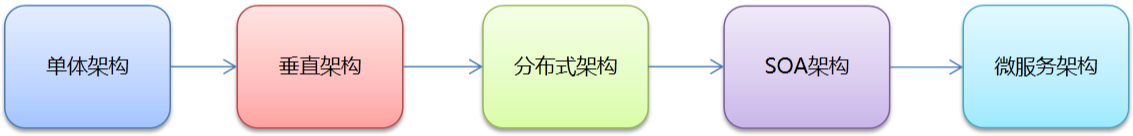
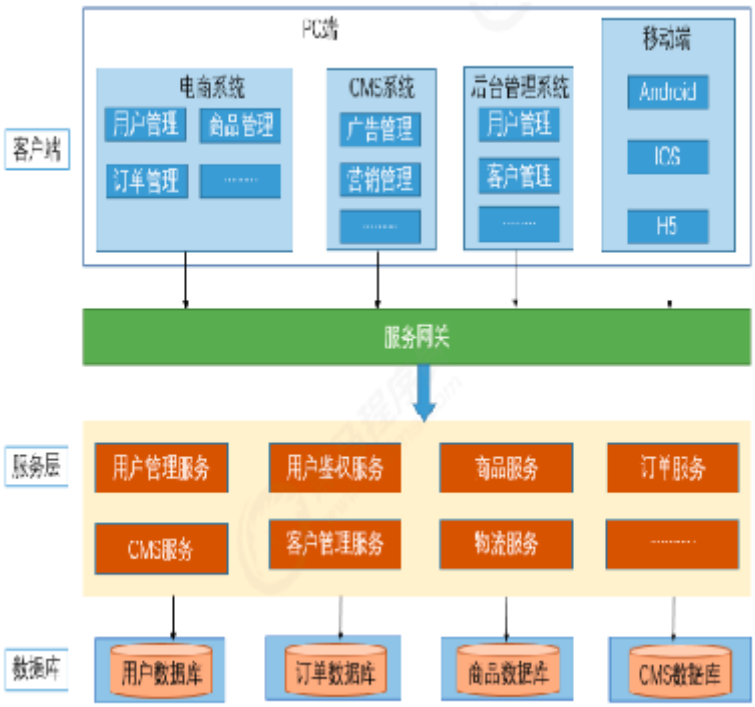
初识 Spring Cloud

微服务架构



"微服务"一词源于 Martin Fowler的名为 Microservices 的博文,可以在他的官方博客上找到http://martinfowler.com/articles/microservices.html
微服务是系统架构上的一种设计风格,它的主旨是将一个原本独立的系统拆分成多个小型服务,这些小型服务都在各自独立的进程中运行,服务之间一般通过 HTTP 的 RESTfuL API 进行通信协作。
被拆分成的每一个小型服务都围绕着系统中的某一项或些耦合度较高的业务功能进行构建,并且每个服务都维护着白身的数据存储、业务开发自动化测试案例以及独立部署机制。
由于有了轻量级的通信协作基础,所以这些微服务可以使用不同的语言来编写。



走进 Spring Cloud

Spring Cloud 是一系列框架的有序集合。
Spring Cloud 并没有重复制造轮子，它只是将目前各家公司开发的比较成熟、经得起实际考验的服务框架组合起来。
通过 Spring Boot 风格进行再封装屏蔽掉了复杂的配置和实现原理，最终给开发者留出了一套简单易懂、易部署和易维护的分布式系统开发工具包。
它利用Spring Boot的开发便利性巧妙地简化了分布式系统基础设施的开发，如服务发现注册、配置中心、消息总线、负载均衡、 断路器、数据监控等，都可以用Spring Boot的开发风格做到一键启动和部

署。
Spring Cloud项目官方网址：https://spring.io/projects/spring-cloud

Spring Cloud 版本命名方式采用了伦敦地铁站的名称，同时根据字母表的顺序来对应版本时间顺序，比如：最早的Release版本：Angel，第二个Release版本：Brixton，然后是Camden、Dalston、Edgware，Finchley，Greenwich，Hoxton。
目前最新的是Hoxton版本。

