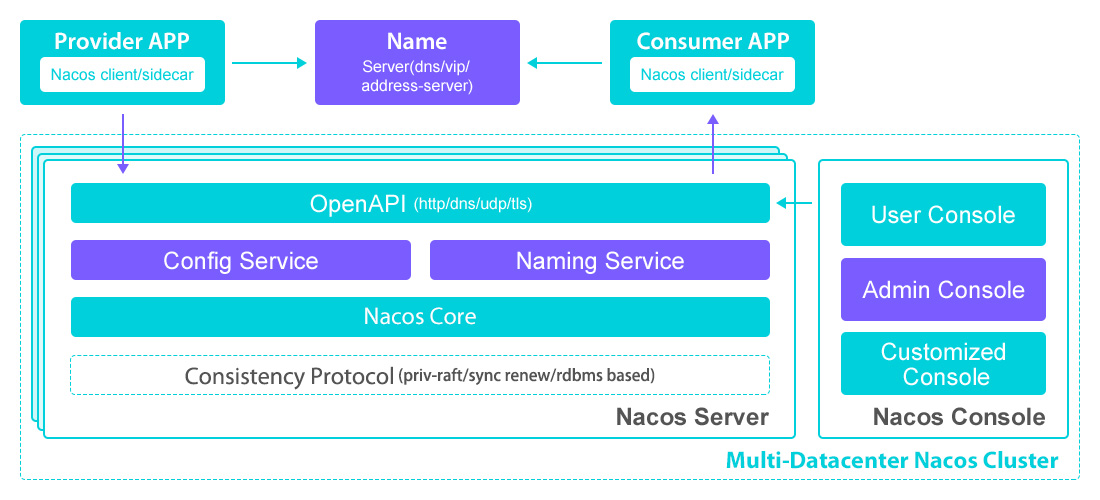
# Spring Cloud – Nacos与Eureka技术对比

## 1、背景

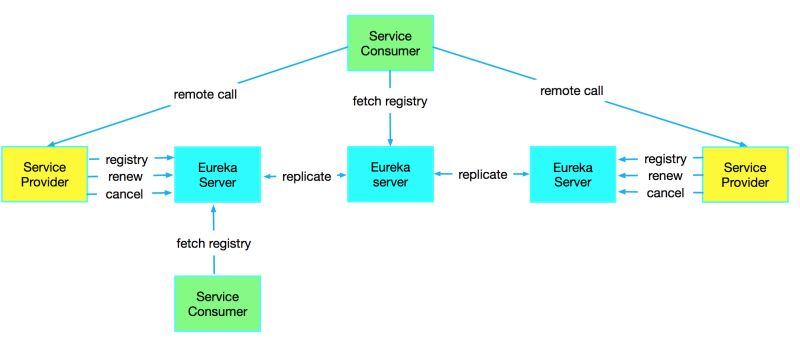
Nacos与Eureka都是微服务服务注册中心，Eureka出现于2014年，与Spring cloud生态深度结合，但于2018年7月Netflix公司决定停止开源2.X版本的Eureka，对日后的更新与维护造成很大困难。而Nacos是阿里集团于2018年开源发布的，携带了阿里大规模生产服务的经验，提供给用户一个新的选择。

