链滴

Spring 微服务之 Eureka 实践

作者：teneous

原文链接：https://ld246.com/article/1558160172058

来源网站：链滴

Spring微服务之-软负载中心

@(syoka)

序言:本篇文章不会教你如何搭建一个eureka（需要请查看spring.io官网用例），而是简单说说自己理解和遇到的一些坑

软负载中心出现的契机

如果我们有两台服务器，它们所提供的功能是一样的。我们使用Nginx等服务来该记录服务的地址和口，当有消费者想要获取服务地址时，我们将类似以下信息返回消费端

```
{
    server 192.168.1.11:8080;
    server 192.168.1.22:8080;
}
```

从这里体现出了负载中心的第一个特性，服务内聚

在内部系统或者服务少的情况下，上述方法确实能够有效的解决问题（可控制服务出入口，适用支付融业），但是面对分布式环境中对多服务管理的话这么做有两个问题:

1. 复杂:新增服务不是通过实例启动自动完成，而是需要手动维护。且直接映射物理服务，这对于Devps人员来说并不友好。

2. 单点故障：一旦负载中心挂了，其余所有消费端获取服务地址都将失败，对于此类问题，可以考虑入一个【从负载均衡器】，在主负载器出现问题时进行故障切换。

因此这里我们要解决的第一个痛点就是一个自动检查新老服务上下线的软负载中心(内聚体现:注册中心)

软负载服务内聚的构建（以下统称注册中心）

想要构建一个注册中心，首先得了解对于注册中心来说，它需要服务提供者提供什么的元数据

从文章开头的静态地址来看

，我们可以看出

- 必须提供服务端的物理地址ip以及端口

（一般为了防止直接对物理地址的操作，会再抽象一层逻辑服务名）

- 为确保服务提供者的高可用，提供者一般也是采用集群部署，因此我们使用GroupId来标识一个集群

最终从结构上来说:

我们需要一个groupid来充当一组服务的逻辑名，每组groupid里存放着每个实例的物理地址以及端口

光有结构还不够，还需了解注册中心服务框架如何与客户端结合

这里给出2钟:

1. 注册中心框架以jar包的方式和服务端一起启动，由客户端发起RESTFUL请求到注册中心服务端完注册（eureka就是采用此方案）。

2. 注册中心框架绑定到容器中。服务被在容器中启动，可以被注册中心感知。

软负载对服务状态的检测

注册中心需要感知所有服务的当前状态，防止向消费者返回已经下线或是有问题的服务地址，这个需提供端定时向注册中心汇报自己的当前状态（称为:心跳包）

当然可能因为网络延时或者当前软负载中心负载很高，注册中心不能及时收到心跳包。此时不应该直剔除服务并删除注册列表，毕竟注册一个服务所耗的资源还是很高的。因此注册中心应该有一定弹性在没有收到心跳包也应该在一定时间内保留注册信息，并另开一个线程监视器再一定时间内继续监视如还收不到心跳包，则判定下线。

软负载的性能策略

服务提供者将信息注册到注册中心，如果注册中心出问题，又没有容灾策略，那么所有消费者都获取到服务提供者的位置。因此，所有服务消费者会在本地缓存一份。

这样做的好处：

1. 注册中心挂了也可以使用本地缓存直接请求。

2. 提升了效率（不用每次请求都向注册中心获取服务地址，降低注册中心的负载，避免网络资源的浪）

客户端本地缓存:注册中心服务注册表发生变换时需要通知客户端刷新本地缓存，此处使用全量更新(现简单)，还是增量更新(实现复杂)是可以权衡的。eureka采用增量更新。

第二个痛点就是容灾策略，采用集群部署来解决

注册中心的集群化

注册中心的统一化管理主要有两种实现方式

1.通过相互注册的同步方式(eureka就是采用这种方式)。

例:A，B两个注册中心，把B当成一个服务提供者注册到A中，A也当成提供者注册在B中。这样两个注中心就可保持同步，以此类推，有N个注册中心那么，每个注册中心需要完成N-1次注册即可。

这么做的好处是可以复用功能，也易于理解。但是从侧面也反映出它的缺点，同步效率会大大降低。

为了解决效率的问题，并且接受部分服务同步延时，可以选择类似batch批量同步。

2.通过数据内聚的管理方式

将所有的地址数据聚合在一起，所有注册中心从聚合服务获取，可以保证软负载服务的无状态，但是得考虑如果聚合服务出现故障时的备用策略。

感觉自己写的很捞，后面有时间再改改吧。

参考文档:

- 大型网站系统与JAVA中间件实践（曾宪杰）
- Spring Microservices in Action-Manning Publications (John Carnell)
- SpringCloud 微服务实战（翟永超）