Release Train	Boot Version
Hoxton	2.2.x
Greenwich	2.1.x
Finchley	2.0.x
Edgware	1.5.x
Dalston	1.5.x

初识 Spring Cloud

Spring Cloud 与 Dubbo 对比

Spring Cloud 与 Dubbo 都是实现微服务有效的工具。
Dubbo 只是实现了服务治理，而 Spring Cloud 子项目分别覆盖了微服务架构下的众多部件。
Dubbo 使用 RPC 通讯协议，Spring Cloud 使用 RESTful 完成通信，Dubbo 效率略高于 Spring Cloud。

	Dubbo	Spring Cloud
服务注册中心	Zookeeper	Spring Cloud Netflix Eureka
服务调用方式	RPC	REST API
服务监控	Dubbo-monitor	Spring Boot Admin
断路器	不完善	Spring Cloud Netflix Hystrix
服务网关	无	Spring Cloud **Gateway**
分布式配置	无	Spring Cloud Config
服务跟踪	无	Spring Cloud Sleuth
消息总线	无	Spring Cloud Bus
数据流	无	Spring Cloud Stream
批量任务	无	Spring Cloud Task

小结

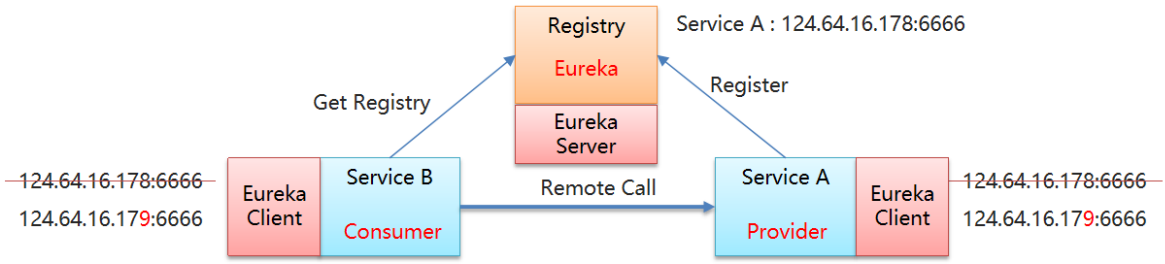
微服务就是将项目的各个模块拆分为可独立运行、部署、测试的架构设计风格。
Spring 公司将其他公司中微服务架构常用的组件整合起来，并使用 SpringBoot 简化其开发、配置。称为 Spring Cloud
Spring Cloud 与 Dubbo都是实现微服务有效的工具。Dubbo 性能更好，而 Spring Cloud 功能更全面。

Spring Cloud 服务治理

- Eureka
- Consul
- Nacos

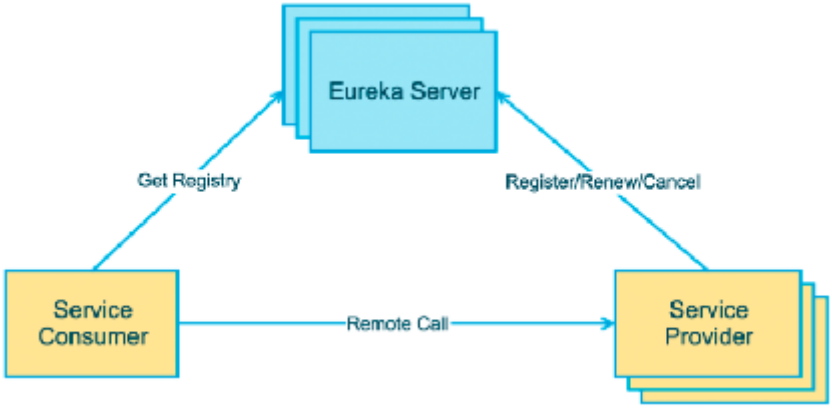
Eureka

Eureka 是 Netflix 公司开源的一个服务注册与发现的组件 。
Eureka 和其他 Netflix 公司的服务组件（例如负载均衡、熔断器、网关等） 一起，被 Spring Cloud 社区整合为
Spring-Cloud-Netflix 模块。
Eureka 包含两个组件：Eureka Server (注册中心) 和 Eureka Client (服务提供者、服务消费者)。