## 2、服务架构图

1、Spring cloud Nacos



2、Spring cloud Eureka



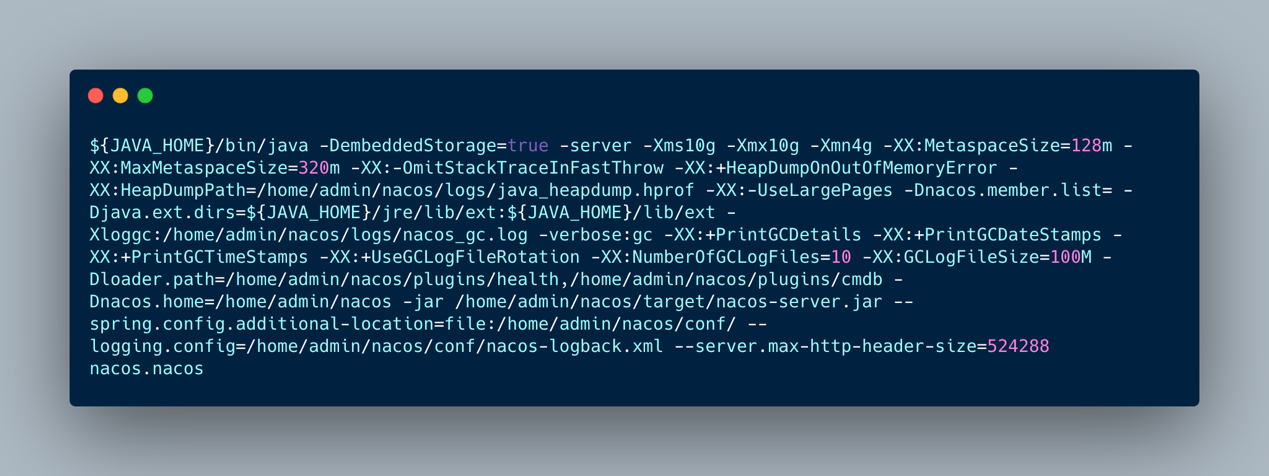
## 3、性能测试

#### 3.1、Spring cloud Nacos

##### 1、测试环境

|  |  |
| --- | --- |
| **指标** | **参数** |
| 机器 | CPU 8核，内存16G |
| 集群规模 | 三节点 |
| Nacos版本 | 服务端：Nacos2.0.0-ALPHA2， 客户端：Nacos2.0.0-ALPHA2 |

##### 2、启动参数



##### 3、测试场景

以下测试场景都是服务发现重要接口： 吞吐量（TPS），响应时间（RT）

###### 服务发现注册实例的能力

相关API：**NamingService.registerInstance(String serviceName, Instance instance)**

| **施压机数量** | **每台线程数** | **平均TPS** | **平均RT** | **最小RT** | **最大RT** | **CPU使用率** |
| --- | --- | --- | --- | --- | --- | --- |
| 1 | 100 | 7256.32 | 13.14 | 0.39 | 2522.25 | 80% |
| 2 | 50 | 16418.04 | 5.8 | 0.41 | 3906.77 | 90% |
| 5 | 20 | 26784.84 | 3.6 | 0.38 | 1606.41 | 90% |

###### 服务发现查询实例的能力

相关API：**NacosNamingService.getAllInstances(String serviceName, boolean subscribe)**

| **施压机数量** | **每台线程数** | **平均TPS** | **平均RT** | **最小RT** | **最大RT** | **CPU使用率** |
| --- | --- | --- | --- | --- | --- | --- |
| 1 | 100 | 12998.46 | 7.54 | 0.55 | 213.86 | 40% |
| 2 | 50 | 12785.01 | 7.93 | 0.38 | 900.48 | 40% |
| 2 | 100 | 18451.78 | 10.63 | 0.6 | 829.42 | 45% |
| 5 | 20 | 30680.48 | 3.12 | 0.46 | 1138.38 | 50% |

###### 服务发现注销实例的能力

相关API：**NacosNamingService.deregisterInstance(String serviceName, String ip, int port)**

| **施压机数量** | **每台线程数** | **平均TPS** | **平均RT** | **最小RT** | **最大RT** | **CPU使用率** |
| --- | --- | --- | --- | --- | --- | --- |
| 1 | 100 | 9614.96 | 9.88 | 0.41 | 1115.27 | 70% |
| 2 | 50 | 22252.07 | 4.28 | 0.39 | 856.1 | 90% -> 60% |
| 5 | 20 | 29393.8 | 2.55 | 0.42 | 741.09 | 90% -> 60% |

##### 4、结果分析

相较Nacos1.X版本，注销性能总体提升至少2倍，在服务端机能减半的情况下，服务实例数基本一致的情况下，TPS仍能做到2倍左右的提高。 关于CPU由90% 降低为 60%的场景， 是由于随着注销的的服务和实例增多，重复注销的操作变得频繁，未命中服务和实例的操作会被快速返回且操作量小，因此CPU下降、TPS相对注册略高。

##### 5、测试结论

Nacos2服务发现性能测试都是针对重点功能，通过对3节点规模集群进行压测，可以看到接口性能负载和容量，以及对比相同/类似场景下Nacos1.X版本的提升。

1. 压测时服务及实例容量达到百万级，集群运行持续稳定，达到预期；（该场景没有计算频繁变更导致的频繁推送内容，仅单纯计算容量上线，附带推送的真实场景将在下轮压测报告中给出）
2. 注册/注销实例TPS达到 26000 以上，总体较Nacos1.X提升至少2倍，接口达到预期；
3. 查询实例TPS能够达到 30000 以上，总体较Nacos1.X提升3倍左右，接口达到预期；

