureka server至上一次收到client的心跳之后，等待下一次心跳的超时时间，在这个时间内若没收到下一次心跳，则将移除该instance。

默认为90秒

如果该值太大，则很可能将流量转发过去的时候，该instance已经不存活了。

如果该值设置太小了，则instance则很可能因为临时的网络抖动而被摘除掉。

该值至少应该大于leaseRenewalIntervalInSeconds

（2）eureka.instance.lease-renewal-interval-in-seconds

leaseRenewalIntervalInSeconds，表示eureka client发送心跳给server端的频率。如果在leaseExpirationDurationInSeconds后，server端没有收到client的心跳，则将摘除该instance。除此之外，如果该instance实现了HealthCheckCallback，并决定让自己unavailable的话，则该instance也不会接收到流量。

## 3.0 获取服务

消费者服务启动时，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注册清单来返回给客户端，同时该缓存清单默认会每隔30秒更新一次。

下面是获取服务的两个重要的属性：

（1）      eureka.client.fetch-registry

是否需要去检索寻找服务，默认是true

（2）eureka.client.registry-fetch-interval-seconds

表示eureka client间隔多久去拉取服务注册信息，默认为30秒，对于api-gateway，如果要迅速获取服务注册状态，可以缩小该值，比如5秒

## 5.2 服务调用

服务消费者在获取服务清单后，通过服务名可以获取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。

## 5.3 服务下线

在系统运行过程中必然会面临关闭或重启服务的某个实例的情况，在服务关闭操作时，会触发一个服务下线的Rest服务请求给Eureka Server，告诉服务注册中心：“我要下线了。”服务端在接收到该请求后，将该服务状态置位下线（DOWN），并把该下线事件传播出去。

## 5.4 端点配置

eureka实例的状态页面和健康监控的url默认为spring boot actuator提供的/info端点和/health端点。我们必须确保Eureka客户端的/health端点在发送元数据的时候，是一个能够被注册中心访问到的地址，否则服务注册中心不会根据应用的健康检查来更改状态（仅当开启了healthcheck功能时，以该端点信息作为健康检查标准）。而如果/info端点不正确的话，会导致在Eureka面板中单击服务时，无法访问到服务实例提供的信息接口。

大多数情况下，我们不需要修改这个几个url配置。但是当应用不使用默认的上下文(context path或servlet path，比如配置server.servletPath=/test），或者管理终端路径（比如配置management.contextPath=/admin）时，我们需要修改健康检查和状态页的url地址信息。

application.yml配置文件如下：

server.context-path=/helloeureka

//下面配置为相对路径，也支持配置成绝对路径，例如需要支持https

eureka.instance.health-check-url-path=${server.context-path}/health

eureka.instance.status-page-url-path=${server.context-path}/info

## 5.5 元数据

元数据是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；以及一些用于负载均衡策略或是其他特殊用途的自定义元数据信息。

我们可以通过eureka.instance.<properties>=<value>的格式对标准化元数据直接进行配置，其中<properties>就是EurekaInstanceConfigBean对象中的成员变量。而对于自定义元数据，可以通过eureka.instance.metadataMap.<key>=<value>的格式来进行配置。比如：

eureka.instance.metadataMap.zone=tianjin

//随机生成实例名

eureka.instance.metadataMap.instanceId=${spring.application.name}:${random.value}

## 5.6 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-acturator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。在Eureka的服务续约与剔除机制下，客户端的健康状态从注册到注册中心开始都会处于UP状态，除非心跳终止一段时间之后，服务注册中心将其剔除。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。

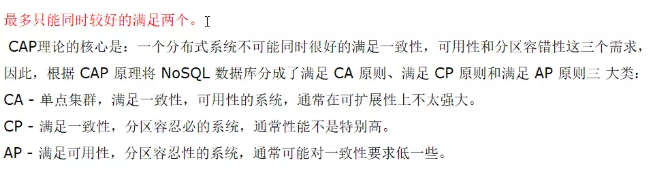
在Spring Cloud Eureka中，可以把Eureka客户端的健康检测交给spring-boot-actuator模块的health端点，以实现更加全面的健康状态维护，设置方式如下：

