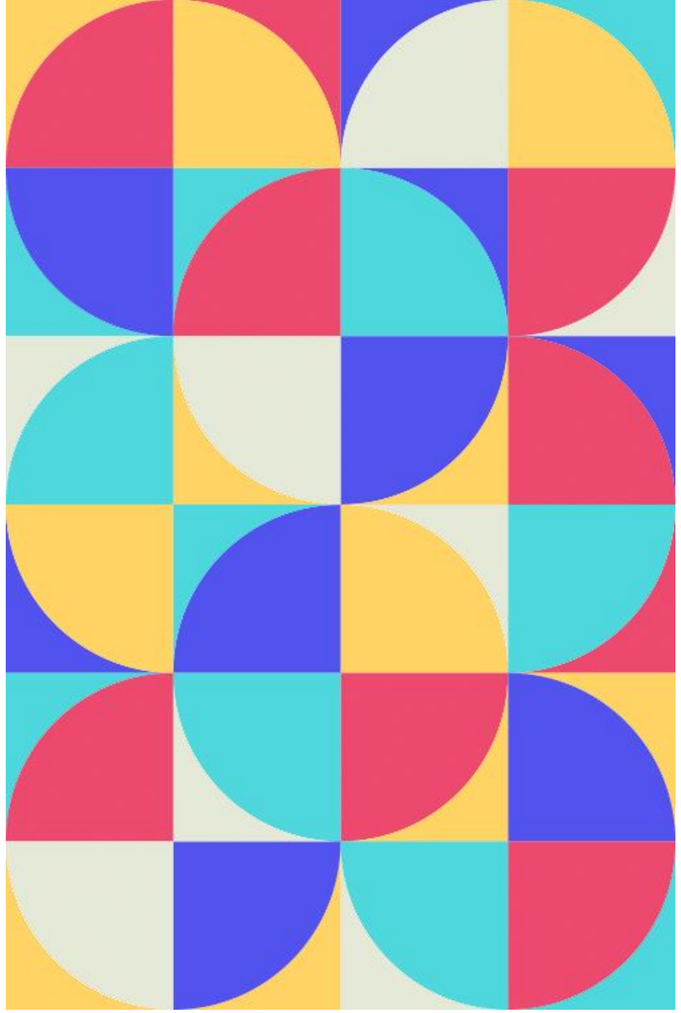
Spring Cloud

全链路灰度发布方案



大纲

CONTENTS

- 灰度概念解析
- 实现灰度的核心问题
- 灰度方案实现
- 方案缺陷

灰度概念解析

1. 灰度发布概念

灰度发布是相较于“全量发布”的改进，灰度发布会按照一定的策略上线部分新版本内容，同时保留老版本功能。让一部分用户体验新版本功能，通过一定时间的对新版本的观察以及反馈收集（比如：功能，性能、稳定性等指标），来决定最终是否逐步升级直至全量或全部回滚至老版本。

