一、功能说明

该工具包的作用: 是发布试运营版本的代码到正式环境, 开发者可以根据配置灰度策略, 使部分真实用户使用灰度代码体验新业务功能。

其中包含微服务调用的灰度持续功能, 实现了微服务全链路灰度支持。

背景

- 由于测试环境和正式环境存在细微差异, 测试环境的测试结果不能全面反映其业务的可靠性。
- 初版支持三种灰度策略混合使用: ips、tokens、weight, 混合使用时须所配策略均命中才会开启灰度请求。
- 后端灰度只需要后端开发人员参与开发, 不需要任何其他人员的参与, 且用户无感知。

二、工具使用

- 添加依赖: pom.xml

```
<dependency>
    <groupId>cn.huolala</groupId>
    <artifactId>common-api</artifactId>
    <version>2.0.1</version>
</dependency>
```

- 启用工具

> 在启动类上或其他配置类上扫描包: cn.huolala.common

```
@SpringBootApplication
@ComponentScan("cn.huolala.common")
public class XxxApp {
    public static void main(String[] args) {
        SpringApplication.run(XxxApp.class, args);
    }
}
```

- 灰度配置

```
# **: 任意层任意名称
# * : 当前层任意名称
common.api.grays[0].urlPatterns=http://localhost:9740/**
# 需发布响应版本的灰度资源: @ApiCoexist("1.0.0-GRAY")
common.api.grays[0].version=1.0.0-GRAY
# 可混合使用: ps, tokens, weight
# 没有配置的规则不需要配置相应的数据, 如以下注释的部分
common.api.grays[0].rule=weight
# common.api.grays[0].ips[0]=192.168.1
# common.api.grays[0].ips[1]=192.168.2
# common.api.grays[0].tokens[0]=476922151215454515115151 5
# common.api.grays[0].tokens[1]=476922151215454515115151 6
common.api.grays[0].weight=50
```

- 发布灰度资源
  1. 旧版资源

```
@PostMapping("gray")
@ApiOperation(value = "旧版测试")
@ApiPrint("旧版测试")
public JsonResult grayFalse(
    @RequestHeader(value = "version", required = false) String version,
    @CookieValue(value = "GRAY_COOKIE", required = false) String
grayToken
) {
    System.out.println("进入旧版..");
    System.out.println("================= version: " + version);
    System.out.println("================= grayToken: " + grayToken);
    return JsonResult.OK.newly("进入旧版..");
}
```

  2. 灰度资源: @ApiCoexist("1.0.0-GRAY")

> 使用 @ApiCoexist 注解指定灰度资源的版本号

```
@PostMapping("gray")
@ApiOperation(value = "灰度测试")
@ApiPrint("灰度测试")
@ApiCoexist("1.0.0-GRAY")
public JsonResult grayTrue(
    @RequestHeader(value = "version", required = false) String version,
    @CookieValue(value = "GRAY_COOKIE", required = false) String
grayToken
) {
    System.out.println("进入灰度..");
    System.out.println("================= version: " + version);
    System.out.println("================= grayToken: " + grayToken);
    return JsonResult.OK.newly("进入灰度..");
}
```

- 注解说明: @ApiCoexist("1.0.0-GRAY")
  - 默认值: 必须与灰度配置version一致, 否则走旧版不走灰度。

三、工具原理

逻辑

- 灰度发布依赖于接口共存工具 (请不要排除接口共存自动装载配置)
- 灰度过滤器按照开发者配置的规则对每个请求进行匹配
- 匹配命中后将请求转为灰度请求 (灰度资源不存在自动请求旧版资源)

代码

1. 通过包扫描注册自动注册灰度过滤器
2. 灰度过滤器过滤所有请求: /*
3. 请求进入即遍历所有灰度规则的配置数据
4. 与规则相匹配的请求则在请求头中追加 `version` 参数

   > 参数值与灰度配置中的版本号一致
   >
   > 具有该值视为请求灰度资源
   >
   > 灰度资源不存在则请求旧版资源

5. 第一次命中后在响应中追加灰度Cookie
6. 当Cookies中包含灰度请求并且请求的url与请求一致即直接进入灰度模式

   > 不需要再经过灰度策略, Cookie永久有效
   >
   > 取消灰度方式: 去除灰度资源使请求自动进入旧版资源

数据

- 权重数据: `visitsMap` 每次请求的url与灰度规则不匹配则清理一次
- 灰度Cookie: 每次请求的url与灰度规则不匹配则清理一次

重写

- 排除默认实现: @SpringBootApplication

```
@SpringBootApplication(exclude = {
        ApiCoexistAutoConfiguration.class, // 不需要也可以排除, 一般微服务会留下, 网关等服务会排除
        ApiGrayFilterAutoConfiguration.class,
        ApiCoexistFeignLineGrayAutoConfiguration.class //不需要灰度持续功能可以排除: 全链路灰度的支撑
})
public class XxxApp {
    public static void main(String[] args) {
        SpringApplication.run(XxxApp.class, args);
    }
}
```

- 新建过滤器继承抽象类: AbstractGrayFilter<Request, Response>

  > 微服务重写的话将以下 GlobalFilter 换成 Filter 即可; Spring Cloud Gateway 重写案例如下:

```
@Component
public class GatewayApiCoexistGrayFilter extends
AbstractGrayFilter<ServerHttpRequest, ServerHttpResponse> implements
GlobalFilter {
    ...
}
```

- 网关重写说明:

> 网关虽然也可以集成灰度, 但不建议这么做, 原因是:
>
> 在各个项目的token机制不一致时, 无法使用 TOKENS 方案配置灰度规则

四、使用示例

- 首次访问

```
{
  "code": 200,
  "msg": "进入旧版..",
  "data": null,
  "responseTime": "1606894125558"
}
```

- 再次访问

> 由于配置了50%的权重, 所以第二次将进入灰度, 并且写入固用灰度的Cookie
>
> 写入固用灰度Cookie后, 每次请求携带Cookie, 遇此Cookie则直接进入灰度

```
{
  "code": 200,
  "msg": "进入灰度..",
  "data": null,
  "responseTime": "1606894138387"
}
```

- 首次灰度后再次访问

```
{
  "code": 200,
  "msg": "进入灰度..",
  "data": null,
  "responseTime": "1606894247562"
}
```