##### 6、参考

https://nacos.io/zh-cn/docs/nacos-naming-benchmark.html

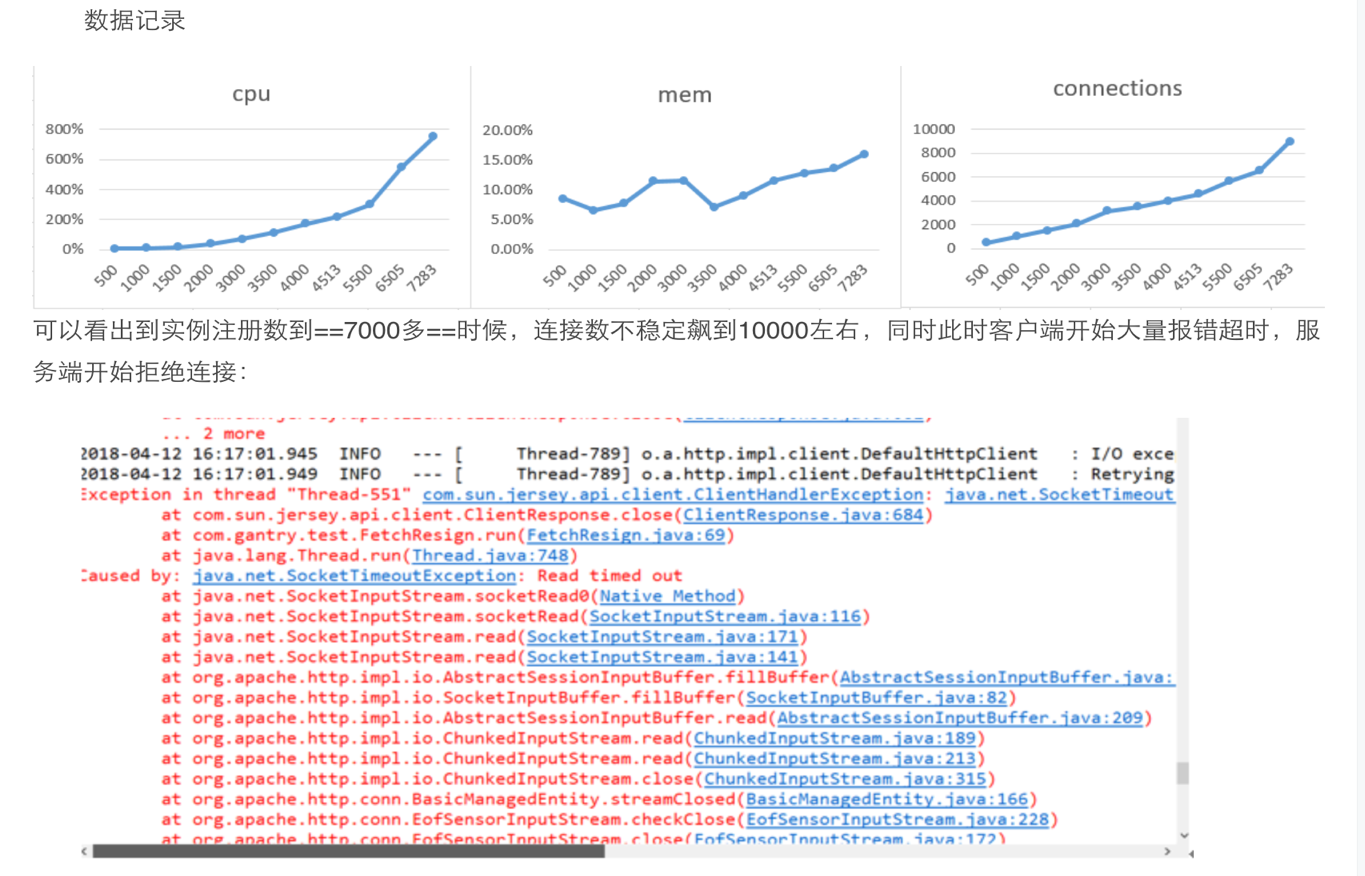
https://nacos.io/zh-cn/docs/nacos2-naming-benchmark.html

### 3.2、Spring cloud Eureka

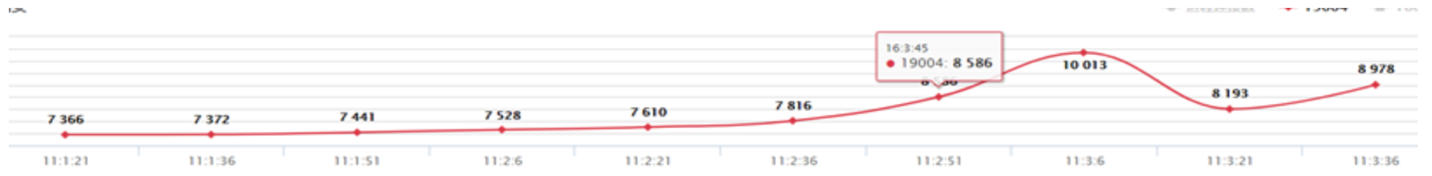
##### 1、测试环境

| **工具选项** | **描述** | **配置/能力** |
| --- | --- | --- |
| 测试机器 | CentOS release 5.4 (Final) 3台 | 8c16g，open files：65535，max user processes：65535 |
| Eureka服务端 | 单机部署,boot版本(Dalston.SR5) | -XX:MaxHeapSize=4g（默认） |
| Wireshark | Windows平台抓包工具 | 抓取HTTP、TCP等报文内容、协议相关信息 |
| UAV监控 | 公司自研监控平台 | 实时监控采集应用性能指标 |