Eureka - 搭建服务

1. 搭建 Provider 和 Consumer 服务。
2. 使用 RestTemplate 完成远程调用。
    - Spring提供的一种简单便捷的模板类，用于在 java 代码里访问 restful 服务。
    - 其功能与 HttpClient 类似，但是 RestTemplate 实现更优雅，使用更方便。
3. 搭建 Eureka Server 服务。
    - 创建 eureka-server 模块
    - 引入 SpringCloud 和 euraka-server 相关依赖
    - 完成 Eureka Server 相关配置
    - 启动该模块
4. 改造 Provider 和 Consumer 称为 Eureka Client。
    - 引 eureka-client 相关依赖
    - 完成 eureka client 相关配置
    - 启动 测试
5. Consumer 服务 通过从 Eureka Server 中抓取 Provider 地址 完成 远程调用



Eureka – 相关配置及特性

eureka 一共有4部分 配置

1. server : eureka 的服务端配置
2. client : eureka 的客户端配置
3. instance : eureka 的实例配置
4. dashboard : eureka 的web控制台配置

Eureka – 相关配置及特性 - instance

```
eureka:•
    instance:•
        hostname: localhost # 主机名•
        prefer-ip-address: # 是否将自己的ip注册到eureka中，默认false 注册 主机名•
        ip-address: # 设置当前实例ip•
        instance-id:  # 修改instance-id显示•
        lease-renewal-interval-in-seconds: 30 # 每一次eureka client 向eureka
server发送心跳的时间间隔•
        lease-expiration-duration-in-seconds: 90 # 如果90秒内eureka server没有收到
eureka client的心跳包，则剔除该服务
```

Eureka – 相关配置及特性 - server

```
eureka:•
    server:
        #是否开启自我保护机制，默认true•
        enable-self-preservation:
        #清理间隔（单位毫秒，默认是60*1000）•
        eviction-interval-timer-in-ms:
```

image-20210822195212906

```
eureka:
    instance:
        lease-renewal-interval-in-seconds: 30 # 每一次eureka client 向 eureka
server发送心跳的时间间隔•
        lease-expiration-duration-in-seconds: 90 # 如果90秒内eureka server没有收到
eureka client的心跳包，则剔除该服务
```

EMERGENCY! EUREKA MAY BE INCORRECTLY CLAIMING INSTANCES ARE UP WHEN THEY'RE NOT. RENEWALS ARE LESSER THAN THRESHOLD AND HENCE THE INSTANCES ARE NOT BEING EXPIRED JUST TO BE SAFE.

DS Replicas

Eureka – 相关配置及特性 - client

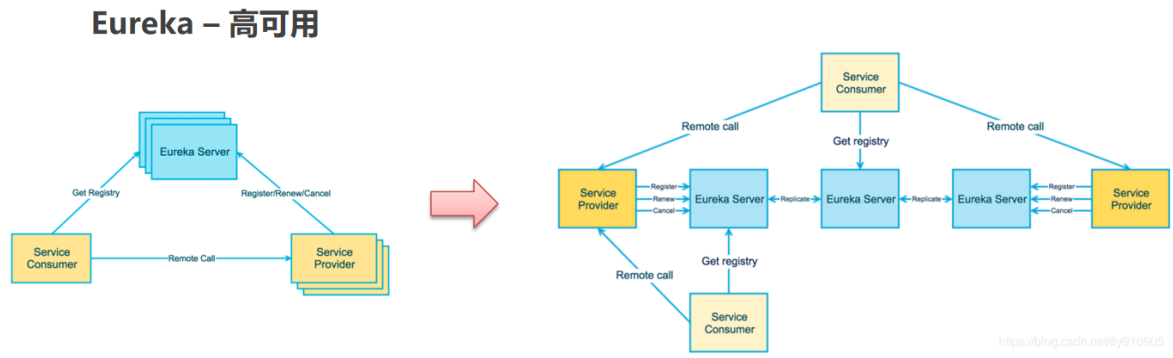
```
eureka:
    client:
        service-url:
            # eureka服务端地址，将来客户端使用该地址和eureka进行通信
            defaultZone:
        register-with-eureka: # 是否将自己的路径 注册到eureka上。
        fetch-registry: # 是否需要从eureka中抓取数据。
```

