



Discovery 【探索】

蓝绿灰度发布
最佳实践



<http://www.nepxion.com>

<https://github.com/Nepxion>

©2017-2050 Nepxion Studio. All Rights Reserved.

Let us start Discovery



Discovery 【探索】



框架集成

Pom引入



Zuul引入

```
<!-- 1.注册中心插件-->
<dependency>
<groupId>com.nepxion</groupId>
<artifactId>discovery-plugin-register-center-starter-nacos</artifactId>
</dependency>

<!-- 2.配置中心插件-->
<dependency>
<groupId>com.nepxion</groupId>
<artifactId>discovery-plugin-config-center-starter-nacos</artifactId>
</dependency>

<!-- 3.管理中心插件-->
<dependency>
<groupId>com.nepxion</groupId>
<artifactId>discovery-plugin-admin-center-starter</artifactId>
</dependency>

<!-- 4.网关策略编排插件-->
<dependency>
<groupId>com.nepxion</groupId>
<artifactId>discovery-plugin-strategy-starter-zuul</artifactId>
</dependency>
```



Spring Cloud
Gateway引入

```
<!-- 1.注册中心插件-->
<dependency>
<groupId>com.nepxion</groupId>
<artifactId>discovery-plugin-register-center-starter-nacos</artifactId>
</dependency>

<!-- 2.配置中心插件-->
<dependency>
<groupId>com.nepxion</groupId>
<artifactId>discovery-plugin-config-center-starter-nacos</artifactId>
</dependency>

<!-- 3.管理中心插件-->
<dependency>
<groupId>com.nepxion</groupId>
<artifactId>discovery-plugin-admin-center-starter</artifactId>
</dependency>

<!-- 4.网关策略编排插件-->
<dependency>
<groupId>com.nepxion</groupId>
<artifactId>discovery-plugin-strategy-starter-gateway</artifactId>
</dependency>
```



Service引入

```
<!-- 1.注册中心插件-->
<dependency>
<groupId>com.nepxion</groupId>
<artifactId>discovery-plugin-register-center-starter-nacos</artifactId>
</dependency>

<!-- 2.配置中心插件-->
<dependency>
<groupId>com.nepxion</groupId>
<artifactId>discovery-plugin-config-center-starter-nacos</artifactId>
</dependency>

<!-- 3.管理中心插件-->
<dependency>
<groupId>com.nepxion</groupId>
<artifactId>discovery-plugin-admin-center-starter</artifactId>
</dependency>

<!-- 4.服务策略编排插件-->
<dependency>
<groupId>com.nepxion</groupId>
<artifactId>discovery-plugin-strategy-starter-service</artifactId>
</dependency>
```

* 以Nacos注册中心和配置中心为例



框架集成

异步场景下DiscoveryAgent引入

异步场景描述

寄存在ThreadLocal中的Header等上下文对象，在如下异步场景下，线程切换后，发生丢失情况

- WebFlux Reactor
- @Async
- Hystrix Thread Pool Isolation
- Runnable
- Callable
- Supplier
- Single Thread
- Thread Pool
- SLF4J MDC

通过引入异步跨线程DiscoveryAgent进行解决

引入

启动参数中引入Agent

```
-javaagent:/discovery-agent/discovery-agent-starter-  
${discovery.agent.version}.jar  
-Dthread.scan.packages=com.abc.xyz
```

说明

启动参数说明

`/discovery-agent`
Agent所在的目录

`-Dthread.scan.packages=com.abc.xyz`
解决该目录下用户自定义Runnable/Callable/Thread/ThreadPool的异步
当用户未定义上述四个异步类，不需要扫描目录

更多内容参考

适用于一切Java技术栈的异步场景
<https://github.com/Nepxion/DiscoveryAgent>



* Spring cloud 2020以上（含），必须引入DiscoveryAgent

* Spring cloud Hoxton以下（含），如果含有异步调用场景，必须引入DiscoveryAgent



流量染色

组染色

参数方式

- 染色方式：启动参数 `java -jar -Dmetadata.group=nepxion abc.jar`

配置方式

- 染色方式：配置文件 `spring.cloud.discovery.metadata.group=nepxion`



流量染色

版本染色

静态方式

版本不可排序模式。版本号采用固定的值的方式，每次上线中来回切换，交替使用根据服务实例全局唯一ID的时间戳前缀进行排序，把上线时间最早的服务实例的版本号作为旧的稳定版本，作为版本故障转移、版本偏好和兜底路由的基准值