##### 2、测试结果



此时连接数截图详细，可以清楚看到conns达到10000阀值后接着下降：





可以看出，在修改了tomcat对应配置，将最大连接数调至20000，线程数调至5000后，Eureka可注册的实例数突破了7000，连接数也突破了10000，实例数注册到8000后才开始报错，看出此时cpu已经接近满负载，操作系统本身调度已经压到极限，于是结束了本次测试。

##### 3、结论

Eureka Server服务实例注册量的负载值和操作系统、应用容器本身对应的配置相关，调整操作系统可打开最大文件句柄、进程数，调整应用容器相关最大连接数、线程数、NIO服务器模型引入等等手段都可提高我们应用服务整体吞吐量。

##### 4、参考

<http://springcloud.cn/view/31>

### 3.3、比对结果

上两种测试均在8核16G服务器上运行，从结果可看出在高负载和多注册应用上Nacos的性能更加，能容纳更多的实例，并且Eureka实例数过高时会导致CPU高负荷运载，而Nacos的测试CPU最高只处于90%状态。

## 4、优点

1. Nacos开源方为阿里，有专门的社区和教程，维护与更新及时，并带有可视化控制界面。
2. Nacos同时支持CP（一致性与分区容错性）与AP（可用性与分区容错性），根据服务注册选择临时和永久来决定走AP模式还是CP模式，而eureka只支持AP。
3. Nacos还支持管理台动态配置服务，效果如同Spring cloud Eureka + Spring cloud Config。

## 5、功能差异

| **模块** | **Nacos** | **Eureka** | **说明** |
| --- | --- | --- | --- |
| 注册中心 | ✅ | ✅ | 服务治理基本功能，负责服务中心化注册 |
| 配置中心 | ✅ | ❌ | Eureka需要配合Config实现配置中心，且不提供管理界面 |
| 动态刷新 | ✅ | ❌ | Eureka需要配合MQ实现配置动态刷新，Nacos采用Netty保持TCP长连接实时推送 |
| 可用区AZ | ✅ | ✅ | 对服务集群划分不同区域，实现区域隔离，并提供容灾自动切换 |
| 分组 | ✅ | ❌ | Nacos可用根据业务和环境进行分组管理 |
| 元数据 | ✅ | ✅ | 提供服务标签数据，例如环境或服务标识 |
| 权重 | ✅ | ❌ | Nacos默认提供权重设置功能，调整承载流量压力 |
| 健康检查 | ✅ | ✅ | Nacos支持由客户端或服务端发起的健康检查，Eureka是由客户端发起心跳 |
| 负载均衡 | ✅ | ✅ | 均提供负责均衡策略，Eureka采用Ribion |
| 管理界面 | ✅ | ❌ | Nacos支持对服务在线管理，Eureka只是预览服务状态 |

## 6、安装与使用教程

## <https://github.com/XiaoTiJun/ExperienceSharing/blob/master/JAVA/nacos%26eureka/nacos/installNacos.md>

## 附录（CAP定理）📎

CAP定理，指的是在一个分布式系统中， Consistency（一致性）、 Availability（可用性）、Partition tolerance（分区容错性），三者不可得兼。

一致性（C）：在分布式系统中的所有数据备份，在同一时刻是否同样的值。（等同于所有节点访问同一份最新的数据副本）

可用性（A）：在集群中一部分节点故障后，集群整体是否还能响应客户端的读写请求。（对数据更新具备高可用性）

分区容忍性（P）：以实际效果而言，分区相当于对通信的时限要求。系统如果不能在时限内达成数据一致性，就意味着发生了分区的情况，必须就当前操作在C和A之间做出选择。

CAP原则的精髓就是要么AP，要么CP，要么AC，但是不存在CAP。如果在某个分布式系统中数据无副本， 那么系统必然满足强一致性条件， 因为只有独一数据，不会出现数据不一致的情况，此时C和P两要素具备，但是如果系统发生了网络分区状况或者宕机，必然导致某些数据不可以访问，此时可用性条件就不能被满足，即在此情况下获得了CP系统，但是CAP不可同时满足。因此在进行分布式架构设计时，必须做出取舍。