Eureka – 相关配置及特性 - dashboard

####

```
eureka:
    dashboard:
        enabled: true # 是否启用eureka web控制台
        path: / # 设置eureka web控制台默认访问路径
```

Eureka – 高可用



1. 准备两个Eureka Server
2. 分别进行配置，相互注册
3. Eureka Client 分别注册到这两个 Eureka Server中

Euraka配置详解

Eureka包含四个部分的配置

1. instance：当前Eureka Instance实例信息配置
2. client：Eureka Client客户端特性配置
3. server：Eureka Server注册中心特性配置
4. dashboard：Eureka Server注册中心仪表盘配置

Eureka Instance实例信息配置

Eureka Instance的配置信息全部保存在org.springframework.cloud.netflix.eureka.EurekaInstanceConfigBean配置类里，实际上它是com.netflix.appinfo.EurekaInstanceConfig的实现类，替代了netflix的com.netflix.appinfo.CloudInstanceConfig的默认实现。

Eureka Instance的配置信息全部以eureka.instance.xxx的格式配置。

配置列表

- appname = unknown

应用名，首先获取spring.application.name的值，如果取值为空，则取默认unknown。

- appGroupName = null

应用组名

- instanceEnabledOnit = false

实例注册到Eureka上是，是否立刻开启通讯。有时候应用在准备好服务之前需要一些预处理。

- nonSecurePort = 80

非安全的端口

- securePort = 443

安全端口

- nonSecurePortEnabled = true

是否开启非安全端口通讯

- securePortEnabled = false

是否开启安全端口通讯

- leaseRenewalIntervalInSeconds = 30

实例续约间隔时间

- leaseExpirationDurationInSeconds = 90

实例超时时间，表示最大leaseExpirationDurationInSeconds秒后没有续约，Server就认为他不可用了，随之就会将其剔除。

- virtualHostName = unknown

虚拟主机名，首先获取spring.application.name的值，如果取值为空，则取默认unknown。

- instanceId

注册到eureka上的唯一实例ID，不能与相同appname的其他实例重复。

- secureVirtualHostName = unknown

安全虚拟主机名，首先获取spring.application.name的值，如果取值为空，则取默认unknown。

- metadataMap = new HashMap();

实例元数据，可以供其他实例使用。比如spring-boot-admin在监控时，获取实例的上下文和端口。

- dataCenterInfo = new MyDataCenterInfo(DataCenterInfo.Name.MyOwn);

实例部署的数据中心。如AWS、MyOwn。

- ipAddress=null

实例的IP地址

- statusPageUrlPath = "/actuator/info"

实例状态页相对url

- statusPageUrl = null

实例状态页绝对URL

- homePageUrlPath = "/"

实例主页相对URL

- homePageUrl = null

实例主页绝对URL

- healthCheckUrlUrlPath = "/actuator/health"

实例健康检查相对URL

- healthCheckUrl = null

实例健康检查绝对URL

- secureHealthCheckUrl = null

实例安全的健康检查绝对URL

- namespace = "eureka"

配置属性的命名空间（Spring Cloud中被忽略）

- hostname = null

主机名,不配置的时候讲根据操作系统的主机名来获取

- preferIpAddress = false

是否优先使用IP地址作为主机名的标识

Eureka Client客户端特性配置

Eureka Client客户端特性配置是对作为Eureka客户端的特性配置，包括Eureka注册中心，本身也是一个Eureka Client。

Eureka Client特性配置全部在org.springframework.cloud.netflix.eureka.EurekaClientConfigBean中，实际上它是com.netflix.discovery.EurekaClientConfig的实现类，替代了netxflix的默认实现。