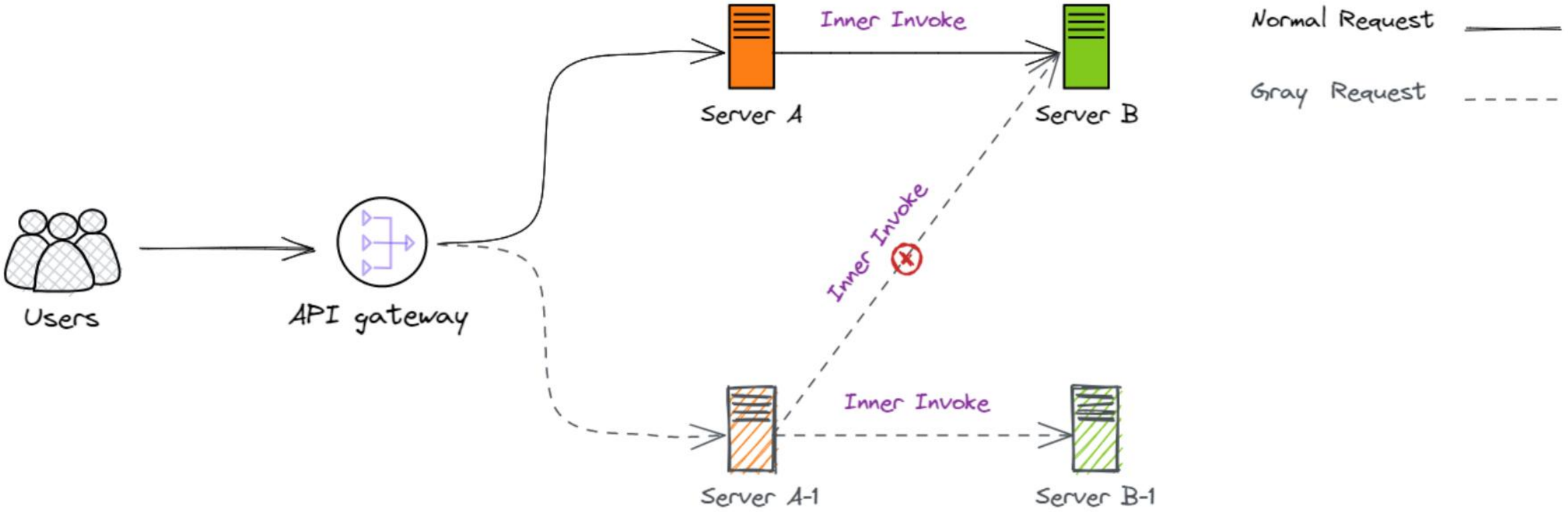
2. 灰度发布的好处

- **降低发布影响面：**就算发布过程中出现问题，也只会影响部分用户，并且可以提前发现新版本中的问题，避免影响更多用户。
- **提升用户体验：**除了提前发现bug,还能够很好的收集新版本中用户的使用反馈，从而提前优化系统，提升用户体验，也能够为后续产品的演进带来参考价值。

2. 灰度发布分类

- 金丝雀发布
- 滚动发布
- 蓝绿发布

实现灰度的核心问题


Inner Invoke
Server A
Server B
Normal Request
Gray Request
Users
API gateway
Inner Invoke
Inner Invoke
Server A-1
Server B-1