（1）      在pom.xml中引入spring-boot-starter-actuator模块的依赖

（2）      在application.properties中增加参数配置eureka.client.healthcheck.enabled=true

## 5.7 注册中心组件对比





* 服务的健康检查

Euraka 使用时需要显式配置健康检查支持；Zookeeper,Etcd 则在失去了和服务进程的连接情况下任务不健康，而 Consul 相对更为详细点，比如内存是否已使用了90%，文件系统的空间是不是快不足了。

* 多数据中心支持

Consul 通过 WAN 的 Gossip 协议，完成跨数据中心的同步；而且其他的产品则需要额外的开发工作来实现；

* KV 存储服务

除了 Eureka ,其他几款都能够对外支持 k-v 的存储服务，所以后面会讲到这几款产品追求高一致性的重要原因。而提供存储服务，也能够较好的转化为动态配置服务哦。

* 产品设计中 CAP 理论的取舍

Eureka 典型的 AP,作为分布式场景下的服务发现的产品较为合适，服务发现场景的可用性优先级较高，一致性并不是特别致命。其次 CA 类型的场景 Consul,也能提供较高的可用性，并能 k-v store 服务保证一致性。 而Zookeeper,Etcd则是CP类型 牺牲可用性，在服务发现场景并没太大优势；

* 多语言能力与对外提供服务的接入协议

Zookeeper的跨语言支持较弱，其他几款支持 http11 提供接入的可能。Euraka 一般通过 sidecar的方式提供多语言客户端的接入支持。Etcd 还提供了Grpc的支持。 Consul除了标准的Rest服务api,还提供了DNS的支持。

* Watch的支持（客户端观察到服务提供者变化）

Zookeeper 支持服务器端推送变化，Eureka 2.0(正在开发中)也计划支持。 Eureka 1,Consul,Etcd则都通过长轮询的方式来实现变化的感知；

* 自身集群的监控

除了 Zookeeper ,其他几款都默认支持 metrics，运维者可以搜集并报警这些度量信息达到监控目的；

* 安全

Consul,Zookeeper 支持ACL，另外 Consul,Etcd 支持安全通道https.

* Spring Cloud的集成

目前都有相对应的 boot starter，提供了集成能力。

总的来看，目前Consul 自身功能，和 ***[spring](http://www.voidcn.com/article/javascript:void()" \t "http://www.voidcn.com/article/_blank)*** cloud 对其集成的支持都相对较为完善，而且运维的复杂度较为简单（没有详细列出讨论），Eureka 设计上比较符合场景，但还需持续的完善。

## 5.8 Ribbon和nginx的区别

服务器端负载均衡 Nginx

nginx 是客户端所有请求统一交给 nginx，由 nginx 进行实现负载均衡请求转发，属于服务器端负载均衡。

既请求由 nginx 服务器端进行转发。

客户端负载均衡 Ribbon

Ribbon 是从 eureka 注册中心服务器端上获取服务注册信息列表，缓存到本地，然后在本地实现轮询负载均衡策略。

既在客户端实现负载均衡。

\*\* 应用场景的区别：\*\*

Nginx 适合于服务器端实现负载均衡 比如 Tomcat ，Ribbon 适合与在微服务中 RPC 远程调用实现本地服务负载均衡，比如 Dubbo、SpringCloud 中都是采用本地负载均衡。

补充：ribbon和feign的区别（转）

spring cloud的Netflix中提供了两个组件实现软负载均衡调用：ribbon和feign。

Ribbon

是一个基于 HTTP 和 TCP 客户端的负载均衡器

它可以在客户端配置 ribbonServerList（服务端列表），然后轮询请求以实现均衡负载。

Feign

Spring Cloud Netflix 的微服务都是以 HTTP 接口的形式暴露的，所以可以用 Apache 的 HttpClient 或 Spring 的 RestTemplate 去调用，而 Feign 是一个使用起来更加方便的 HTTP 客戶端，使用起来就像是调用自身工程的方法，而感觉不到是调用远程方法。

注意：spring-cloud-starter-feign 里面已经包含了 spring-cloud-starter-ribbon（Feign 中也使用了 Ribbon）