Eureka Client客户端特性配置全部以eureka.client.xxx的格式配置。

配置列表

- enabled=true

是否启用Eureka client。

- registryFetchIntervalSeconds=30

定时从Eureka Server拉取服务注册信息的间隔时间

- instanceInfoReplicationIntervalSeconds=30

定时将实例信息（如果变化了）复制到Eureka Server的间隔时间。（InstanceInfoReplicator线程）

- initialInstanceInfoReplicationIntervalSeconds=40

首次将实例信息复制到Eureka Server的延迟时间。（InstanceInfoReplicator线程）

- eurekaServiceUrlPollIntervalSeconds=300

拉取Eureka Server地址的间隔时间（Eureka Server有可能增减）

- proxyPort=null

Eureka Server的代理端口

- proxyHost=null

Eureka Server的代理主机名

- proxyUserName=null

Eureka Server的代理用户名

- proxyPassword=null

Eureka Server的代理密码

- eurekaServerReadTimeoutSeconds=8

从Eureka Server读取信息的超时时间

- eurekaServerConnectTimeoutSeconds=5

连接Eureka Server的超时时间

- backupRegistryImpl=null

Eureka Client第一次启动时获取服务注册信息的调用的回溯实现。Eureka Client启动时首次会检查有没有BackupRegistry的实现类，如果有实现类，则优先从这个实现类里获取服务注册信息。

- eurekaServerTotalConnections=200

Eureka client连接Eureka Server的链接总数

- eurekaServerTotalConnectionsPerHost=50

Eureka client连接单台Eureka Server的链接总数

- eurekaServerURLContext=null

当Eureka server的列表在DNS中时，Eureka Server的上下文路径。如http://xxxx/eureka。

- eurekaServerPort=null

当Eureka server的列表在DNS中时，Eureka Server的端口。

- eurekaServerDNSName=null

当Eureka server的列表在DNS中时，且要通过DNSName获取Eureka Server列表时，DNS名字。

- region="us-east-1"

实例所属区域。

- eurekaConnectionIdleTimeoutSeconds = 30

Eureka Client和Eureka Server之间的Http连接的空闲超时时间。

- heartbeatExecutorThreadPoolSize=2

心跳（续约）执行器线程池大小。

- heartbeatExecutorExponentialBackOffBound=10

心跳执行器在续约过程中超时后的再次执行续约的最大延迟倍数。默认最大延迟时间=10 * eureka.instance.leaseRenewalIntervalInSeconds

- cacheRefreshExecutorThreadPoolSize=2

cacheRefreshExecutord的线程池大小（获取注册信息）

- cacheRefreshExecutorExponentialBackOffBound=10

cacheRefreshExecutord的再次执行的最大延迟倍数。默认最大延迟时间=10 *eureka.client.registryFetchIntervalSeconds

- serviceUrl= new HashMap();serviceUrl.put(DEFAULT_ZONE, DEFAULT_URL);

Eureka Server的分区地址。默认添加了一个defualtZone。也就是最常用的配置eureka.client.service-url.defaultZone=xxx

- registerWithEureka=true

是否注册到Eureka Server。

- preferSameZoneEureka=true

是否使用相同Zone下的Eureka server。

- logDeltaDiff=false

是否记录Eureka Server和Eureka Client之间注册信息的差异

- disableDelta=false

是否开启增量同步注册信息。

- fetchRemoteRegionsRegistry=null

获取注册服务的远程地区，以逗号隔开。

- availabilityZones=new HashMap()

可用分区列表。用逗号隔开。

- filterOnlyUpInstances = true

是否只拉取UP状态的实例。

- fetchRegistry=true

是否拉取注册信息。

- shouldUnregisterOnShutdown = true

是否在停止服务的时候向Eureka Server发起Cancel指令。

- shouldEnforceRegistrationAtInit = false

是否在初始化过程中注册服务。

Eureka Server注册中心端配置

Eureka Server注册中心端的配置是对注册中心的特性配置。Eureka Server的配置全部在org.springframework.cloud.netflix.eureka.server.EurekaServerConfigBean里，实际上它是com.netflix.eureka.EurekaServerConfig的实现类，替代了netflix的默认实现。