如果业务服务开启版本故障转移、版本偏好功能，需要将版本号排序类型改为时间戳（time）
`spring.application.strategy.version.sort.type=time`

动态方式

版本可排序模式。版本号采用时间戳或者数字递增的方式将排序后版本号列表的第一个值作为旧的稳定版本，作为版本故障转移、版本偏好和兜底路由的基准值



流量染色

版本染色

静态方式

- 版本格式：服务实例旧版本赋值为green，新版本赋值为blue，每次上线中来回切换，交替使用
- 染色方式：启动参数 `java -jar -Dmetadata.version=blue abc.jar`

动态方式

- ① 基础架构平台使用编译插件git-commit-id-plugin以时间戳方式进行染色
 - 版本格式：例如， `20210601-567`，日期 + Git提交次数
 - 染色方式：权力下放给基础架构平台，自动染色，不需要加启动参数
- ② 基础架构平台使用编译插件git-commit-id-plugin以POM版本号方式进行染色
 - 版本格式：例如， `1.0.0`， `1.0.1`， `1.1.0`
 - 染色方式：权力下放给基础架构平台，自动染色，不需要加启动参数
- ③ DevOps运维平台以时间戳方式进行染色
 - 版本格式：例如， `20210601-0003`，时期 + 指定服务当天发布次数
 - 染色方式：权力下放给DevOps运维平台，需要加启动参数 `-Dmetadata.version=20210601-0003`
- ④ DevOps运维平台以数字递增方式进行染色
 - 版本格式：例如， `1.0.0`， `1.0.1`， `1.1.0`
 - 染色方式：权力下放给DevOps运维平台，需要加启动参数 `-Dmetadata.version=1.0.0`