实现灰度的核心问题

根据既定的灰度策略来识别灰度流量

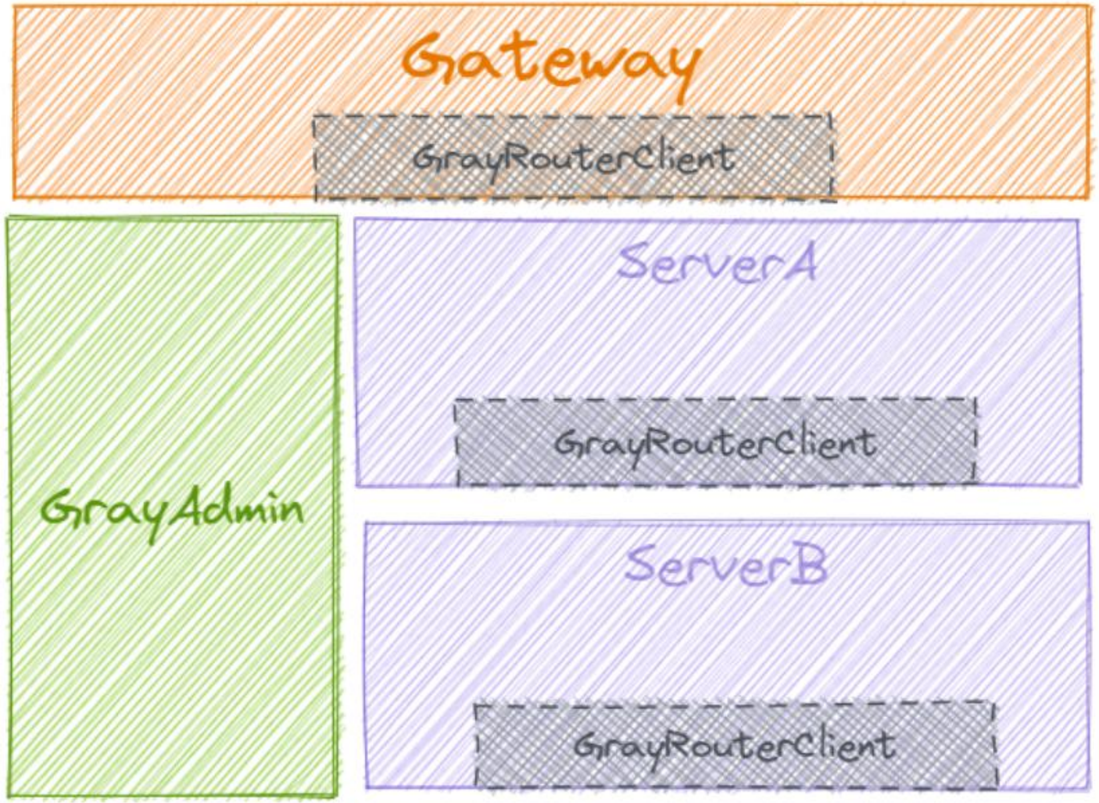
为灰度流量打上标识，方便下游服务识别

如何将灰度流量传递至下游服务

如何将灰度流量准确的转发到服务实例上

灰度方案实现

1. 灰度方案架构



- GrayAdmin：负责灰度策略的管控，主要包括灰度规则的发布、回滚，灰度记录查询。

- GrayRouterClient：灰度路由客户端组件，主要分为网关端客户端以及服务端客户端。主要负责灰度流量的识别以及路由转发，相比于网关端的客户端而言，服务端的客户端要额外提供客户端元数据注册的功能。

灰度方案实现

2. 流量识别 & 流量标记

```java
public Mono<Void> filter(ServerWebExchange exchange, GatewayFilterChain chain) {
    //从token中解析用户名匹配灰度规则
    ServerHttpRequest request = exchange.getRequest();
    String authorization = request.getHeaders().getFirst( headerName: "Authorization");
    List<GrayRule> rules = HAZELCAST_INSTANCE.getList(GrayConstant.GRAY_RULE);
    boolean needGrayRoute = false;
    String targetVersion = "";
    for (GrayRule rule : rules) {
        if (rule.getRouteKey().equals("username")){
            if (rule.getMatchRule().equals("equals") && authorization.equals(rule.getMatchValue())){
                needGrayRoute = true;
                targetVersion = rule.getTargetVersion();
                break;
            } else if (rule.getMatchRule().equals("like") && authorization.startsWith(rule.getMatchValue())){
                needGrayRoute = true;
                targetVersion = rule.getTargetVersion();
                break;
            }
        }else if (rule.getMatchRule().equals("all")){
            needGrayRoute = true;
            targetVersion = rule.getTargetVersion();
            break;
        }
    }
    if (needGrayRoute){
        log.info(String.format("request need router ! gray version : %s", targetVersion));
        GrayContextHolder.putContext(targetVersion);
        // request请求头不支持直接修改 需要通过mutate修改，ServerWebExchange也是如此
        exchange = exchange.mutate().request(request.mutate().header(GrayConstant.GRAY_HEAD, targetVersion).build()).build();
    }
    return chain.filter(exchange);
}
```

获取GrayAdmin所发布的灰度规则

匹配灰度规则，判断当前请求是否需要灰度

灰度流量标记

网关层灰度客户端

- **灰度规则监听：**可基于Zookeeper的监听机制实现GrayAdmin灰度规则的监听。也可基于Etcd、消息中间件、分布式缓存（Redis,Hazelcast）来实现。
- **流量识别：**基于灰度规则来判断当前请求是否是灰度请求。本方案主要从用户层来进行灰度。
- **流量标记：**识别灰度流量后，需要对流量打上Tag，保存在ThreadLocal中，以便在服务路由的时候进行转发。