Eureka Server的配置全部以eureka.server.xxx的格式进行配置。

配置列表

- enableSelfPreservation=true

是否开启自我保护

- renewalPercentThreshold = 0.85

自我保护续约百分比阀值因子。如果实际续约数小于续约数阀值，则开启自我保护

- renewalThresholdUpdateIntervalMs = 15 * 60 * 1000

续约数阀值更新频率。

- peerEurekaNodesUpdateIntervalMs = 10 * 60 * 1000

Eureka Server节点更新频率。

- enableReplicatedRequestCompression = false

是否启用复制请求压缩。

- waitTimeInMsWhenSyncEmpty=5 * 60 * 1000

当从其他节点同步实例信息为空时等待的时间。

- peerNodeConnectTimeoutMs=200

节点间连接的超时时间。

- peerNodeReadTimeoutMs=200

节点间读取信息的超时时间。

- peerNodeTotalConnections=1000

节点间连接总数。

- peerNodeTotalConnectionsPerHost = 500;

单个节点间连接总数。

- peerNodeConnectionIdleTimeoutSeconds = 30;

节点间连接空闲超时时间。

- retentionTimeInMSInDeltaQueue = 3 * MINUTES;

增量队列的缓存时间。

- deltaRetentionTimerIntervalInMs = 30 * 1000;

清理增量队列中过期的频率。

- evictionIntervalTimerInMs = 60 * 1000;

剔除任务频率。

- responseCacheAutoExpirationInSeconds = 180;

注册列表缓存超时时间（当注册列表没有变化时）

- responseCacheUpdateIntervalMs = 30 * 1000;

注册列表缓存更新频率。

- useReadOnlyResponseCache = true;

是否开启注册列表的二级缓存。

- disableDelta=false。

是否为client提供增量信息。

- maxThreadsForStatusReplication = 1;

状态同步的最大线程数。

- maxElementsInStatusReplicationPool = 10000;

状态同步队列的最大容量。

- syncWhenTimestampDiffers = true;

当时间差异时是否同步。

- registrySyncRetries = 0;

注册信息同步重试次数。

- registrySyncRetryWaitMs = 30 * 1000;

注册信息同步重试期间的时间间隔。

- maxElementsInPeerReplicationPool = 10000;

节点间同步事件的最大容量。

- minThreadsForPeerReplication = 5;

节点间同步的最小线程数。

- maxThreadsForPeerReplication = 20;

节点间同步的最大线程数。

- maxTimeForReplication = 30000;

节点间同步的最大时间，单位为毫秒。

- disableDeltaForRemoteRegions = false;

是否启用远程区域增量。

- remoteRegionConnectTimeoutMs = 1000;

远程区域连接超时时间。

- remoteRegionReadTimeoutMs = 1000;

远程区域读取超时时间。

- remoteRegionTotalConnections = 1000;

远程区域最大连接数

- remoteRegionTotalConnectionsPerHost = 500;

远程区域单机连接数

- remoteRegionConnectionIdleTimeoutSeconds = 30;

远程区域连接空闲超时时间。

- remoteRegionRegistryFetchInterval = 30;

远程区域注册信息拉取频率。

- remoteRegionFetchThreadPoolSize = 20;

远程区域注册信息线程数。

Eureka Server注册中心仪表盘配置

注册中心仪表盘的配置主要是控制注册中心的可视化展示。以eureka.dashboard.xxx的格式配置。

- path="/"