流量染色

版本染色

Git编译插件染色

- 添加编译插件git-commit-id-plugin

<plugin>

<groupId>pl.project13.maven</groupId>

<artifactId>git-commit-id-plugin</artifactId>

<executions>

<execution>

<goals>

<goal>revision</goal>

</goals>

</execution>

</executions>

<configuration>

<generateGitPropertiesFile>true</generateGitPropertiesFile>

<dateFormat>yyyyMMdd</dateFormat>

</configuration>

</plugin>

- 开启Git插件产生版本号的开关

spring.application.git.generator.enabled=true

- 日期 + Git提交次数的版本号格式

spring.application.git.version.key={git.commit.time}-{git.total.commit.count}

- POM版本号格式

spring.application.git.version.key={git.build.version}



过程实施

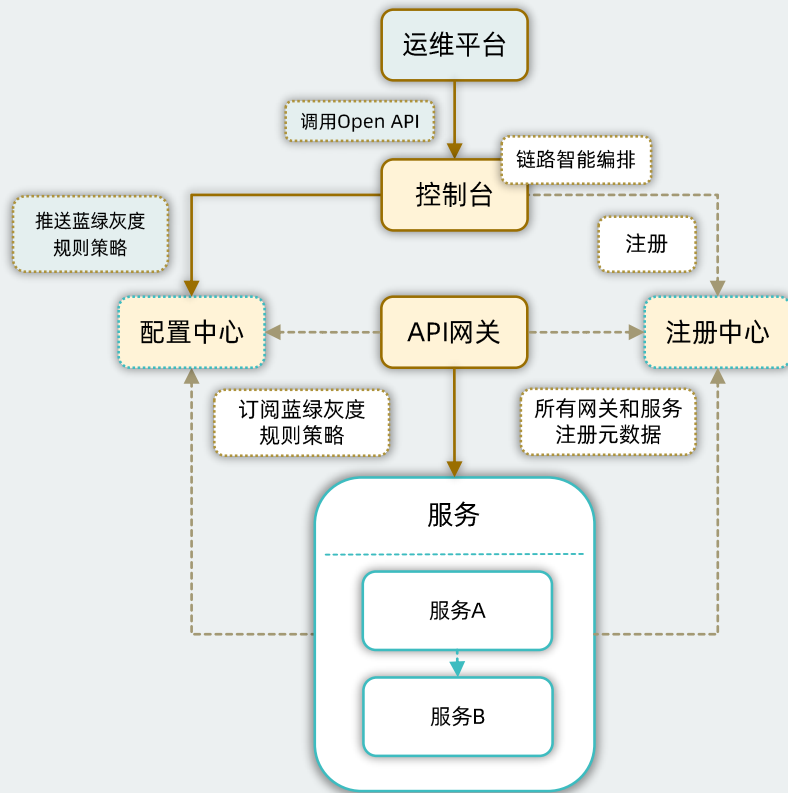
对接DevOps运维平台实施蓝绿灰度发布

架构

- ① 控制台需要连接注册中心和配置中心
- ② 控制台建议实现高可用架构，控制台前面部署API网关和运维平台对接

方案

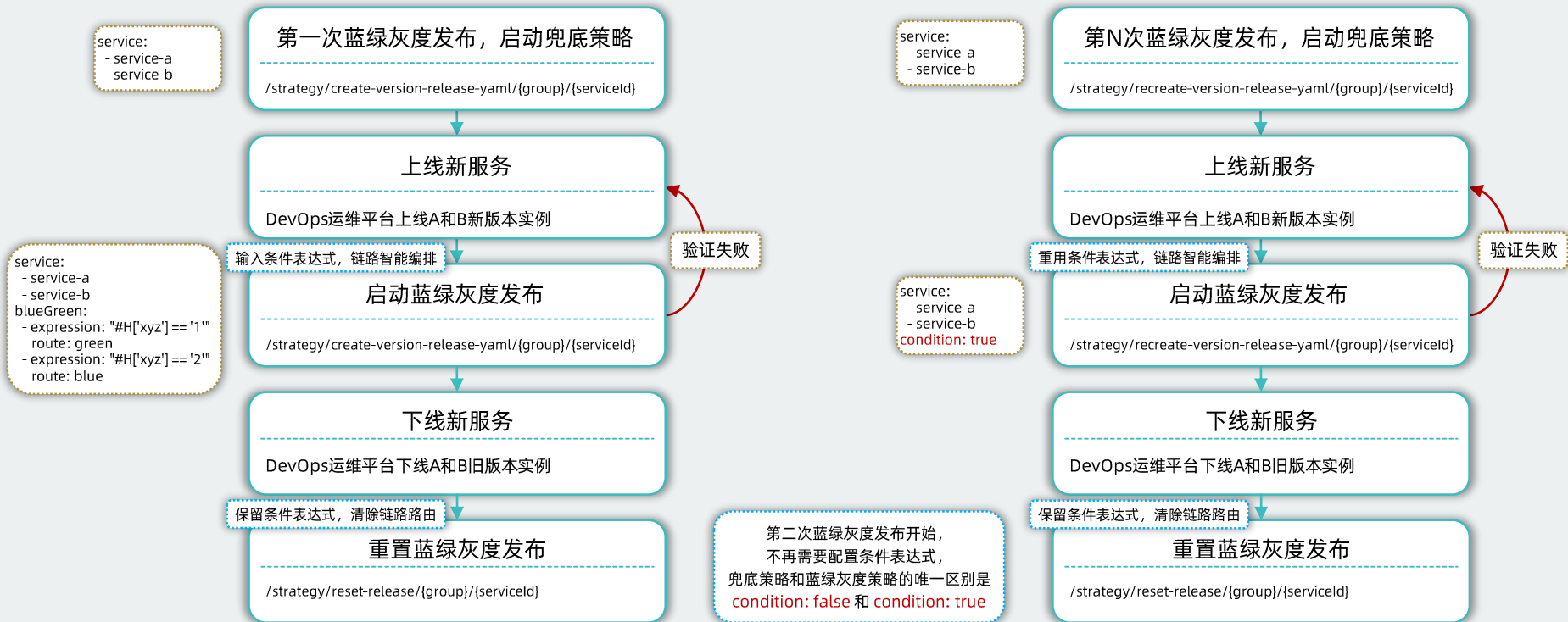
- ① 运维平台调用控制台的Open API，控制台进行链路智能编排
- ② 控制台把最终蓝绿灰度规则策略推送到配置中心





