Eureka服务注册与发现

**背景**

众所周知，服务发现必须同时具备高可用性和高弹性。

先举个例子，当调用api或者发起网络通信的时候，我们需要知道被调用方的服务IP和端口或者通过域名和端口，事实上这种方法就是基于DNS的服务发现。但由于DNS缓存、无法自治和其它不利因素的存在，目前该机制还存在很多问题和缺陷。传统的DNS方式都是通过nginx或者其它代理来实现，物理机的ip和port都是固定的，所以nginx中配置的ip和port也就是固定的，此时服务列表更新只能通过人工来做。但如果后端服务很多，手动更新不仅效率很低，而且还容易出错。并且在后端发生故障时，服务不可用时间就可能加长。甚至在一些服务中(如Docker)，ip、port和服务实例数都是动态变化的，所以就需要精细而准确的服务发现机制。

Eureka就是为服务发现所设计的。

**Eureka是什么？**

Eureka是Netflix开源的一个RESTful(代表性状态传输)服务，主要用于服务注册与发现。Eureka由两个组件组成：Eureka服务器和Eureka客户端。Eureka服务器用作服务注册服务器；Eureka客户端是一个java客户端，用来简化与服务器的交互、作为轮询负载均衡器，并提供服务的故障切换支持。其主要特点如下：

* 主要特点
  + 开源
  + 可靠
  + 功能齐全
  + 基于Java
  + Spring Cloud集成

**Eureka的几个重要概念**

**1. Register(服务注册)**

当Eureka客户端向Eureka Server注册时，它提供自身的元数据，如ip、port、运行状况、URL和主页等。

**2. Renew(服务续约)**

Eureka Client每隔30秒发送一次心跳来完成续约。 Eureka Client通过续约来告知Eureka Server自己仍然存在，没有任何问题。正常情况下，如果Eureka Server在90秒内没有收到Eureka Client的续约，那么它会将实例从其注册表中删除。 此间隔时间建议用户不要更改。

**3. Get Registry(获取注册表信息)**

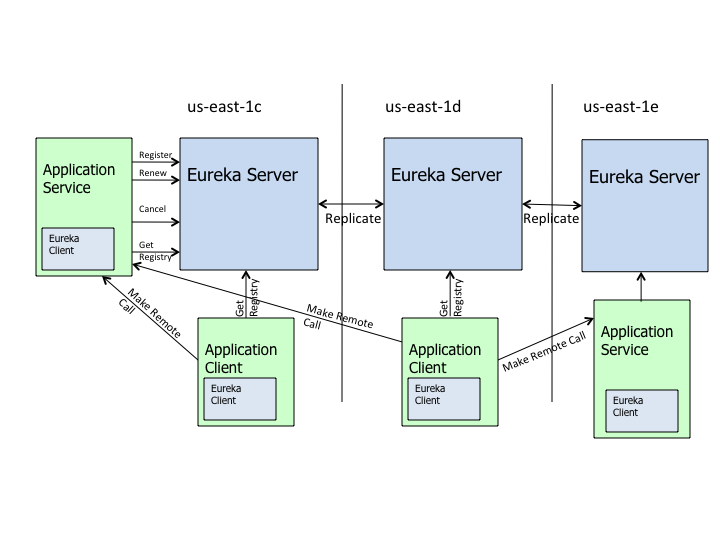
Eureka Client从Eureka Server获取注册表信息，并将其缓存到本地。Eureka Client和Eureka Server间支持JSON/XML格式进行通讯。默认压缩JSON格式。

**4. Cancel(服务下线)**

Eureka Client在程序关闭时向Eureka服务器发送取消请求。发送请求后，该客户端实例信息将从服务器的实例注册表中删除。

**Eureka高可用架构**

下图是Eureka官方提供的架构图：



由图可以看出，架构中主要3种角色：

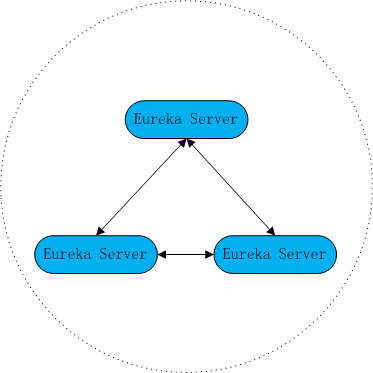
* 角色
  + Eureka Server：通过Register、Get、Renew等接口提供服务注册和发现功能
  + Application Service：服务提供方，把自身服务实例注册到Eureka Server
  + Application Client：服务调用方，通过Eureka Server获取服务实例，并调用Application Service