灰度方案实现

3. 流量透传

- **网关服务流量透传：**网关服务在识别灰度流量后需要将打上标记的灰度流量透传至下游服务，方便下游服务在发起服务调用的时候进行服务路由。

```
if (needGrayRoute){
    log.info(String.format("request need router ! gray version : %s", targetVersion));
    GrayContextHolder.putContext(targetVersion);
    // request请求头不支持直接修改 需要通过mutate修改，ServerWebExchange也是如此
    exchange = exchange.mutate().request(request.mutate().header(GrayConstant.GRAY_HEAD, targetVersion).build()).build();
}
return chain.filter(exchange);
```

```
@Configuration
public class OpenFeignRequestInterceptor implements RequestInterceptor {

    @Override
    public void apply(RequestTemplate requestTemplate) {
        ServletRequestAttributes servletRequestAttributes = (ServletRequestAttributes) RequestContextHolder.getRequestAttributes();
        if (null != servletRequestAttributes){
            HttpServletRequest request = servletRequestAttributes.getRequest();
            Enumeration<String> headerNames = request.getHeaderNames();
            while (headerNames.hasMoreElements()){
                String headName = headerNames.nextElement();
                String headValue = request.getHeader(headName);
                requestTemplate.header(headName, headValue);
                if (GrayConstant.GRAY_HEAD.equals(headName)){
                    GrayContextHolder.putContext(headValue);
                }
            }
        }
    }
}
```

- **OpenFeign服务调用流量透传：**
  基于FeignClient的内部服务调用在开启熔断后需要将请求头信息Copy至，Hystrx调用线程中，要不后续下游服务无法获取到灰度Tag而无法识别流量并进行正确的路由转发。