过程实施

对接DevOps运维平台实施蓝绿灰度发布链路智能编排流程



概念

- ① 路由链路在后台会智能化编排
- ② 无需关心服务版本的信息
- ③ 只需配置条件表达式

方案

- ① 向控制台发送请求
- ② 控制台根据新旧版本的判断，智能编排出两条新旧路由链路，并给它们赋予不同的条件表达式
- ③ 解析简单规则策略为最终规则策略，保存至配置中心

重点

- ① 动态版本：版本号采用时间戳或者数字递增的方式
- ② 静态版本：版本号采用固定方式。例如，base, gray, green, blue等

兜底规则策略

```
service:
- a
- b
sort: version
```

蓝绿规则策略

```
service:
- a
- b
blueGreen:
- expression: "#H['xyz'] == '1'"
  route: green
- expression: "#H['xyz'] == '2'"
  route: blue
sort: version
```

灰度规则策略

```
service:
- a
- b
gray:
- expression: "#H['xyz'] == '3'"
  weight:
    - 90
    - 10
- expression: "#H['xyz'] == '4'"
  weight:
    - 70
    - 30
- weight:
    - 100
    - 0
sort: version
```

二次蓝绿灰度规则策略

```
service:
- a
- b
condition: true
sort: version
```

规则策略解释

- ① 指定要实施的服务列表（例如，a和b）
- ② 版本号排序类型（sort），可选值为version和time，缺省为version（不需要配置sort: version）
当排序类型为version时，适用于动态版本。排序后版本号列表的第一个值作为旧的稳定版本
当排序类型为time时，适用于动态和静态版本。根据服务实例全局唯一ID的时间戳前缀进行排序，把上线时间最早的服务实例的版本号作为旧的稳定版本
- ③ 兜底规则策略无需指定条件（expression）
- ④ 蓝绿规则策略必须指定蓝绿两个条件（expression）和路由（route）项
路由由green代表绿（旧版本）路由链路，路由由blue代表蓝（新版本）路由链路
- ⑤ 灰度规则策略条件和权重项可以无数个（不同条件下的不同权重配比），允许有一项条件（expression）为空（无条件驱动下的权重）
灰度权重（weight）必须为两个整数（可以大于100）的数字，按次序为别代表稳定（旧版本）路由链路权重，灰度（新版本）路由链路权重
- ⑥ 上述三个规则策略组合在一起，为蓝绿灰度混合发布
- ⑦ 二次蓝绿灰度，表示第一次执行过蓝绿灰度发布后，会保留条件（expression）项，清除路由（route）项，以后蓝绿灰度不必填写条件（expression）项，只需用condition: true代替即可
- ⑧ 支持Yaml和Json两种格式



测试验收

全链路自动化模拟流程测试

方案

- ① 全链路智能编排 + 流量侦测
- ② 网关或服务为侦测入口
- ③ 自动判断结果准确性
- ④ 服务引入discovery-plugin-admin-center-starter依赖

过程

- ① 云上测试
 - 访问测试平台
 - 日志根据测试用例UUID输出和采集，测试并行控制，Web界面展现
- ② 本地测试
 - 获取离线包
 - 通过命令行 (.bat或者.sh) 执行

适用

- ① 测试环境
- ② 开发环境

测试结果部分示例

【模拟场景3】蓝绿策略，测试全链路侦测，Header : {xyz=1}...
侦测次数：100
侦测结果：discovery-guide-service-a@@1.0 命中次数=100
侦测结果：discovery-guide-service-a@@1.1 命中次数=0
侦测结果：discovery-guide-service-b@@1.0 命中次数=100
侦测结果：discovery-guide-service-b@@1.1 命中次数=0
测试耗时：1 秒

