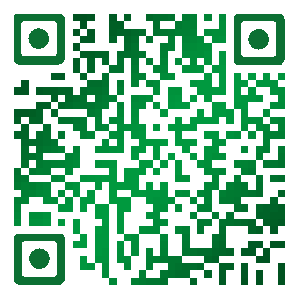
**| 前言**

Nepxion Discovery是由开发者任浩军创建的开源项目，这是一款以Nacos为核心技术，是对Spring Cloud Discovery服务注册发现、Ribbon负载均衡、Feign和RestTemplate调用等增强的中间件。

其功能包括：

* 灰度发布（包括切换发布和平滑发布）、服务隔离、服务路由、服务权重、黑/白名单的IP地址过滤、限制注册、限制发现等；
* 支持Eureka、Consul、Zookeeper和阿里巴巴的Nacos为服务注册发现中间件；
* 支持阿里巴巴的Nacos、携程的Apollo和Redis为远程配置中心；
* 支持Spring Cloud Api Gateway（Finchley版）、Zuul网关和微服务的灰度发布；
* 支持多数据源的数据库灰度发布等客户特色化灰度发布；
* 支持用户自定义和编程灰度路由策略（包括RPC和REST两种调用方式）；
* 兼容Spring Cloud Edgware版和Finchley版；
* 现有的Spring Cloud微服务很方便引入该中间件，代码零侵入。



扫码访问Nepxion Discovery

**| 如何基于Nacos实现Spring Cloud灰度发布和路由**

主要分为如下三部分：

* 整合Nacos服务注册发现机制，实现Spring Cloud的灰度发布和路由；
* 利用Nacos配置中心，实现Spring Cloud的灰度发布和路由规则的推送、订阅；
* 利用Nacos控制台，实现Spring Cloud的灰度发布和路由规则的配置；

无论是原生的Nacos Client，还是Nacos Spring、Nacos SpringBoot，或者Nacos Spring Cloud都具有非常好的用户易用性和扩展性，尤其是Nacos Spring系列，紧紧遵循Spring生态的规范，所以大家可以看到整合起来代码量相对较少，也比较简单。本文考虑到篇幅，只介绍涉及到整合Nacos的部分，涉及到具体灰度发布和路由的逻辑则不在讲述范围内，请自行访问Github相关代码和文档进行研究。

**| 整合Nacos服务注册发现机制，实现spring cloud的灰度发布和路由**

本模块是基于spring-cloud-alibaba-nacos-discovery（见 https://github.com/spring-cloud-incubator/spring-cloud-alibaba）标准化的服务注册发现机制而实现的，我们可以完全可以象扩展Eureka、Consul或者Zookeeper Discovery组件一样，去扩展Nacos组件做灰度发布和路由，下文主要讲述几个扩展步骤，对所有的服务注册发现组件都是大体一致。

* **装饰类**

NacosServiceRegistryDecorator，服务注册层面的装饰类，继承和装饰NacosServiceRegistry，实现通过RegisterListenerExecutor注册监听执行器对它的核心方法进行拦截，从而实现在注册层面的“黑/白名单的IP地址注册的过滤规则”、“最大注册数的限制的过滤规则”等功能。

NacosServerListDecorator，服务发现层面的装饰类，继承NacosServerList，实现通过LoadBalanceListenerExecutor负载均衡监听执行器对它的核心方法进行拦截过滤，从而实现在负载均衡层面的“版本访问的灰度路由规则”、“版本权重的灰度路由规则”、“区域权重的灰度路由规则”等功能。

* **适配类**

NacosAdapter，适配器类。由于在不同的服务注册发现组件（Eureka、Consul、Zookeeper、Nacos）中，获得Metadata的逻辑是实现在Server的子类上，所以我们要做一层适配。Metadata的数据在灰度发布和路由中起着至关重要的作用，比如灰度发布中涉及到的版本（Version）、组（Group）和区域（Region）都是通过Metadata方式提供。