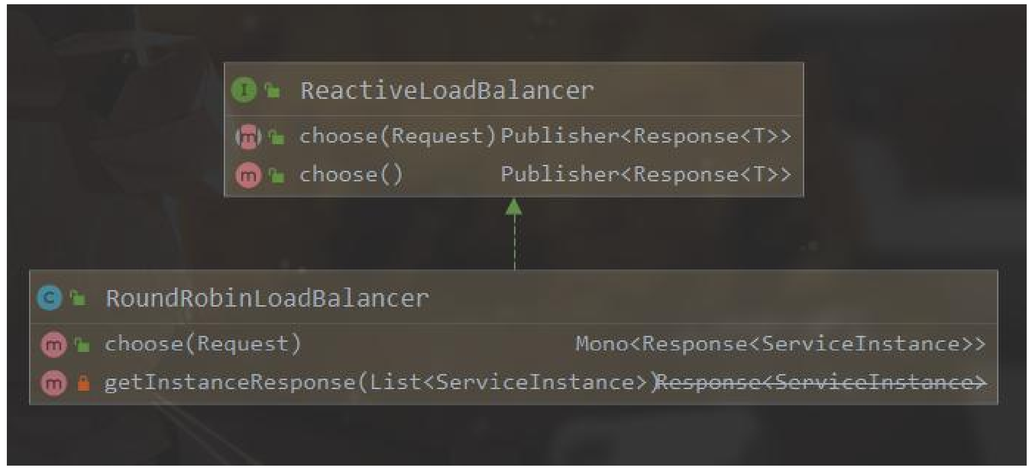
灰度方案实现

4. 流量路由-负载均衡算法

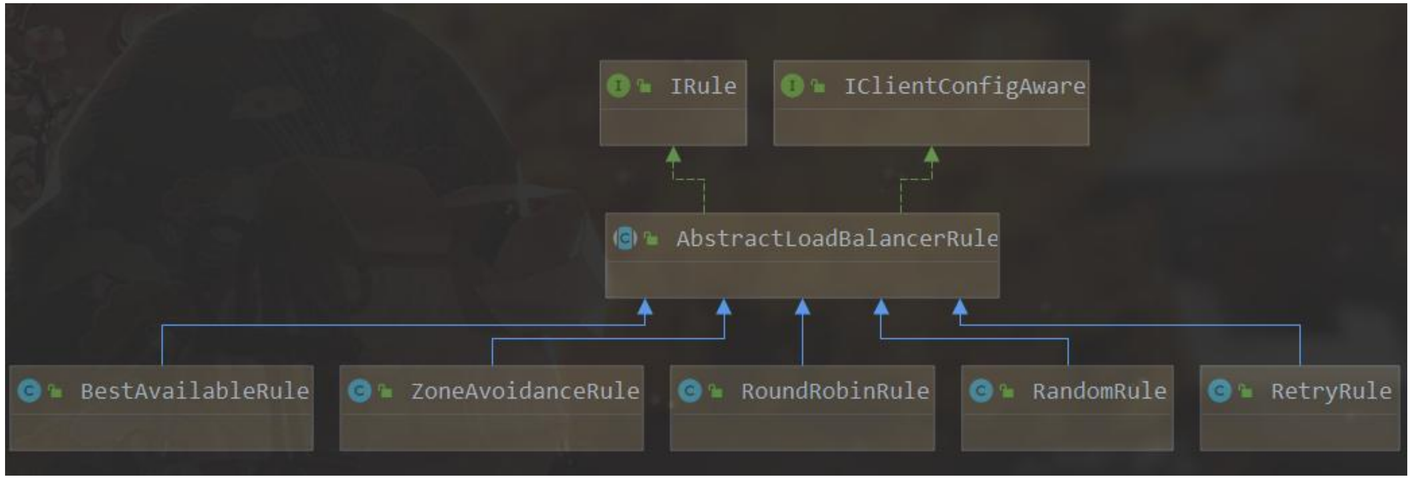
SpringCloudGateway中默认提供了两种负载均衡实现:

- 一种基于LoadBalancerClientFilter通过其中维护的 LoadBalancerClient 调用最终的负载均衡算法(Netflix实现)
- 另一种基于ReactiveLoadBalancerClilentFilter通过其中 ReactorLoadBalancer 调用最终的负载均衡算法(SpringCloud官方实现)

SpringCloud负载均衡实现



Netflix负载均衡实现



灰度方案实现

4. 流量路由-客户端路由

```java
@Override
public Server choose(Object o) {
    Server chooseServer = null;
    try {
        //从ThreadLocal中获取灰度标记
        String grayTag = GrayContextHolder.getContext();
        //获取所有可用服务
        List<Server> serverList = this.getLoadBalancer().getReachableServers();
        //灰度发布的服务
        List<Server> grayServerList = new ArrayList<>();
        for(Server server : serverList) {
            NacosServer nacosServer = (NacosServer) server;
            //从nacos中获取元素剧进行匹配
            if(nacosServer.getMetadata().containsKey(GrayConstant.SERVER_VERSION) && nacosServer.getMetadata().get(GrayConstant.SERVER_VERSION).equals(grayTag)) {
                grayServerList.add(server);
            }
        }
        //如果被标记为灰度发布，则调用灰度发布的服务
        if(StringUtils.isNotEmpty(grayTag)) {
            if (!CollectionUtils.isEmpty(grayServerList)){
                chooseServer = grayServerList.get(grayPosition.getAndIncrement() % grayServerList.size());
            }
        } else {
            chooseServer = serverList.get(position.getAndIncrement() % serverList.size());
        }
    } finally {
        //清除灰度标记
        GrayContextHolder.removeContext();
        return chooseServer;
    }
}
```

自定灰度路由实现：

网关服务灰度客户端以及服务会对客户端所使用的具体路由算法。

灰度方案实现

5. 客户端服务元数据注册

客户端服务启动时自动添加服务版本号元数据信息注册至注册中心

```java
@Configuration
@ConditionalOnNacosDiscoveryEnabled
@AutoConfigureBefore({SimpleDiscoveryClientAutoConfiguration.class, CommonsClientAutoConfiguration.class})
public class ServerMetaDataAutoConfiguration {

    @Value("${server.version:1.0.0}")
    private String version;

    @Bean
    @ConditionalOnMissingBean
    @ConditionalOnProperty(value = {"spring.cloud.nacos.discovery.watch.enabled"},matchIfMissing = true)
    public NacosWatch nacosWatch(NacosDiscoveryProperties nacosDiscoveryProperties, NacosServiceManager nacosServiceManager){
        nacosDiscoveryProperties.getMetadata().put("version", version);
        return new NacosWatch(nacosServiceManager,nacosDiscoveryProperties);
    }
}
```

方案缺陷

- 无法实现数据库的灰度
- 无法实现消息中间件的灰度

THANKS FOR WATCHING

感谢您的观看