**处理流程**

1. 每个区域有一个Eureka Cluster，每个区域的可用区中至少有一个Eureka Server。
2. Application Service作为Eureka Client，通过register注册到Eureka Server，并通过发送心跳的方式进行续约，如果90秒内没有进行续约，Eureka Server将会移除该service实例。
3. 当一个Eureka Server收到数据或数据发生改变后，会将自己的数据同步到其它Eureka Server中。
4. Application Client也作为Eureka Client通过get接口从Eureka Server中获取Application Service实例信息。
5. Application Client通过模块内的负载均衡机制调用Application Service，同时可以跨区域调用。

Application Client可以从Eureka Server获取到Application Service的注册信息表，并缓存到本地。之后，客户端使用该信息查找其它Application service，这些信息通过获取最后一次读取周期和当前读取周期之间的增量信息来进行每30秒一次的定期更新，增量信息在Eureka Server中保存较长时间，约3分钟。因此，增量获取可能会再次返回相同的实例，Eureka Client会自动处理重复的信息。获得增量信息后，Eureka Client通过比较Eureka Server返回的实例数来与Eureka Server进行协调，如果信息由于某种原因不匹配，则会重新提取整个注册表信息。来自Eureka Client客户端的所有操作可能都需要一段时间才能反映在Eureka Server中，随后在其他Eureka Server中反映。这是因为Eureka Server上的有效载荷的缓存，它定期刷新以反映新的信息。而Eureka Client也会定期获取增量信息，因此可能需要2分钟才能将更改传播给所有的Eureka Client。由于Eureka Client已将有关可用服务的信息缓存在本地，因此程序可以灵活地选择经过负载均衡处理的终端，虽然该机制会占用少量内存，但却拥有了更好的弹性。

**模式介绍**

1. Eureka Server采用的是对等通信(P2P)，一种无中心化架构，没有master/slave之分，每一个server都是对等的，既是Server又是Client，所以其集群方式可以自由发挥，可以各点互连，也可以接力互连。例如，Eureka Server间的架构可以设计如下：

Eureka Server通过运行多个实例以及彼此之间互相注册来提高可用性，每个节点需要添加一个或多个有效的serviceUrl指向另一个节点。

1. Eureka采用的是客户端发现模式，其优点是服务调用、负载均衡都不需要和Eureka Server进行通信，直接使用本地注册表副本，因此Eureka Server不可用时是不会影响正常的服务调用，性能也不会因为网络延迟和服务端延迟受到影响。但该机制也有其缺点，即当某个Application Service不可用时，Eureka Client不能及时获悉，可能需要1到3个心跳周期才能知晓。针对该问题，我们可以根据实际使用情况采用相应的熔断和降级机制进行弥补。如Netflix就是服务端使用Hystrix来容错和降级。

**Eureka搭建和部署**

**环境准备**

* VMWare虚拟机
* centos7
* Git
* Grade
* Tomcat

**步骤**

1. 安装Git、Grade和Tomcat
2. 构建Eureka
3. git clone https://github.com/Netflix/eureka.git
4. cd eureka/
5. ./gradlew clean build
6. 部署Eureka Server
7. /usr/local/tomcat/bin/shutdown.sh
8. cp eureka-server-1.1.134.war /usr/local/tomcat/webapps/eureka.war
9. /usr/local/tomcat/bin/startup.sh
10. 通过URL即可进行访问和查看相应信息。

目前，在实际工作中使用Eureka的例子很少，其在微服务和Spring cloud中应用较为广泛。因为Spring Cloud 集成了Eureka，并提供开箱即用的支持。

本文链接：<https://addops.cn/post/eureka.html?utm_source=tuicool&utm_medium=referral>