* **初始化类**

NacosApplicationContextInitializer，作为初始化类，是在Spring容器初始化的时候执行，可以对Spring容器中的Bean进行拦截和替换。对NacosServiceRegistry对象进行拦截，由NacosServiceRegistryDecorator去代理；对NacosDiscoveryProperties对象进行拦截，并把本地相关的Metadata数据写入，并注册到Nacos服务器上，这样让第三方监控系统拿到相关的灰度发布数据，做监控分析。

* **配置类**

NacosLoadBalanceConfiguration，负载均衡的配置类。由于NacosRibbonClientConfiguration的ribbonServerList方法用@ConditionalOnMissingBean注解，这样我们可以用自定义的扩展替换掉它。在NacosLoadBalanceConfiguration里，我们用NacosServerListDecorator装饰类替换NacosServerList作为ribbonServerList方法返回值，放入灰度发布的负载均衡拦截执行器LoadBalanceListenerExecutor。

在NacosAutoConfiguration里，我们需要指定RibbonClients注解的配置类列表，包含PluginLoadBalanceConfiguration和NacosLoadBalanceConfiguration。PluginLoadBalanceConfiguration封装了通用灰度发布逻辑。

最后，基于Nacos官方提供的配置文件进行配置，整个基于spring-cloud-alibaba-nacos-discovery的灰度发布和路由功能就完成了，代码简单而扩展性强。

**| 利用Nacos配置中心**

**实现Spring Cloud的灰度发布和路由规则的推送、订阅**

本模块并没有直接用spring-cloud-alibaba-nacos-config（见 https://github.com/spring-cloud-incubator/spring-cloud-alibaba），因为灰度规则各项操作相对较复杂，所以采用了原生的Nacos Client Api（见 https://github.com/alibaba/nacos）来实现。

* **Common层实现**

NacosOperation，封装了几乎所有对Nacos配置中心的操作逻辑，包括：

a. 根据微服务所在的组和应用名，获取配置

b. 根据微服务所在的组和应用名，删除配置

c. 根据微服务所在的组和应用名，发布配置

d. 根据微服务所在的组和应用名，订阅配置

e. 根据微服务所在的组和应用名，反订阅配置

NacosAutoConfiguration，通过AutoConfiguration初始化NacosConfigService和NacosOperation

a. 通过@ConditionalOnMissingBean的方式，允许用户通过自己实现的ConfigService进行注入，来代替内置方式；

b. 如果通过内置方式，那么用户只需要在配置文件里，填入相关配置，即可完成初始化。如下配置除了url必填之外，其它也可以由用户自行去定义。

* **微服务端实现**

NacosConfigAdapter，继承实现ConfigAdapter（处理灰度发布配置的适配器）

a. getConfig，用于微服务端在启动的时候，向Nacos服务器请求灰度配；

b. subscribeConfig，用于微服务端在启动的时候，完成初始化对灰度配置的监听行为；

c. close，用于当微服务端断开和Nacos服务器连接或者Spring Bean销毁的时候，执行反订阅和线程池销毁；

* **控制台实现**

控制平台的作用是当用户自行研发第三方管理界面的时候，可以通过微服务的方式发布Nacos服务器配置操作和汇聚的接口（我们统称它为控制平台）。对于本系统来说，目前它的作用是为Java Desktop图形化界面提供接口，您也可以使用它自行研发符合您口味的灰度发布界面 NacosConfigAdapter，继承实现ConfigAdapter（控制平台操作配置的适配器），该类和“服务端”的类同名，但并不是同一个，主要有三个方法：

a. updateConfig，用于用户界面更新配置；

b. clearConfig，用于用户界面清楚配置；

c. getConfig，用于用户界面获取配置；

**| 利用Nacos控制台，**

**实现Spring Cloud的灰度发布和路由规则的配置**

敬请期待Nacos 0.3.0版本，推出Nacos控制台。