【模拟场景3】灰度策略，测试全链路侦测，Header : {xyz=3}...
侦测次数：500
侦测进度：第100次...
侦测进度：第200次...
侦测进度：第300次...
侦测进度：第400次...
侦测进度：第500次...
侦测结果：discovery-guide-service-a@@1.0 命中次数=448
侦测结果：discovery-guide-service-a@@1.1 命中次数=52
侦测结果：discovery-guide-service-b@@1.0 命中次数=448
侦测结果：discovery-guide-service-b@@1.1 命中次数=52
期望结果：旧版本路由权重=90%，新版本路由权重=10%
最终结果：旧版本路由权重=89.6%，新版本路由权重=10.4%
测试耗时：7 秒



测试验收

全链路自动化流量侦测测试

方案

- ① 全链路流量侦测
- ② 网关或服务为侦测入口
- ③ 人工判断结果准确性
- ④ 服务引入discovery-plugin-admin-center-starter依赖

过程

- ① 云上测试
 - 访问测试平台
 - 日志根据测试用例UUID输出和采集, Web界面展现
- ② 本地测试
 - 获取离线包
 - 通过命令行 (.bat或者.sh) 执行

适用

- ① 生产环境

测试结果部分示例

【侦测场景1】测试全链路侦测...

侦测次数: 10

侦测结果: [ID=discovery-guide-gateway][V=1.0] -> [ID=discovery-guide-service-a][V=1.0] -> [ID=discovery-guide-service-b][V=1.0]

侦测结果: [ID=discovery-guide-gateway][V=1.0] -> [ID=discovery-guide-service-a][V=1.1] -> [ID=discovery-guide-service-b][V=1.1]

侦测结果: [ID=discovery-guide-gateway][V=1.1] -> [ID=discovery-guide-service-a][V=1.0] -> [ID=discovery-guide-service-b][V=1.0]

侦测结果: [ID=discovery-guide-gateway][V=1.1] -> [ID=discovery-guide-service-a][V=1.1] -> [ID=discovery-guide-service-b][V=1.1]

侦测结果: [ID=discovery-guide-gateway][V=1.0] -> [ID=discovery-guide-service-a][V=1.0] -> [ID=discovery-guide-service-b][V=1.0]

侦测结果: [ID=discovery-guide-gateway][V=1.0] -> [ID=discovery-guide-service-a][V=1.1] -> [ID=discovery-guide-service-b][V=1.1]

侦测结果: [ID=discovery-guide-gateway][V=1.1] -> [ID=discovery-guide-service-a][V=1.0] -> [ID=discovery-guide-service-b][V=1.0]

侦测结果: [ID=discovery-guide-gateway][V=1.1] -> [ID=discovery-guide-service-a][V=1.1] -> [ID=discovery-guide-service-b][V=1.1]

侦测结果: [ID=discovery-guide-gateway][V=1.0] -> [ID=discovery-guide-service-a][V=1.0] -> [ID=discovery-guide-service-b][V=1.0]

侦测结果: [ID=discovery-guide-gateway][V=1.0] -> [ID=discovery-guide-service-a][V=1.1] -> [ID=discovery-guide-service-b][V=1.1]

测试耗时: 0 秒

【侦测场景1】结束



文档指南

文档指南和联系方式



[如何执行全链路智能编排高级蓝绿灰度发布](#)

[如何部署对接DevOps运维平台的控制台](#)

[如何对接DevOps运维平台执行半自动化蓝绿灰度发布](#)

[如何执行全链路自动化模拟流程测试](#)

[如何执行全链路自动化流量侦测测试](#)

微信



钉钉



公众号



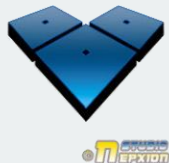
文档





文档指南

文档指南和联系方式



主页地址: <http://www.nepxion.com>

源码地址: <https://github.com/Nepxion/Discovery>

镜像地址: <https://gitee.com/Nepxion/Discovery>

指南地址: <https://github.com/Nepxion/DiscoveryGuide>

框架文档: <https://nepxion.com/discovery>

平台文档: <https://nepxion.com/discovery-platform>

微信



钉钉



公众号



文档



Thanks for Watching



Discovery 【探索】