仪表盘访问路径

- enabled=true

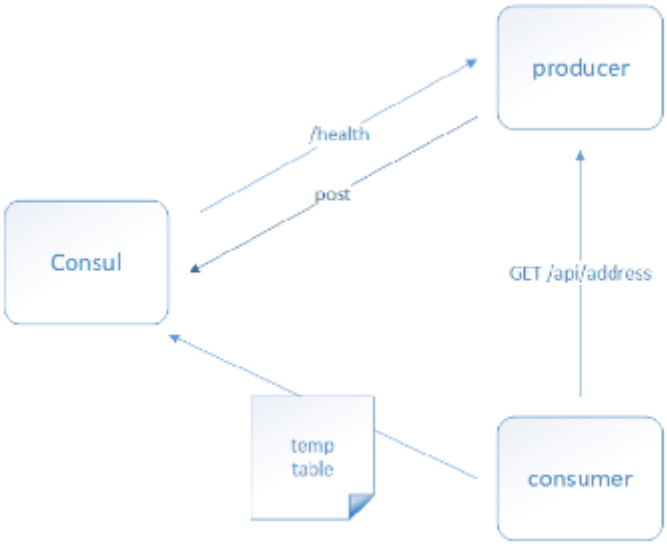
是否启用仪表盘

Consul

- Consul 是由 HashiCorp 基于 Go 语言开发的，支持多数据中心，分布式高可用的服务发布和注册服务软件。
- 用于实现分布式系统的服务发现与配置。
- 使用起来也较 为简单。具有天然可移植性(支持Linux、windows和Mac OS X)；安装包仅包含一个可执行文件，方便部署 。
- 官网地址： https://www.consul.io

Consul – 快速入门

1. 搭建 Provider 和 Consumer 服务。
2. 使用 RestTemplate 完成远程调用。
3. 将Provider服务注册到Consul中。
4. Consumer 服务 通过从 Consul 中抓取 Provider 地址 完成 远程调用



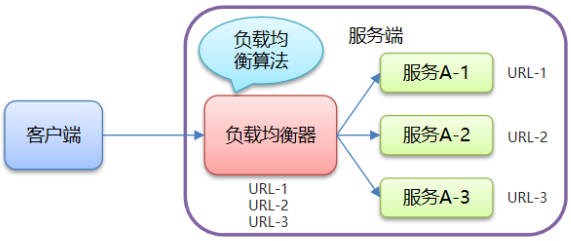
Nacos

- Nacos（Dynamic Naming and Configuration Service） 是阿里巴巴2018年7月开源的项目。
- 它专注于服务发现和配置管理领域 致力于帮助您发现、配置和管理微服务。Nacos 支持几乎所有主流类型的“服务”的发现、配置和管理。
- 一句话概括就是Nacos = Spring Cloud注册中心 + Spring Cloud配置中心。
  官网：https://nacos.io/
  下载地址： https://github.com/alibaba/nacos/releases

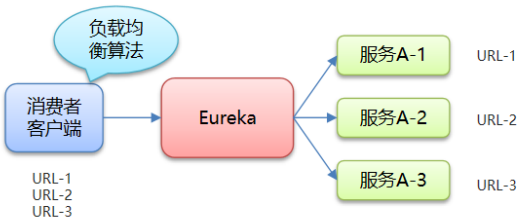
Ribbon 客户端负载均衡

Ribbon 概述

- Ribbon是 Netflix 提供的一个基于HTTP和TCP的客户端负载均衡工具。
- Ribbon主要有两个功能：
  1. 简化远程调用
  2. 负载均衡



- 服务端负载均衡
  - 负载均衡算法在服务端
  - 由负载均衡器维护服务地址列表
- 客户端负载均衡
  - 负载均衡算法在客户端
  - 客户端维护服务地址列表



Ribbon 远程调用

Ribbon 可以与 简化 RestTemplate 的远程调用

Ribbon 负载均衡

Ribbon 负责均衡策略：

- 随机： RandomRule
- 轮询： RoundRobinRule
- 最小并发：BestAvailableRule
- 过滤：AvailabilityFilteringRule
- 响应时间：WeightedResponseTimeRule
- 轮询重试：RetryRule
- 性能可用性：ZoneAvoidanceRule

设置负载均衡策略

1. 编码
2. 配置

```
user-service: # 生产者服务名称
    ribbon:
        NFloadBalancerRuleClassName: XxxRule # 负载均衡策略类
```